CS 473: Undergraduate Algorithms, Fall 2013
Homework 0

Due Tuesday, September 3, 2013 at 12:30pm

Quiz 0 (on the course Moodle page)
is also due Tuesday, September 3, 2013 at noon.

- **Please carefully read the course policies on the course web site.** These policies may be different than other classes you have taken. (For example: No late anything ever; "I don't know" is worth 25%, but "Repeat this for all n" is an automatic zero; ***every*** homework question requires a proof; collaboration is allowed, but you must cite your collaborators.) If you have *any* questions, please ask in lecture, in headbanging, on Piazza, in office hours, or by email.

- Homework 0 and Quiz 0 test your familiarity with prerequisite material—big-Oh notation, elementary algorithms and data structures, recurrences, graphs, and most importantly, induction—to help you identify gaps in your background knowledge. **You are responsible for filling those gaps.** The course web page has pointers to several excellent online resources for prerequisite material. If you need help, please ask in headbanging, on Piazza, in office hours, or by email.

- ***Each student must submit individual solutions for these homework problems.*** You may use any source at your disposal—paper, electronic, or human—but you ***must*** cite ***every*** source that you use. For all future homeworks, groups of up to three students may submit joint solutions.

- **Submit your solutions on standard printer/copier paper, not notebook paper.** If you write your solutions by hand, please use the last three pages of this homework as a template. At the top of each page, please clearly print your name and NetID, and indicate your registered discussion section. Use both sides of the page. If you plan to typeset your homework, you can find a LaTeX template on the course web site; well-typeset homework will get a small amount of extra credit.

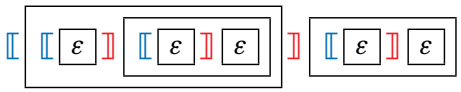
- Submit your solution to each numbered problem (stapled if necessary) in the corresponding drop box outside 1404 Siebel, ***or*** in the corresponding box in Siebel 1404 immediately before/after class. (This is the last homework we'll collect in 1404.) ***Do not staple your entire homework together.***

---

1. Consider the following recursively-defined sets of strings of left brackets ⟦ and right brackets ⟧:

   - A string x is ***balanced*** if it satisfies one of the following conditions:
     - x is the empty string, or
     - $x = ⟦\, y \,⟧\, z$, where y and z are balanced strings.

     For example, the following diagram shows that the string ⟦ ⟦ ⟧ ⟦ ⟧ ⟧ ⟦ ⟧ is balanced. Each boxed substring is balanced, and ε is the empty string.

   - A string x is ***erasable*** if it satisfies one of two conditions:
     - x is the empty string, or
     - $x = y ⟦\, ⟧\, z$, where yz is an erasable string.

     For example, we can prove that the string ⟦ ⟦ ⟧ ⟦ ⟧ ⟧ ⟦ ⟧ is erasable as follows:

     $$⟦\,⟦\,⟧\,\cancel{⟦\,⟧}\,⟧\,⟦\,⟧ \;\to\; ⟦\,\cancel{⟦\,⟧}\,⟧\,⟦\,⟧ \;\to\; ⟦\,⟧\,\cancel{⟦\,⟧} \;\to\; \cancel{⟦\,⟧} \;\to\; \varepsilon$$

   Your task is to prove that these two definitions are equivalent.

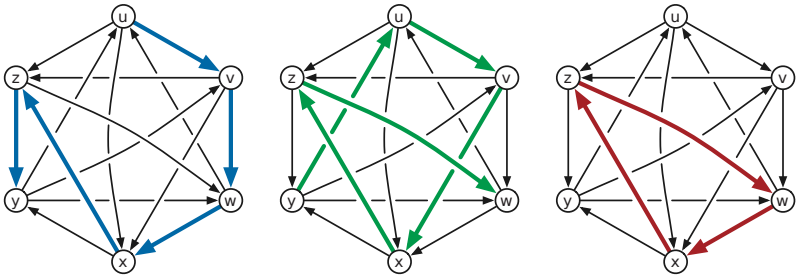
   (a) Prove that every balanced string is erasable.

   (b) Prove that every erasable string is balanced.

2. A ***tournament*** is a directed graph with exactly one directed edge between each pair of vertices. That is, for any vertices v and w, a tournament contains either an edge $v \to w$ or an edge $w \to v$, but not both. A ***Hamiltonian path*** in a directed graph G is a directed path that visits every vertex of G exactly once.

   (a) Prove that every tournament contains a Hamiltonian path.

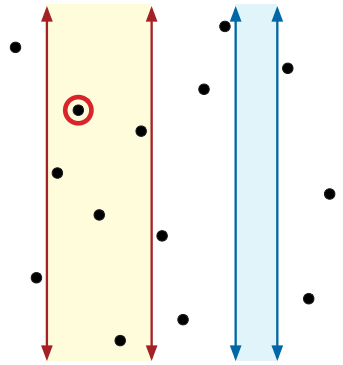
   (b) Prove that every tournament contains either *exactly one* Hamiltonian path or a directed cycle of length three.



A tournament with two Hamiltonian paths $u \to v \to w \to x \to z \to y$ and $y \to u \to v \to x \to z \to w$ and a directed triangle $w \to x \to z \to w$.

3. Suppose you are given a set $P = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ of n points in the plane with distinct x- and y-coordinates. Describe a data structure that can answer the following query as quickly as possible:

   > Given two numbers l and r, find the highest point in P inside the vertical slab $l < x < r$. More formally, find the point $(x_i, y_i) \in P$ such that $l < x_i < r$ and y_i is as large as possible. Return NONE if the slab does not contain any points in P.



A query with the left slab returns the indicated point.
A query with the right slab returns NONE.

To receive full credit, your solution must include (a) a concise description of your data structure, (b) a concise description of your query algorithm, (c) a proof that your query algorithm is correct, (d) a bound on the size of your data structure, and (e) a bound on the running time of your query algorithm. You do ***not*** need to describe or analyze an algorithm to construct your data structure.

Smaller data structures and faster query times are worth more points.

Starting with this homework, groups of up to three students may submit a single solution for each numbered problem. Every student in the group receives the same grade. Groups can be different for different problems.

---

1. Consider the following cruel and unusual sorting algorithm.

```
CRUEL(A[1..n]):
  if n > 1
      CRUEL(A[1..n/2])
      CRUEL(A[n/2+1..n])
      UNUSUAL(A[1..n])
```

```
UNUSUAL(A[1..n]):
  if n = 2
      if A[1] > A[2]                      ⟨⟨the only comparison!⟩⟩
          swap A[1] ↔ A[2]
  else
      for i ← 1 to n/4                    ⟨⟨swap 2nd and 3rd quarters⟩⟩
          swap A[i + n/4] ↔ A[i + n/2]
      UNUSUAL(A[1..n/2])                  ⟨⟨recurse on left half⟩⟩
      UNUSUAL(A[n/2+1..n])                ⟨⟨recurse on right half⟩⟩
      UNUSUAL(A[n/4+1..3n/4])             ⟨⟨recurse on middle half⟩⟩
```

Notice that the comparisons performed by the algorithm do not depend at all on the values in the input array; such a sorting algorithm is called ***oblivious***. Assume for this problem that the input size n is always a power of 2.

   (a) Prove that CRUEL correctly sorts any input array. *[Hint: Consider an array that contains $n/4$ 1s, $n/4$ 2s, $n/4$ 3s, and $n/4$ 4s. Why is considering this special case enough? What does* UNUSUAL *actually do?]*

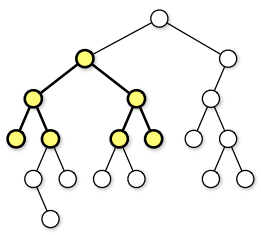
   (b) Prove that CRUEL would *not* always sort correctly if we removed the for-loop from UNUSUAL.

   (c) Prove that CRUEL would *not* always sort correctly if we swapped the last two lines of UNUSUAL.

   (d) What is the running time of UNUSUAL? Justify your answer.

   (e) What is the running time of CRUEL? Justify your answer.

2. For this problem, a *subtree* of a binary tree means any connected subgraph. A binary tree is *complete* if every internal node has two children, and every leaf has exactly the same depth. Describe and analyze a recursive algorithm to compute the *largest complete subtree* of a given binary tree. Your algorithm should return the root and the depth of this subtree.



The largest complete subtree of this binary tree has depth 2.

3. (a) Suppose we are given two sorted arrays $A[1..n]$ and $B[1..n]$. Describe an algorithm to find the median of the union of A and B in $O(\log n)$ time. Assume the arrays contain no duplicate elements.

   (b) Now suppose we are given *three* sorted arrays $A[1..n]$, $B[1..n]$, and $C[1..n]$. Describe an algorithm to find the median element of $A \cup B \cup C$ in $O(\log n)$ time.

*4. ***Extra credit; due September 17.*** (The "I don't know" rule does not apply to extra credit problems.)

Bob Ratenbur, a new student in CS 225, is trying to write code to perform preorder, inorder, and postorder traversal of binary trees. Bob understands the basic idea behind the traversal algorithms, but whenever he tries to implement them, he keeps mixing up the recursive calls. Five minutes before the deadline, Bob submitted code with the following structure:

```
PreOrder(v):
  if v = Null
    return
  else
    print label(v)
    [redacted]Order(left(v))
    [redacted]Order(right(v))
```

```
InOrder(v):
  if v = Null
    return
  else
    [redacted]Order(left(v))
    print label(v)
    [redacted]Order(right(v))
```

```
PostOrder(v):
  if v = Null
    return
  else
    [redacted]Order(left(v))
    [redacted]Order(right(v))
    print label(v)
```

Each [redacted] represents either Pre, In, or Post. Moreover, each of the following function calls appears exactly once in Bob's submitted code:

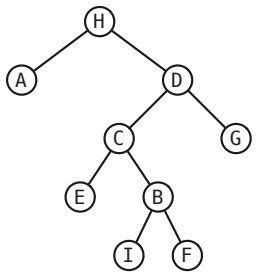
PreOrder(*left*(*v*))	InOrder(*left*(*v*))	PostOrder(*left*(*v*))
PreOrder(*right*(*v*))	InOrder(*right*(*v*))	PostOrder(*right*(*v*))

Thus, there are exactly 36 possibilities for Bob's code. Unfortunately, Bob accidentally deleted his source code after submitting the executable, so neither you nor he knows which functions were called where.

Your task is to reconstruct a binary tree T from the output of Bob's traversal algorithms, which has been helpfully parsed into three arrays $Pre[1..n]$, $In[1..n]$, and $Post[1..n]$. Your algorithm should return the unknown tree T. You may assume that the vertex labels of the unknown tree are distinct, and that every internal node has exactly two children. For example, given the input

$$Pre[1..n] = [\texttt{H A E C B I F G D}]$$
$$In[1..n] = [\texttt{A H D C E I F B G}]$$
$$Post[1..n] = [\texttt{A E I B F C D G H}]$$

your algorithm should return the following tree:



In general, the traversal sequences may not give you enough information to reconstruct Bob's code; however, to produce the example sequences above, Bob's code must look like this:

```
PreOrder(v):
  if v = Null
      return
  else
      print label(v)
      PreOrder(left(v))
      PostOrder(right(v))
```

```
InOrder(v):
  if v = Null
      return
  else
      PostOrder(left(v))
      print label(v)
      PreOrder(right(v))
```

```
PostOrder(v):
  if v = Null
      return
  else
      InOrder(left(v))
      InOrder(right(v))
      print label(v)
```

1. Suppose we are given an array $A[1..n]$ of integers, some positive and negative, which we are asked to partition into contiguous subarrays, which we call ***chunks***. The *value* of any chunk is the *square* of the sum of elements in that chunk; the value of a partition of A is the sum of the values of its chunks.

   For example, suppose $A = [3, -1, 4, -1, 5, -9]$. The partition $[3, -1, 4], [-1, 5], [-9]$ has three chunks with total value $(3 - 1 + 4)^2 + (-1 + 5)^2 + (-9)^2 = 6^2 + 4^2 + 9^2 = 133$, while the partition $[3, -1], [4, -1, 5, -9]$ has two chunks with total value $(3 - 1)^2 + (4 - 1 + 5 - 9)^2 = 5$.

   (a) Describe and analyze an algorithm that computes the minimum-value partition of a given array of n numbers.

   (b) Now suppose we also given an integer $k > 0$. Describe and analyze an algorithm that computes the minimum-value partition of a given array of n numbers ***into at most k chunks***.

2. Consider the following solitaire form of Scrabble. We begin with a fixed, finite sequence of tiles; each tile contains a letter and a numerical value. At the start of the game, we draw the seven tiles from the sequence and put them into our hand. In each turn, we form an English word from some or all of the tiles in our hand, place those tiles on the table, and receive the total value of those tiles as points. If no English word can be formed from the tiles in our hand, the game immediately ends. Then we repeatedly draw the next tile from the start of the sequence until either (a) we have seven tiles in our hand, or (b) the sequence is empty. (Sorry, no double/triple word/letter scores, bingos, blanks, or passing.) Our goal is to obtain as many points as possible.

   For example, suppose we are given the tile sequence

   $$\boxed{I_2}\boxed{N_2}\boxed{X_8}\boxed{A_1}\boxed{N_2}\boxed{A_1}\boxed{D_3}\boxed{U_5}\boxed{D_3}\boxed{I_2}\boxed{D_3}\boxed{K_8}\boxed{U_5}\boxed{B_4}\boxed{L_2}\boxed{A_1}\boxed{K_8}\boxed{H_5}\boxed{A_1}\boxed{N_2}.$$

   Then we can earn 68 points as follows:

   - We initially draw $\boxed{I_2}\boxed{N_2}\boxed{X_8}\boxed{A_1}\boxed{N_2}\boxed{A_1}\boxed{D_3}$.
   - Play the word $\boxed{N_2}\boxed{A_1}\boxed{I_2}\boxed{A_1}\boxed{D_3}$ for 9 points, leaving $\boxed{N_2}\boxed{X_8}$ in our hand.
   - Draw the next five tiles $\boxed{U_5}\boxed{D_3}\boxed{I_2}\boxed{D_3}\boxed{K_8}$.
   - Play the word $\boxed{U_5}\boxed{N_2}\boxed{D_3}\boxed{I_2}\boxed{D_3}$ for 15 points, leaving $\boxed{K_8}\boxed{X_8}$ in our hand.
   - Draw the next five tiles $\boxed{U_5}\boxed{B_4}\boxed{L_2}\boxed{A_1}\boxed{K_8}$.
   - Play the word $\boxed{B_4}\boxed{U_5}\boxed{L_2}\boxed{K_8}$ for 19 points, leaving $\boxed{K_8}\boxed{X_8}\boxed{A_1}$ in our hand.
   - Draw the next three tiles $\boxed{H_5}\boxed{A_1}\boxed{N_2}$, emptying the list.
   - Play the word $\boxed{A_1}\boxed{N_2}\boxed{K_8}\boxed{H_5}$ for 16 points, leaving $\boxed{X_8}\boxed{A_1}$ in our hand.
   - Play the word $\boxed{A_1}\boxed{X_8}$ for 9 points, emptying our hand and ending the game.

   Design and analyze an algorithm to compute the maximum number of points that can be earned from a given sequence of tiles. ***The input consists of two arrays*** $\mathrm{Letter}[1..n]$***, containing a sequence of letters between*** `A` ***and*** `Z`***, and*** $\mathrm{Value}[\texttt{A}..\texttt{Z}]$***, where*** $\mathrm{Value}[i]$ ***is the value of letter*** i***.*** The output is a single number. Assume that you can find all English words that can be made from any seven tiles, along with the point values of those words, in $O(1)$ time.

3. ***Extra credit.*** Submit your answer to Homework 1 problem 4.

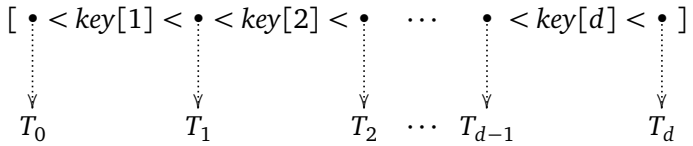
1. A standard method to improve the cache performance of search trees is to pack more search keys and subtrees into each node. A ***B-tree*** is a rooted tree in which each internal node stores up to B keys and pointers to up to $B+1$ children, each the root of a smaller B-tree. Specifically each node v stores three fields:

   - a positive integer $v.d \leq B$,
   - a *sorted* array $v.key[1..v.d]$, and
   - an array $v.child[0..v.d]$ of child pointers.

   In particular, the number of child pointers is always exactly one more than the number of keys.

   Each pointer $v.child[i]$ is either NULL or a pointer to the root of a B-tree whose keys are all larger than $v.key[i]$ and smaller than $v.key[i+1]$. In particular, all keys in the leftmost subtree $v.child[0]$ are smaller than $v.key[1]$, and all keys in the rightmost subtree $v.child[v.d]$ are larger than $v.key[v.d]$.

   Intuitively, you should have the following picture in mind:

   Here T_i is the subtree pointed to by $child[i]$.

   The ***cost*** of searching for a key x in a B-tree is the number of nodes in the path from the root to the node containing x as one of its keys. A 1-tree is just a standard binary search tree.

   Fix an arbitrary positive integer $B > 0$. (I suggest $B = 8$.) Suppose we are given a sorted array $A[1,\ldots,n]$ of search keys and a corresponding array $F[1,\ldots,n]$ of frequency counts, where $F[i]$ is the number of times that we will search for $A[i]$.

   Describe and analyze an efficient algorithm to find a B-tree that minimizes the total cost of searching for n keys with a given array of frequencies.

   - For 5 points, describe a polynomial-time algorithm for the special case $B = 2$.
   - For 10 points, describe an algorithm for arbitrary B that runs in $O(n^{B+c})$ time for some fixed integer c.
   - For 15 points, describe an algorithm for arbitrary B that runs in $O(n^c)$ time for some fixed integer c that does *not* depend on B.

   Like all other homework problems, 10 points is full credit; any points above 10 will be awarded as extra credit.

A few comments about *B*-trees. Normally, B-trees are required to satisfy two additional constraints, which guarantee a worst-case search cost of $O(\log_B n)$: Every leaf must have exactly the same depth, and every node except possibly the root must contain at least $B/2$ keys. However, in this problem, we are not interested in optimizing the *worst-case* search cost, but rather the *total* cost of a sequence of searches, so we will not impose these additional constraints.

In most large database systems, the parameter B is chosen so that each node exactly fits in a cache line. Since the entire cache line is loaded into cache anyway, and the cost of loading a cache

line exceeds the cost of searching within the cache, the running time is dominated by the number of cache faults. This effect is even more noticeable if the data is too big to lie in RAM at all; then the cost is dominated by the number of page faults, and B should be roughly the size of a page.

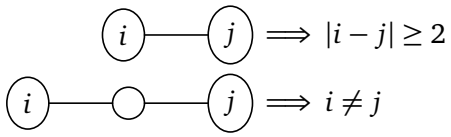
Finally, don't worry about the cache/disk performance in your homework solutions; just analyze the CPU time as usual. Designing algorithms with few cache misses or page faults is a interesting pastime; simultaneously optimizing CPU time *and* cache misses *and* page faults is even more interesting. But this kind of design and analysis requires tools we won't see in this class.

2. ***Extra credit***, because we screwed up the first version.

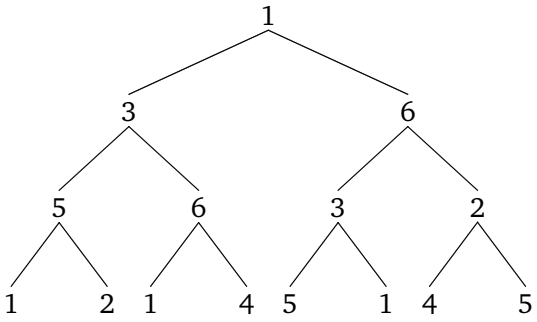
A ***k-2 coloring*** of a tree assigns each vertex a value from the set $\{1, 2, \ldots, k\}$, called its *color*, such that the following constraints are satisfied:

- A node and its parent cannot have the same color or adjacent colors.
- A node and its grandparent cannot have the same color.
- Two nodes with the same parent cannot have the same color.

The last two rules can be written more simply as "Two nodes that are two edges apart cannot have the same color." Diagrammatically, if we write the names of the colors inside the vertices,



For example, here is a valid 6-2 coloring of the complete binary tree with depth 3:



(a) Describe and analyze an algorithm that computes a 6-2 coloring of a given binary tree. The existence of such an algorithm proves that every binary tree has a 6-2 coloring.

(b) Prove that not every binary tree has a 5-2 coloring.

(c) A *ternary* tree is a rooted tree where every node has at most *three* children. What is the smallest integer k such that every ternary tree has a k-2 coloring? Prove your answer is correct.

1. A *meldable priority queue* stores a set of values, called *priorities*, from some totally-ordered universe (such as the integers) and supports the following operations:

   - MAKEQUEUE: Return a new priority queue containing the empty set.
   - FINDMIN(Q): Return the smallest element of Q (if any).
   - DELETEMIN(Q): Remove the smallest element in Q (if any).
   - INSERT(Q, x): Insert priority x into Q, if it is not already there.
   - DECREASE(Q, x, y): Replace some element $x \in Q$ with a smaller priority y. (If $y > x$, the operation fails.) The input is a pointer directly to the node in Q containing x.
   - DELETE(Q, x): Delete the priority $x \in Q$. The input is a pointer directly to the node in Q containing x.
   - MELD(Q_1, Q_2): Return a new priority queue containing all the elements of Q_1 and Q_2; this operation destroys Q_1 and Q_2.

   A simple way to implement such a data structure is to use a ***heap-ordered binary tree*** — each node stores a priority, which is smaller than the priorities of its children, along with pointers to its parent and at most two children. MELD can be implemented using the following randomized algorithm:

```
MELD(Q1, Q2):
  if Q1 is empty return Q2
  if Q2 is empty return Q1

  if priority(Q1) > priority(Q2)
      swap Q1 ↔ Q2

  with probability 1/2
      left(Q1) ← MELD(left(Q1), Q2)
  else
      right(Q1) ← MELD(right(Q1), Q2)

  return Q1
```

   (a) Prove that for *any* heap-ordered binary trees Q_1 and Q_2 (***not*** just those constructed by the operations listed above), the expected running time of MELD(Q_1, Q_2) is $O(\log n)$, where $n = |Q_1| + |Q_2|$. *[Hint: How long is a random root-to-leaf path in an n-node binary tree if each left/right choice is made uniformly and independently at random?]*

   (b) Show that each of the other meldable priority queue operations can be implemented with at most one call to MELD and $O(1)$ additional time. (This implies that every operation takes $O(\log n)$ expected time.)

2. Recall that a *priority search tree* is a binary tree in which every node has both a *search key* and a *priority*, arranged so that the tree is simultaneously a binary search tree for the keys and a min-heap for the priorities. A *treap* is a priority search tree whose search keys are given by the user and whose priorities are independent random numbers.

   A ***heater*** is a priority search tree whose *priorities* are given by the user and whose *search keys* are distributed uniformly and independently at random in the real interval $[0, 1]$. Intuitively, a heater is a sort of anti-treap.[1]

[1] There are those who think that life has nothing left to chance, a host of holy horrors to direct our aimless dance.

The following problems consider an n-node heater T. We identify nodes in T by their *priority rank*; for example, "node 5" means the node in T with the 5th smallest priority. The min-heap property implies that node 1 is the root of T. You may assume all search keys and priorities are distinct. Finally, let i and j be arbitrary integers with $1 \le i < j \le n$.

(a) Prove that if we permute the set $\{1, 2, \ldots, n\}$ uniformly at random, integers i and j are adjacent with probability $2/n$.

(b) Prove that node i is an ancestor of node j with probability $2/(i+1)$. *[Hint: Use part (a)!]*

(c) What is the probability that node i is a *descendant* of node j? *[Hint: Don't use part (a)!]*

(d) What is the *exact* expected depth of node j?

(e) Describe and analyze an algorithm to insert a new item into an n-node heater.

(f) Describe and analyze an algorithm to delete the smallest priority (the root) from an n-node heater.

*3. ***Extra credit; due October 15.*** In the usual theoretical presentation of treaps, the priorities are random real numbers chosen uniformly from the interval $[0, 1]$. In practice, however, computers have access only to random *bits*. This problem asks you to analyze an implementation of treaps that takes this limitation into account.

Suppose the priority of a node v is abstractly represented as an infinite sequence $\pi_v[1..\infty]$ of random bits, which is interpreted as the rational number

$$\mathit{priority}(v) = \sum_{i=1}^{\infty} \pi_v[i] \cdot 2^{-i}.$$

However, only a finite number ℓ_v of these bits are actually known at any given time. When a node v is first created, *none* of the priority bits are known: $\ell_v = 0$. We generate (or "reveal") new random bits only when they are necessary to compare priorities. The following algorithm compares the priorities of any two nodes in $O(1)$ expected time:

```
LargerPriority(v, w):
  for i ← 1 to ∞
      if i > ℓ_v
          ℓ_v ← i;  π_v[i] ← RandomBit
      if i > ℓ_w
          ℓ_w ← i;  π_w[i] ← RandomBit
      if π_v[i] > π_w[i]
          return v
      else if π_v[i] < π_w[i]
          return w
```

Suppose we insert n items one at a time into an initially empty treap. Let $L = \sum_v \ell_v$ denote the total number of random bits generated by calls to LargerPriority during these insertions.

(a) Prove that $E[L] = \Theta(n)$.

(b) Prove that $E[\ell_v] = \Theta(1)$ for any node v. *[Hint: This is equivalent to part (a). Why?]*

(c) Prove that $E[\ell_{\text{root}}] = \Theta(\log n)$. *[Hint: Why doesn't this contradict part (b)?]*

1. Recall that a standard (FIFO) queue maintains a sequence of items subject to the following operations.

   - PUSH(x): Add item x to the end of the sequence.
   - PULL(): Remove and return the item at the beginning of the sequence.
   - SIZE(): Return the current number of items in the sequence.

   It is easy to implement a queue using a doubly-linked list, so that it uses $O(n)$ space (where n is the number of items in the queue) and the worst-case time for each of these operations is $O(1)$.

   Consider the following new operation, which removes every tenth element from the queue, starting at the beginning, in $\Theta(n)$ worst-case time.

```
DECIMATE():
  n ← SIZE()
  for i ← 0 to n − 1
      if i mod 10 = 0
          PULL()    ⟨⟨result discarded⟩⟩
      else
          PUSH(PULL())
```

   Prove that in any intermixed sequence of PUSH, PULL, and DECIMATE operations, the amortized cost of each operation is $O(1)$.

2. ***This problem is extra credit***, because the original problem statement had several confusing small errors. I believe these erors are corrected in the current revision.

   Deleting an item from an open-addressed hash table is not as straightforward as deleting from a chained hash table. The obvious method for deleting an item x simply empties the entry in the hash table that contains x. Unfortunately, the obvious method doesn't always work. (Part (a) of this question asks you to prove this.)

   Knuth proposed the following *lazy* deletion strategy. Every cell in the table stores both an *item* and a *label*; the possible labels are **EMPTY**, **FULL**, and **JUNK**. The DELETE operation marks cells as **JUNK** instead of actually erasing their contents. Then FIND pretends that **JUNK** cells are occupied, and INSERT pretends that **JUNK** cells are actually empty. In more detail:

```
FIND(H, x):
  for i ← 0 to m − 1
      j ← h_i(x)
      if H.label[j] = FULL and H.item[j] = x
          return j
      else if H.label[j] = EMPTY
          return NONE
```

```
INSERT(H, x):
  for i ← 0 to m − 1
      j ← h_i(x)
      if H.label[j] = FULL and H.item[j] = x
          return          ⟨⟨already there⟩⟩
      if H.label[j] ≠ FULL
          H.item[j] ← x
          H.label[j] ← FULL
          return
```

```
DELETE(H, x):
  j ← FIND(H, x)
  if j ≠ NONE
      H.label[j] ← JUNK
```

Lazy deletion is always *correct*, but it is only *efficient* if we don't perform too many deletions. The search time depends on the fraction of non-**EMPTY** cells, not on the number of actual items stored in the table; thus, even if the number of items stays small, the table may fill up with **JUNK** cells, causing unsuccessful searches to scan the entire table. Less significantly, the data structure may use significantly more space than necessary for the number of items it actually stores. To avoid both of these issues, we use the following rebuilding rules:

- After each INSERT operation, if less than $1/4$ of the cells are **EMPTY**, rebuild the hash table.
- After each DELETE operation, if less than $1/4$ of the cells are **FULL**, rebuild the hash table.

To rebuild the hash table, we allocate a new hash table whose size is twice the number of **FULL** cells (unless that number is smaller than some fixed constant), INSERT each item in a **FULL** cell in the old hash table into the new hash table, and then discard the old hash table, as follows:

```
REBUILD(H):
  count ← 0
  for j ← 0 to H.size − 1
      if H.label[j] = FULL
          count ← count + 1
  H′ ← new hash table of size max{2 · count, 32}
  for j ← 0 to H.size − 1
      if H.label[j] = FULL
          INSERT(H′, H.item[j])
  discard H
  return H′
```

Finally, here are your actual homework questions!

(a) Describe a ***small*** example where the "obvious" deletion algorithm is incorrect; that is, show that the hash table can reach a state where a search can return the wrong result. Assume collisions are resolved by linear probing.

(b) Suppose we use Knuth's lazy deletion strategy instead. Prove that after several INSERT and DELETE operations into a table of arbitrary size m, it is possible for a single item x to be stored in *almost half* of the table cells. (However, at most one of those cells can be labeled **FULL**.)

(c) For purposes of analysis,[1] suppose FIND and INSERT run in $O(1)$ time when at least $1/4$ of the table cells are **EMPTY**. Prove that in any intermixed sequence of INSERT and DELETE operations, using Knuth's lazy deletion strategy, the amortized time per operation is $O(1)$.

*3. ***Extra credit.*** Submit your answer to Homework 4 problem 3.

[1] In fact, FIND and INSERT run in $O(1)$ *expected* time when at least $1/4$ of the table cells are **EMPTY**, and therefore each INSERT and DELETE takes $O(1)$ *expected* amortized time. But probability doesn't play any role whatsoever in the amortized analysis, so we can safely ignore the word "expected".

1. Suppose we want to maintain an array $X[1..n]$ of bits, which are all initially subject to the following operations.

   - LOOKUP(i): Given an index i, return $X[i]$.
   - BLACKEN(i): Given an index $i < n$, set $X[i] \leftarrow 1$.
   - NEXTWHITE(i): Given an index i, return the smallest index $j \geq i$ such that $X[j] = 0$. (Because we never change $X[n]$, such an index always exists.)

   If we use the array $X[1..n]$, it is trivial to implement LOOKUP and BLACKEN in $O(1)$ time and NEXTWHITE in $O(n)$ time. But you can do better! Describe data structures that support LOOKUP in $O(1)$ worst-case time and the other two operations in the following time bounds. (We want a different data structure for each set of time bounds, not one data structure that satisfies all bounds simultaneously!)

   (a) The worst-case time for both BLACKEN and NEXTWHITE is $O(\log n)$.

   (b) The amortized time for both BLACKEN and NEXTWHITE is $O(\log n)$. In addition, the *worst-case* time for BLACKEN is $O(1)$.

   (c) The amortized time for BLACKEN is $O(\log n)$, and the worst-case time for NEXTWHITE is $O(1)$.

   (d) The worst-case time for BLACKEN is $O(1)$, and the amortized time for NEXTWHITE is $O(\alpha(n))$. *[Hint: There is no* WHITEN*.]*

2. Recall that a standard (FIFO) queue maintains a sequence of items subject to the following operations:

   - PUSH(x): Add item x to the back of the queue (the end of the sequence).
   - PULL(): Remove and return the item at the front of the queue (the beginning of the sequence).

   It is easy to implement a queue using a doubly-linked list and a counter, using $O(n)$ space altogether, so that each PUSH or PULL requires $O(1)$ time.

   (a) Now suppose we want to support the following operation instead of PULL:

   - MULTIPULL(k): Remove the first k items from the front of the queue, and return the kth item removed.

   Suppose further that we implement MULTIPULL using the obvious algorithm:

```
MULTIPULL(k):
  for i ← 1 to k
      x ← PULL()
  return x
```

   Prove that in any intermixed sequence of PUSH and MULTIPULL operations, starting with an empty queue, the amortized cost of each operation is $O(1)$. You may assume that k is never larger than the number of items in the queue.

   (b) Now suppose we ***also*** want to support the following operation instead of PUSH:

   - MULTIPUSH(x, k): Insert k copies of x into the back of the queue.

   Suppose further that we implement MULTIPUSH using the obvious algorithm:

```
MULTIPUSH(k, x):
  for i ← 1 to k
    PUSH(x)
```

Prove that for any integers ℓ and n, there is a sequence of ℓ MULTIPUSH and MULTIPULL operations that require $\Omega(n\ell)$ time, where n is the maximum number of items in the queue at any time. Such a sequence implies that the amortized cost of each operation is $\Omega(n)$.

(c) Finally, describe a data structure that supports arbitrary intermixed sequences of MULTIPUSH and MULTIPULL operations in $O(1)$ amortized cost per operation. Like a standard queue, your data structure must use only $O(1)$ space per item. *[Hint: **Don't** use the obvious algorithms!]*

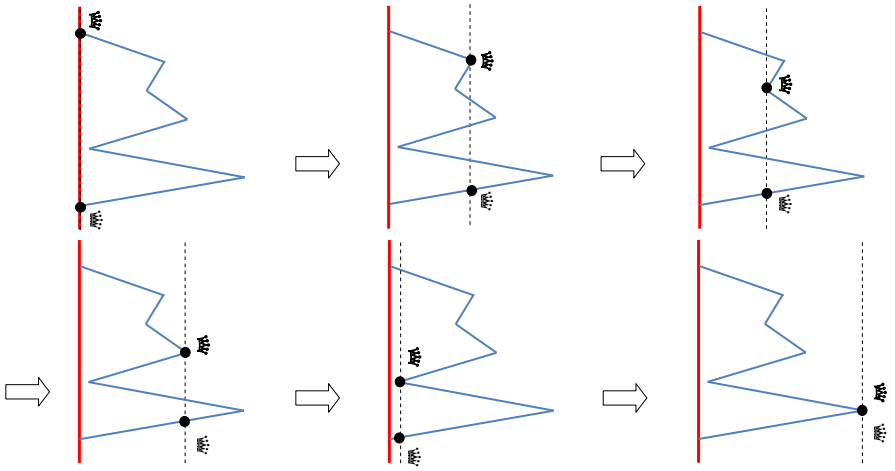
3. In every cheesy romance movie there's always that scene where the romantic couple, physically separated and looking for one another, suddenly matches eyes and then slowly approach one another with unwavering eye contact as the music rolls and in and the rain lifts and the sun shines through the clouds and kittens and puppies. . . .

Suppose a romantic couple—in grand computer science tradition, named Alice and Bob—enters a park from the northwest and southwest corners of the park, locked in dramatic eye contact. However, they can't just walk to one another in a straight line, because the paths of the park zig-zag between the northwest and southwest entrances. Instead, Alice and Bob must traverse the zig-zagging path so that their eyes are always locked perfectly in vertical eye-contact; thus, their x-coordinates must always be identical.

We can describe the zigzag path as an array $P[0..n]$ of points, which are the corners of the path in order from the southwest endpoint to the northwest endpoint, satisfying the following conditions:

- $P[i].y > P[i-1].y$ for every index i. That is, the path always moves upward.
- $P[0].x = P[n].x = 0$, and $P[i].x > 0$ for every index $1 \le i \le n-1$. Thus, the ends of the path are further to the left then any other point on the path.

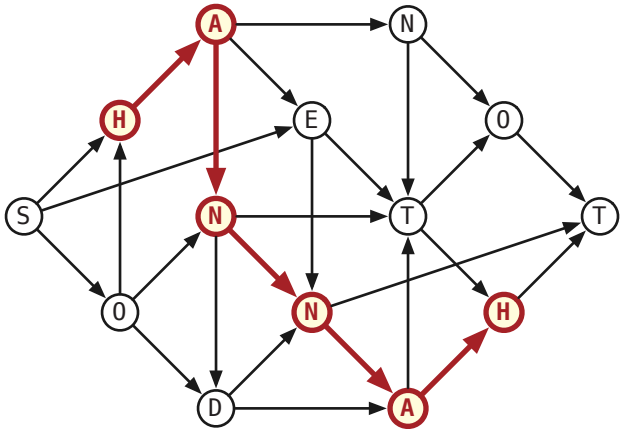
Prove that Alice and Bob can *always* meet.[1] *[Hint: Describe a graph that models all possible locations of the couple along the path. What are the vertices of this graph? What are the edges? What can we say about the degrees of the vertices?]*



[1] It follows that every cheesy romance movie (that reaches this scene) must have a happy, sappy ending.

1. Suppose we are given a directed acyclic graph G with labeled vertices. Every path in G has a label, which is a string obtained by concatenating the labels of its vertices in order. Recall that a *palindrome* is a string that is equal to its reversal.

   Describe and analyze an algorithm to find the length of the longest palindrome that is the label of a path in G. For example, given the graph below, your algorithm should return the integer 6, which is the length of the palindrome **HANNAH**.



2. Let G be a connected directed graph that contains both directions of every edge; that is, if $u \to v$ is an edge in G, its reversal $v \to u$ is also an edge in G. Consider the following non-standard traversal algorithm.

```
SPAGHETTITRAVERSAL(G):
  for all vertices v in G
      unmark v
  for all edges u→v in G
      color u→v white
  s ← any vertex in G
  SPAGHETTI(s)
```

```
SPAGHETTI(v):
  mark v                                  ⟨⟨"visit v"⟩⟩
  if there is a white arc v→w
      if w is unmarked
          color w→v green
      color v→w red                       ⟨⟨"traverse v→w"⟩⟩
      SPAGHETTI(w)
  else if there is a green arc v→w
      color v→w red                       ⟨⟨"traverse v→w"⟩⟩
      SPAGHETTI(w)
  ⟨⟨else every arc v→w is red, so halt⟩⟩
```

   We informally say that this algorithm "visits" vertex v every time it marks v, and it "traverses" edge $v \to w$ when it colors that edge red. Unlike our standard graph-traversal algorithms, SPAGHETTI may (in fact, *will*) mark/visit each vertex more than once.

   The following series of exercises leads to a proof that SPAGHETTI traverses each directed edge of G exactly once. Most of the solutions are very short.

   (a) Prove that no directed edge in G is traversed more than once.

   (b) When the algorithm visits a vertex v for the kth time, exactly how many edges into v are red, and exactly how many edges out of v are red? *[Hint: Consider the starting vertex s separately from the other vertices.]*

(c) Prove each vertex v is visited at most $\deg(v)$ times, except the starting vertex s, which is visited at most $\deg(s)+1$ times. This claim immediately implies that SPAGHETTITRAVERSAL(G) terminates.

(d) Prove that when SPAGHETTITRAVERSAL(G) ends, the last visited vertex is the starting vertex s.

(e) For every vertex v that SPAGHETTITRAVERSAL(G) visits, prove that all edges incident to v (either in or out) are red when SPAGHETTITRAVERSAL(G) halts. *[Hint: Consider the vertices in the order that they are marked for the first time, starting with s, and prove the claim by induction.]*

(f) Prove that SPAGHETTITRAVERSAL(G) visits every vertex of G.

(g) Finally, prove that SPAGHETTITRAVERSAL(G) traverses every edge of G exactly once.

1. Let G be a directed graph with (possibly negative!) edge weights, and let s be an arbitrary vertex of G. Suppose every vertex $v \neq s$ stores a pointer $pred(v)$ to another vertex in G.

   Describe and analyze an algorithm to determine whether these predecessor pointers define a single-source shortest path tree rooted at s. Do ***not*** assume that the graph G has no negative cycles.

   [Hint: There is a similar problem in head-banging, where you're given distances instead of predecessor pointers.]

2. Let G be a directed graph with positive edge weights, and let s and t be an arbitrary vertices of G. Describe an algorithm to determine the *number* of different shortest paths in G from s to t. Assume that you can perform arbitrary arithmetic operations in $O(1)$ time. *[Hint: Which edges of G belong to shortest paths from s to t?]*

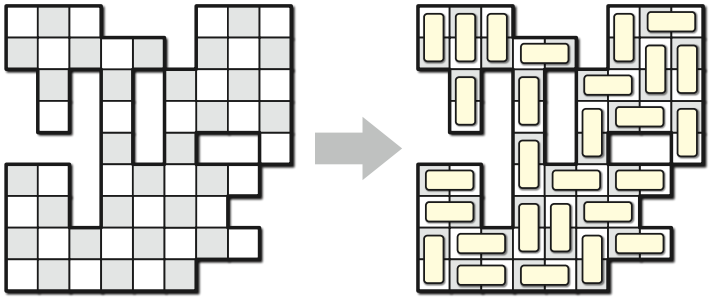
3. Describe and analyze and algorithm to find the second smallest spanning tree of a given undirected graph G with weighted edges, that is, the spanning tree of G with smallest total weight except for the minimum spanning tree.

1. You're organizing the First Annual UIUC Computer Science 72-Hour Dance Exchange, to be held all day Friday, Saturday, and Sunday. Several 30-minute sets of music will be played during the event, and a large number of DJs have applied to perform. You need to hire DJs according to the following constraints.

   - Exactly k sets of music must be played each day, and thus $3k$ sets altogether.
   - Each set must be played by a single DJ in a consistent music genre (ambient, bubblegum, dubstep, horrorcore, hyphy, trip-hop, Nitzhonot, Kwaito, J-pop, Nashville country, ...).
   - Each genre must be played at most once per day.
   - Each candidate DJ has given you a list of genres they are willing to play.
   - Each DJ can play at most three sets during the entire event.

   Suppose there are n candidate DJs and g different musical genres available. Describe and analyze an efficient algorithm that either assigns a DJ and a genre to each of the $3k$ sets, or correctly reports that no such assignment is possible.

2. Suppose you are given an $n \times n$ checkerboard with some of the squares deleted. You have a large set of dominos, just the right size to cover two squares of the checkerboard. Describe and analyze an algorithm to determine whether one can tile the board with dominos—each domino must cover exactly two undeleted squares, and each undeleted square must be covered by exactly one domino.



   Your input is a two-dimensional array $Deleted[1..n, 1..n]$ of bits, where $Deleted[i, j] = \text{TRUE}$ if and only if the square in row i and column j has been deleted. Your output is a single bit; you do ***not*** have to compute the actual placement of dominos. For example, for the board shown above, your algorithm should return TRUE.

3. Suppose we are given an array $A[1..m][1..n]$ of non-negative real numbers. We want to *round* A to an integer matrix, by replacing each entry x in A with either $\lfloor x \rfloor$ or $\lceil x \rceil$, without changing the sum of entries in any row or column of A. For example:

$$\begin{bmatrix} 1.2 & 3.4 & 2.4 \\ 3.9 & 4.0 & 2.1 \\ 7.9 & 1.6 & 0.5 \end{bmatrix} \longmapsto \begin{bmatrix} 1 & 4 & 2 \\ 4 & 4 & 2 \\ 8 & 1 & 1 \end{bmatrix}$$

   Describe and analyze an efficient algorithm that either rounds A in this fashion, or reports correctly that no such rounding is possible.

1. For any integer k, the problem k-COLOR asks whether the vertices of a given graph G can be colored using at most k colors so that neighboring vertices does not have the same color.

   (a) Prove that k-COLOR is NP-hard, for every integer $k \geq 3$.

   (b) Now fix an integer $k \geq 3$. Suppose you are given a magic black box that can determine **in polynomial time** whether an arbitrary graph is k-colorable; the box returns TRUE if the given graph is k-colorable and FALSE otherwise. The input to the magic black box is a graph. Just a graph. Vertices and edges. Nothing else.

   Describe and analyze a **polynomial-time** algorithm that either computes a proper k-coloring of a given graph G or correctly reports that no such coloring exists, using this magic black box as a subroutine.

2. A boolean formula is in *conjunctive normal form* (or *CNF*) if it consists of a *conjuction* (AND) or several *terms*, each of which is the disjunction (OR) of one or more literals. For example, the formula

$$(\overline{x} \vee y \vee \overline{z}) \wedge (y \vee z) \wedge (x \vee \overline{y} \vee \overline{z})$$

   is in conjunctive normal form. The problem ***CNF-SAT*** asks whether a boolean formula in conjunctive normal form is satisfiable. 3SAT is the special case of CNF-SAT where every clause in the input formula must have exactly three literals; it follows immediately that CNF-SAT is NP-hard.

   Symmetrically, a boolean formula is in *disjunctive normal form* (or *DNF*) if it consists of a *disjunction* (OR) or several *terms*, each of which is the conjunction (AND) of one or more literals. For example, the formula

$$(\overline{x} \wedge y \wedge \overline{z}) \vee (y \wedge z) \vee (x \wedge \overline{y} \wedge \overline{z})$$

   is in disjunctive normal form. The problem DNF-SAT asks whether a boolean formula in disjunctive normal form is satisfiable.

   (a) Describe a polynomial-time algorithm to solve DNF-SAT.

   (b) Describe a reduction from CNF-SAT to DNF-SAT.

   (c) Why do parts (a) and (b) not imply that P=NP?

3. The 42-PARTITION problem asks whether a given set S of n positive integers can be partitioned into subsets A and B (meaning $A \cup B = S$ and $A \cap B = \emptyset$) such that

$$\sum_{a \in A} a = 42 \sum_{b \in B} b$$

   For example, we can 42-partition the set $\{1, 2, 34, 40, 52\}$ into $A = \{34, 40, 52\}$ and $B = \{1, 2\}$, since $\sum A = 126 = 42 \cdot 3$ and $\sum B = 3$. But the set $\{4, 8, 15, 16, 23, 42\}$ cannot be 42-partitioned.

   (a) Prove that 42-PARTITION is NP-hard.

   (b) Let M denote the largest integer in the input set S. Describe an algorithm to solve 42-PARTITION in time polynomial in n and M. For example, your algorithm should return TRUE when $S = \{1, 2, 34, 40, 52\}$ and FALSE when $S = \{4, 8, 15, 16, 23, 42\}$.

   (c) Why do parts (a) and (b) not imply that P=NP?

CS 473: Undergraduate Algorithms, Fall 2013

Headbanging 0: Induction!

August 28 and 29

---

1. Prove that any non-negative integer can be represented as the sum of distinct powers of 2. ("Write it in binary" is not a proof; it's just a restatement of what you have to prove.)

2. Prove that every integer (positive, negative, or zero) can be written in the form $\sum_i \pm 3^i$, where the exponents i are distinct non-negative integers. For example:

$$42 = 3^4 - 3^3 - 3^2 - 3^1 \qquad 25 = 3^3 - 3^1 + 3^0 \qquad 17 = 3^3 - 3^2 - 3^0$$

3. Recall that a ***full binary tree*** is either an isolated *leaf*, or an *internal node* with a left subtree and a right subtree, each of which is a full binary tree. Equivalently, a binary tree is ***full*** if every internal node has exactly two children. Give at least *three different* proofs of the following fact: *In every full binary tree, the number of leaves is exactly one more than the number of internal nodes.*

Take-home points:

- Induction is recursion. Recursion is induction.
- All induction is strong/structural induction. There is absolutely no point in using a *weak* induction hypothesis. None. Ever.
- To prove that all snarks are boojums, start with an *arbitrary* snark and remove some tentacles. Do not start with a smaller snark and try to add tentacles. Snarks don't like that.
- Every induction proof requires an exhaustive case analysis. Write down the cases. Make sure they're exhaustive.
- Do the most general cases first. Whatever is left over are the base cases.
- The empty set is the best base case.

Khelm is Warsaw. Warsaw is Khelm. Khelm is Warsaw. Zay gezunt!
Warsaw is Khelm. Khelm is Warsaw. Warsaw is Khelm. For gezunt!

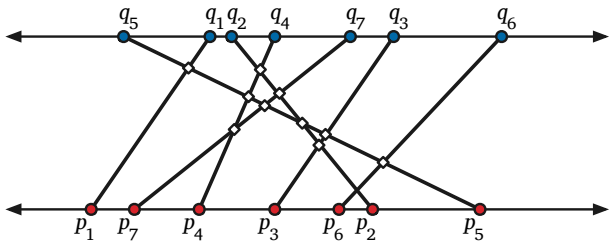
— Golem (feat. Amanda Palmer), "Warsaw is Khelm", *Fresh Off Boat* (2006)

1. An *inversion* in an array $A[1..n]$ is a pair of indices (i, j) such that $i < j$ and $A[i] > A[j]$. The number of inversions in an n-element array is between 0 (if the array is sorted) and $\binom{n}{2}$ (if the array is sorted backward).

   Describe and analyze a divide-and-conquer algorithm to count the number of inversions in an n-element array in $O(n \log n)$ time. Assume all the elements of the input array are distinct.

2. Suppose you are given two sets of n points, one set $\{p_1, p_2, \ldots, p_n\}$ on the line $y = 0$ and the other set $\{q_1, q_2, \ldots, q_n\}$ on the line $y = 1$. Create a set of n line segments by connect each point p_i to the corresponding point q_i. Describe and analyze a divide-and-conquer algorithm to determine how many pairs of these line segments intersect, in $O(n \log n)$ time. *[Hint: Use your solution to problem 1.]*

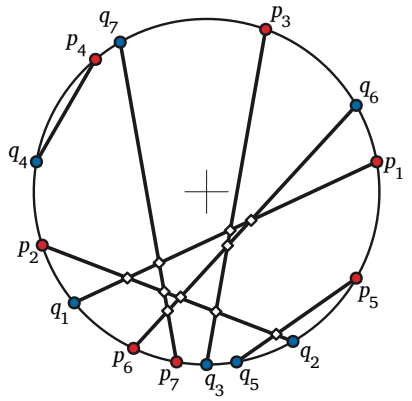
   Assume a reasonable representation for the input points, and assume the x-coordinates of the input points are distinct. For example, for the input shown below, your algorithm should return the number 10.



Ten intersecting pairs of segments with endpoints on parallel lines.

3. Now suppose you are given two sets $\{p_1, p_2, \ldots, p_n\}$ and $\{q_1, q_2, \ldots, q_n\}$ of n points *on the unit circle*. Connect each point p_i to the corresponding point q_i. Describe and analyze a divide-and-conquer algorithm to determine how many pairs of these line segments intersect in $O(n \log^2 n)$ time. *[Hint: Use your solution to problem 2.]*

   Assume a reasonable representation for the input points, and assume all input points are distinct. For example, for the input shown below, your algorithm should return the number 10.



Ten intersecting pairs of segments with endpoints on a circle.

4. **To think about later:** Solve problem 3 in $O(n \log n)$ time.

1. A *longest common subsequence* of a set of strings $\{A_i\}$ is a longest string that is a subsequence of A_i for each i. For example, `alrit` is a longest common subsequence of strings

   `algorithm` and `altruistic`.

   Given two strings $A[1..n]$ and $B[1..n]$, describe and analyze a dynamic programming algorithm that computes the length of a longest common subsequence of the two strings in $O(n^2)$ time.

2. Describe and analyze a dynamic programming algorithm that computes the length of a longest common subsequence of three strings $A[1..n]$, $B[1..n]$, and $C[1..n]$ in $O(n^3)$ time. *[Hint: Try **not** to use your solution to problem 1 directly.]*

3. A *lucky-10 number* is a string $D[1..n]$ of digits from 1 to 9 (no zeros), such that the i-th digit and the last i-th digit sum up to 10; in another words, $D[i] + D[n - i + 1] = 10$ for all i. For example,

   `3141592648159697` and `11599`

   are both lucky-10 numbers. Given a string of digits $D[1..n]$, describe and analyze a dynamic programming algorithm that computes the length of a longest lucky-10 subsequence of the string. *[Hint: Try to use your solution to problem 1 **directly**.]*

4. **To think about later:** Can you solve problem 1 in $O(n)$ space?

1. A ***vertex cover*** of a graph is a subset S of the vertices such that every vertex v either belongs to S or has a neighbor in S. In other words, the vertices in S cover all the edges. Finding the minimum size of a vertex cover is NP-hard, but in trees it can be found using dynamic programming.

   Given a tree T and non-negative weight $w(v)$ for each vertex v, describe an algorithm computing the minimum weight of a vertex cover of T.

2. Suppose you are given an unparenthesized mathematical expression containing n numbers, where the only operators are $+$ and $-$; for example:

$$1+3-2-5+1-6+7$$

   You can change the value of the expression by adding parentheses in different positions. For example:

$$1+3-2-5+1-6+7 = -1$$
$$(1+3-(2-5))+(1-6)+7 = 9$$
$$(1+(3-2))-(5+1)-(6+7) = -17$$

   Design an algorithm that, given a list of integers separated by $+$ and $-$ signs, determines the maximum possible value the expression can take by adding parentheses.

   You can only insert parentheses immediately before and immediately after numbers; in particular, you are not allowed to insert implicit multiplication as in $1+3(-2)(-5)+1-6+7 = 33$.

3. Fix an arbitrary sequence $c_1 < c_2 < \cdots < c_k$ of coin values, all in cents. We have an infinite number of coins of each denomination. Describe a dynamic programming algorithm to determine, given an arbitrary non-negative integer x, the least number of coins whose total value is x. For simplicity, you may assume that $c_1 = 1$.

   To think about later after learning "greedy algorithms":

   (a) Describe a greedy algorithm to make change consisting of quarters, dimes, nickels, and pennies. Prove that your algorithm yields an optimal solution.

   (b) Suppose that the available coins have the values $c^0, c^1, \ldots, c^k$ for some integers $c > 1$ and $k \geq 1$. Show that the greedy algorithm always yields an optimal solution.

   (c) Describe a set of 4 coin values for which the greedy algorithm does ***not*** yield an optimal solution.

Note: All the questions in this session are taken from past CS473 midterms.

1. (Fall 2006) **Multiple Choice:** Each of the questions on this page has one of the following five answers: For each question, write the letter that corresponds to your answer.

A: $\Theta(1)$	B: $\Theta(\log n)$	C: $\Theta(n)$	D: $\Theta(n \log n)$	E: $\Theta(n)$

   (a) What is $\frac{5}{n} + \frac{n}{5}$?

   (b) What is $\sum_{i=1}^{n} \frac{n}{i}$?

   (c) What is $\sum_{i=1}^{n} \frac{i}{n}$?

   (d) How many bits are required to represent the nth Fibonacci number in binary?

   (e) What is the solution to the recurrence $T(n) = 2T(n/4) + \Theta(n)$?

   (f) What is the solution to the recurrence $T(n) = 16T(n/4) + \Theta(n)$?

   (g) What is the solution to the recurrence $T(n) = T(n-1) + \frac{1}{n^2}$?

   (h) What is the worst-case time to search for an item in a binary search tree?

   (i) What is the worst-case running time of quicksort?

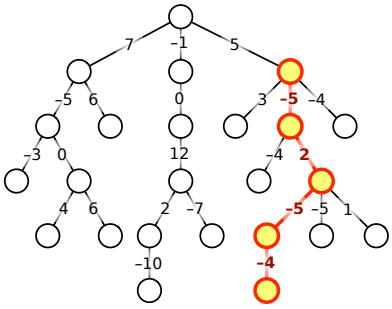
   (j) What is the running time of the fastest possible algorithm to solve Sudoku puzzles? A Sudoku puzzle consists of a 9×9 grid of squares, partitioned into nine 3×3 sub-grids; some of the squares contain digits between 1 and 9. The goal of the puzzle is to enter digits into the blank squares, so that each digit between 1 and 9 appears exactly once in each row, each column, and each 3×3 sub-grid. The initial conditions guarantee that the solution is unique.

2							4	
	7		5					
				1		9		
6		4			2			
	8						5	
			9			3		7
		1		4				
					3		8	
	5							6

A Sudoku puzzle. **Don't try to solve this during the exam!**

2. (Spring 2010) Let T be a rooted tree with integer weights on its edges, which could be positive, negative, or zero. The weight of a path in T is the sum of the weights of its edges. Describe and analyze an algorithm to compute the minimum weight of any path from a node in T down to one of its descendants. It is not necessary to compute the actual minimum-weight path; just its weight. For example, given the tree shown below, your algorithm should return the number -12.

3. (Fall 2006) Suppose you are given an array $A[1..n]$ of n distinct integers, sorted in increasing order. Describe and analyze an algorithm to determine whether there is an index i such that $A[i] = i$, in $o(n)$ time. [Hint: Yes, that's little-oh of n. What can you say about the sequence $A[i] - i$?]



The minimum-weight downward path in this tree has weight −12.

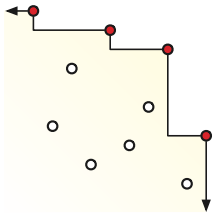
4. (Spring 2010 and Spring 2004) Describe and analyze efficient algorithms to solve the following problems:

   (a) Given a set of n integers, does it contain two elements a, b such that $a + b = 0$?

   (b) Given a set of n integers, does it contain three elements a, b, c such that $a + b = c$?

1. What is the *exact* expected number of leaves in a treap with n nodes?

2. Recall question 5 from Midterm 1:

   > Suppose you are given a set P of n points in the plane. A point $p \in P$ is *maximal* in P if no other point in P is both above and to the right of P. Intuitively, the maximal points define a "staircase" with all the other points of P below it.
   >
   > 
   >
   > A set of ten points, four of which are maximal.
   >
   > Describe and analyze an algorithm to compute the number of maximal points in P in $O(n \log n)$ time. For example, given the ten points shown above, your algorithm should return the integer 4.

   Suppose the points in P are generated independently and uniformly at random in the unit square $[0,1]^2$. What is the *exact* expected number of maximal points in P?

3. Suppose you want to write an app for your new Pebble smart watch that monitors the global Twitter stream and selects a small sample of *random* tweets. You will not know when the stream ends until your app attempts to read the next tweet and receives the error message FAILWHALE. The Pebble has only a small amount of memory, far too little to store the entire stream.

   (a) Describe an algorithm that, as soon as the stream ends, returns a single tweet chosen uniformly at random from the stream. Prove your algorithm is correct. (You may assume that the stream contains at least one tweet.)

   (b) Now fix an arbitrary positive integer k. Describe an algorithm that picks k tweets uniformly at random from the stream. Prove your algorithm is correct. (You may assume that the stream contains at least k tweets.)

Recall the following elementary data structures from CS 225.

- A *stack* supports the following operations.
  - PUSH pushes an element on top of the stack.
  - POP removes the top element from a stack.
  - ISEMPTY checks if a stack is empty.
- A *queue* supports the following operations.
  - PUSH adds an element to the back of the queue.
  - PULL removes an element from the front of the queue.
  - ISEMPTY checks if a queue is empty.
- A *deque*, or double-ended queue, supports the following operations.
  - PUSH adds an element to the back of the queue.
  - PULL removes an element from the back of the queue.
  - CUT adds an element from the front of the queue.
  - POP removes an element from the front of the queue.
  - ISEMPTY checks if a queue is empty.

Suppose you have a stack implementation that supports all stack operations in constant time.

1. Describe how to implement a queue using two stacks and $O(1)$ additional memory, so that each queue operation runs in $O(1)$ amortized time.
2. Describe how to implement a deque using three stacks and $O(1)$ additional memory, so that each deque operation runs in $O(1)$ amortized time.

1. Let P be a set of n points in the plane. Recall from the midterm that the *staircase* of P is the set of all points in the plane that have at least one point in P both above and to the right.

   (a) Describe and analyze a data structure that stores the staircase of a set of points, and an algorithm ABOVE?(x, y) that returns TRUE if the point (x, y) is above the staircase, or FALSE otherwise. Your data structure should use $O(n)$ space, and your ABOVE? algorithm should run in $O(\log n)$ time.

   (b) Describe and analyze a data structure that maintains the staircase of a set of points as new points are inserted. Specifically, your data structure should support a function INSERT(x, y) that adds the point (x, y) to the underlying point set and returns TRUE or FALSE to indicate whether the staircase of the set has changed. Your data structure should use $O(n)$ space, and your INSERT algorithm should run in $O(\log n)$ amortized time.

2. An *ordered stack* is a data structure that stores a sequence of items and supports the following operations.

   - ORDEREDPUSH(x) removes all items smaller than x from the beginning of the sequence and then adds x to the beginning of the sequence.
   - POP deletes and returns the first item in the sequence (or NULL if the sequence is empty).

   Suppose we implement an ordered stack with a simple linked list, using the obvious ORDEREDPUSH and POP algorithms. Prove that if we start with an empty data structure, the amortized cost of each ORDEREDPUSH or POP operation is $O(1)$.

3. Consider the following solution for the union-find problem, called *union-by-weight*. Each set leader $\bar{x}$ stores the number of elements of its set in the field weight$(\bar{x})$. Whenever we UNION two sets, the leader of the *smaller* set becomes a new child of the leader of the *larger* set (breaking ties arbitrarily).

```
MAKESET(x)
  parent(x) ← x
  weight(x) ← 1
```

```
FIND(x)
  while x ≠ parent(x)
      x ← parent(x)
  return x
```

```
UNION(x, y)
  x̄ ← FIND(x)
  ȳ ← FIND(y)
  if weight(x̄) > weight(ȳ)
      parent(ȳ) ← x̄
      weight(x̄) ← weight(x̄) + weight(ȳ)
  else
      parent(x̄) ← ȳ
      weight(x̄) ← weight(x̄) + weight(ȳ)
```

   Prove that if we use union-by-weight, the *worst-case* running time of FIND(x) is $O(\log n)$, where n is the cardinality of the set containing x.

1. Let G be an undirected graph.

   (a) Suppose we start with two coins on two arbitrarily chosen nodes. At every step, each coin must move to an adjacent node. Describe an algorithm to compute the minimum number of steps to reach a configuration that two coins are on the same node.

   (b) Now suppose there are three coins, numbered 0, 1, and 2. Again we start with an arbitrary coin placement with all three coins facing up. At each step, we move each coin to an adjacent node at each step. Moreover, for every integer i, we flip coin i mod 3 at the ith step. Describe an algorithm to compute the minimum number of steps to reach a configuration that all three coins are on the same node and all facing up. What is the running time of your algorithm?

2. Let G be a directed acyclic graph with a unique source s and a unique sink t.

   (a) A *Hamiltonian path* in G is a directed path in G that contains every vertex in G. Describe an algorithm to determine whether G has a Hamiltonian path.

   (b) Suppose several nodes in G are marked to be *important*; also an integer k is given. Design an algorithm which computes all the nodes that can reach t through at least k important nodes.

   (c) Suppose the edges in G have real weights. Describe an algorithm to find a path from s to t with maximum total weight.

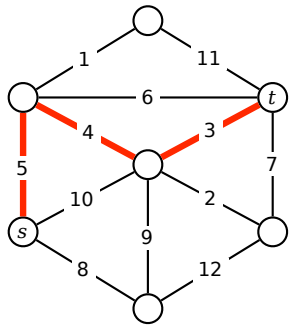
   (d) Suppose the vertices of G have labels from a fixed finite alphabet, and let $A[1..\ell]$ be a string over the same alphabet. Any directed path in G has a label, which is obtained by concatenating the labels of its vertices. Describe an algorithm to find the longest path in G whose labels are a subsequence of A.

3. Let G be a directed graph with a special source that has an edge to each other node in graph, and denote $scc(G)$ as the strong component graph of G. Let S and S' be two strongly connected components in G with $S \to S'$ an arc in $scc(G)$. (That is, if there is an arc between node $u \in S$ and $v \in S'$, then it must be $u \to v$.) Consider a fixed depth-first search performed on G starting at s; we define post$(\cdot)$ as the post-order numbering of the search.

   (a) Prove or disprove that we have $\text{post}(u) > \text{post}(u')$ for any $u \in S$ and $u' \in S'$.

   (b) Prove or disprove that we have $\max_{u \in S} \text{post}(u) > \max_{u' \in S'} \text{post}(u')$.

1. Consider a path between two vertices s and t in an undirected weighted graph G. The *bottleneck length* of this path is the maximum weight of any edge in the path. The *bottleneck distance* between s and t is the minimum bottleneck length of any path from s to t. (If there are no paths from s to t, the bottleneck distance between s and t is ∞.)

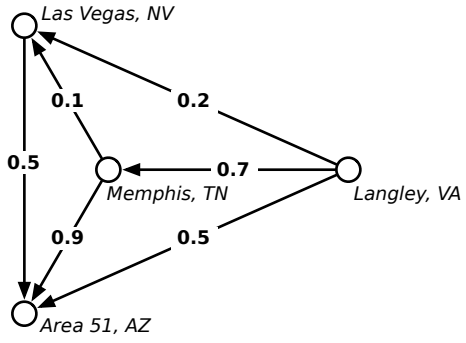


   Describe an algorithm to compute the bottleneck distance between *every* pair of vertices in an arbitrary undirected weighted graph. Assume that no two edges have the same weight.

2. Let G be a directed graph with (possibly negative) edge weights, and let s be an arbitrary vertex of G. Suppose for each vertex v we are given a real number $d(v)$. Describe and analyze an algorithm to determine whether the numbers $d(v)$ on vertices are the shortest path distances from s to each vertex v. Do not assume that the graph G has no negative cycles.

3. Mulder and Scully have computed, for every road in the United States, the exact probability that someone driving on that road *won't* be abducted by aliens. Agent Mulder needs to drive from Langley, Virginia to Area 51, Nevada. What rout should hs take so that he has the least chance of being abducted?

   More formally, you are given a directed graph G, possibly with cycles, where every edge e has an independent safety probability $p(e)$. The *safety* of a path is the product of the safety probabilities of its edges. Design and analyze an algorithm to determine the safest path from a given start vertex s to a given target vertex t.



   For example, with the probabilities shown above, if Mulder tries to drive directly from Langley to Area 51, he has a 50% chance of getting there without being abducted. If he stops in Memphis, he has a $0.7 \times 0.9 = 63\%$ chance of arriving safely. If he stops first in Memphis and then in Las Vegas, he has a $1 - 0.7 \times 0.1 \times 0.5 = 96.5\%$ chance of begin abducted![1]

[1] That's how they got Elvis, you know.

Almost all these review problems from from past midterms.

1. [Fall 2002, Spring 2004] Suppose we want to maintain a set X of numbers, under the following operations:

   - INSERT(x): Add x to the set (if it isn't already there).
   - PRINT&DELETEBETWEEN(a, b): Print every element $x \in X$ such that $a \le x \le b$, in order from smallest to largest, and then delete those elements from X.

   For example, if the current set is $\{1, 5, 3, 4, 8\}$, then PRINT&DELETEBETWEEN$(4, 6)$ prints the numbers 4 and 5 and changes the set to $\{1, 3, 8\}$.

   Describe and analyze a data structure that supports these two operations, each in $O(\log n)$ *amortized* time, where n is the maximum number of elements in X.

2. [Spring 2004] Consider a random walk on a path with vertices numbered $1, 2, \ldots, n$ from left to right. We start at vertex 1. At each step, we flip a coin to decide which direction to walk, moving one step left or one step right with equal probability. The random walk ends when we fall off one end of the path, either by moving left from vertex 1 or by moving right from vertex n.

   Prove that the probability that the walk ends by falling off the *left* end of the path is exactly $n/(n+1)$. [Hint: Set up a recurrence and verify that $n/(n+1)$ satisfies it.]

3. [Fall 2006] ***Prove or disprove*** each of the following statements.

   (a) Let G be an arbitrary undirected graph with arbitrary distinct weights on the edges. The minimum spanning tree of G includes the lightest edge in every cycle in G.

   (b) Let G be an arbitrary undirected graph with arbitrary distinct weights on the edges. The minimum spanning tree of G excludes the heaviest edge in every cycle in G.

4. [Fall 2012] Let $G = (V, E)$ be a connected undirected graph. For any vertices u and v, let $d_G(u, v)$ denote the length of the shortest path in G from u to v. For any sets of vertices A and B, let $d_G(A, B)$ denote the length of the shortest path in G from any vertex in A to any vertex in B:

   $$d_G(A, B) = \min_{u \in A} \min_{v \in B} d_G(u, v).$$

   Describe and analyze a fast algorithm to compute $d_G(A, B)$, given the graph G and subsets A and B as input. You do not need to prove that your algorithm is correct.

5. Let G and H be directed acyclic graphs, whose vertices have labels from some fixed alphabet, and let $A[1..\ell]$ be a string over the same alphabet. Any directed path in G has a label, which is a string obtained by concatenating the labels of its vertices.

   (a) Describe an algorithm to find the longest string that is both a label of a directed path in G and the label of a directed path in H.

   (b) Describe an algorithm to find the longest string that is both a *subsequence* of the label of a directed path in G and *subsequence* of the label of a directed path in H.

1. The Island of Sodor is home to a large number of towns and villages, connected by an extensive rail network. Recently, several cases of a deadly contagious disease (either swine flu or zombies; reports are unclear) have been reported in the village of Ffarquhar. The controller of the Sodor railway plans to close down certain railway stations to prevent the disease from spreading to Tidmouth, his home town. No trains can pass through a closed station. To minimize expense (and public notice), he wants to close down as few stations as possible. However, he cannot close the Ffarquhar station, because that would expose him to the disease, and he cannot close the Tidmouth station, because then he couldn't visit his favorite pub.

   Describe and analyze an algorithm to find the minimum number of stations that must be closed to block all rail travel from Ffarquhar to Tidmouth. The Sodor rail network is represented by an undirected graph, with a vertex for each station and an edge for each rail connection between two stations. Two special vertices f and t represent the stations in Ffarquhar and Tidmouth.

2. Given an undirected graph $G = (V, E)$, with three vertices u, v, and w, describe and analyze an algorithm to determine whether there is a path from u to w that passes through v.

3. Suppose you have already computed a maximum flow f^* in a flow network G with *integer* edge capacities.

   (a) Describe and analyze an algorithm to update the maximum flow after the capacity of a single edge is increased by 1.

   (b) Describe and analyze an algorithm to update the maximum flow after the capacity of a single edge is decreased by 1.

   Both algorithms should be significantly faster than recomputing the maximum flow from scratch.

1. An *American graph* is a directed graph with each vertex colored *red*, *white*, or *blue*. An *American Hamiltonian path* is a Hamiltonian path that cycles between red, white, and blue vertices; that is, every edge goes from red to white, or white to blue, or blue to red. The AMERICANHAMILTONIANPATH problem asks whether there is an American Hamiltonian path in an American graph.

   (a) Prove that AMERICANHAMILTONIANPATH is NP-complete by reducing from HAMILTONIANPATH.

   (b) In the opposite direction, reduce AMERICANHAMILTONIANPATH to HAMILTONIANPATH.

2. Given a graph G, the DEG17SPANNINGTREE problem asks whether G has a spanning tree in which each vertex of the spanning tree has degree at most 17. Prove that DEG17SPANNINGTREE is NP-complete.

3. Two graphs are *isomorphic* if one can be transformed into the other by relabeling the vertices. Consider the following related decision problems:

   - GRAPHISOMORPHISM: Given two graphs G and H, determine whether G and H are isomorphic.
   - EVENGRAPHISOMORPHISM: Given two graphs G and H, such that every vertex of G and H have even degree, determine whether G and H are isomorphic.
   - SUBGRAPHISOMORPHISM: Given two graphs G and H, determine whether G is isomorphic to a subgraph of H.

   (a) Describe a polynomial time reduction from GRAPHISOMORPHISM to EVENGRAPHISOMORPHISM.

   (b) Describe a polynomial time reduction from GRAPHISOMORPHISM to SUBGRAPHISOMORPHISM.

1. Prove that the following problem is NP-hard.

   SETCOVER: Given a collection of sets $\{S_1, \ldots, S_m\}$, find the smallest sub-collection of S_i's that contains all the elements of $\bigcup_i S_i$.

2. Given an undirected graph G and a subset of vertices S, a *Steiner tree* of S in G is a subtree of G that contains every vertex in S. If S contains every vertex of G, a Steiner tree is just a spanning tree; if S contains exactly two vertices, any path between them is a Steiner tree.

   Given a graph G, a vertex subset S, and an integer k, the *Steiner tree problem* requires us to decide whether there is a Steiner tree of S in G with at most k edges. Prove that the Steiner tree problem is NP-hard. *[Hint: Reduce from* VERTEXCOVER*, or* SETCOVER*, or* 3SAT*.]*

3. Let G be a directed graph whose edges are colored red and white. A *Canadian Hamiltonian path* is a Hamiltonian path whose edges are alternately red and white. The CANADIANHAMILTONIANPATH problem ask us to find a Canadian Hamiltonian path in a graph G. (Two weeks ago we looked for Hamiltonian paths that cycled through colors on the *vertices* instead of edges.)

   (a) Prove that CANADIANHAMILTONIANPATH is NP-Complete.

   (b) Reduce CANADIANHAMILTONIANPATH to HAMILTONIANPATH.

This exam lasts 120 minutes.
Write your answers in the separate answer booklet.
Please return this question sheet and your cheat sheet with your answers.

1. Each of these ten questions has one of the following five answers:

   A: $\Theta(1)$ B: $\Theta(\log n)$ C: $\Theta(n)$ D: $\Theta(n \log n)$ E: $\Theta(n^2)$

   (a) What is $\dfrac{n^5 - 3n^3 - 5n + 4}{4n^3 - 2n^2 + n - 7}$?

   (b) What is $\displaystyle\sum_{i=1}^{n} i$?

   (c) What is $\displaystyle\sum_{i=1}^{n} \sqrt{\frac{n}{i}}$?

   (d) How many bits are required to write the integer n^{10} in binary?

   (e) What is the solution to the recurrence $E(n) = E(n/2) + E(n/4) + E(n/8) + 16n$?

   (f) What is the solution to the recurrence $F(n) = 6F(n/6) + 6n$?

   (g) What is the solution to the recurrence $G(n) = 9G(\lceil n/3 \rceil + 1) + n$?

   (h) The *total path length* of a binary tree is the sum of the depths of all nodes. What is the total path length of an n-node binary tree in the worst case?

   (i) Consider the following recursive function, defined in terms of a fixed array $X[1..n]$:

$$WTF(i,j) = \begin{cases} 0 & \text{if } i > j \\ 1 & \text{if } i = j \\ \max \left\{ \begin{array}{c} 2 \cdot \big[X[i] \neq X[j]\big] + WTF(i+1, j-1) \\ 1 + WTF(i+1, j) \\ 1 + WTF(i, j-1) \end{array} \right\} & \text{otherwise} \end{cases}$$

   How long does it take to compute $WTF(1,n)$ using dynamic programming?

   (j) Voyager 1 recently became the first made-made object to reach interstellar space. Currently the spacecraft is about 18 billion kilometers (roughly 60,000 light seconds) from Earth, traveling outward at approximately 17 kilometers per second (approximately 1/18000 of the speed of light). Voyager carries a golden record containing over 100 digital images abd approximately one hour of sound recordings. In digital form, the recording would require about 1 gigabyte. Voyager can transmit data back to Earth at approximately 1400 bits per second. Suppose the engineers at JPL sent instructions to Voyager 1 to send the complete contents of the Golden Record back to Earth; how many seconds would they have to wait to receive the entire record?

2. You are a visitor at a political convention (or perhaps a faculty meeting) with n delegates; each delegate is a member of exactly one political party. It is impossible to tell which political party any delegate belongs to; in particular, you will be summarily ejected from the convention if you ask. However, you can determine whether any pair of delegates belong to the *same* party or not simply by introducing them to each other—members of the same party always greet each other with smiles and friendly handshakes; members of different parties always greet each other with angry stares and insults.

   Suppose more than half of the delegates belong to the same political party. Describe and analyze an efficient algorithm that identifies all members of this majority party.

3. Recall that a *tree* is a connected undirected graph with no cycles. ***Prove*** that in any tree, the number of nodes is exactly one more than the number of edges.

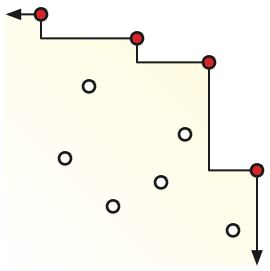
4. Next spring break, you and your friends decide to take a road trip, but before you leave, you decide to figure out *exactly* how much money to bring for gasoline. Suppose you compile a list of all gas stations along your planned route, containing the following information:

   - A sorted array $Dist[0..n]$, where $Dist[0] = 0$ and $Dist[i]$ is the number of miles from the beginning of your route to the ith gas station. Your route ends at the nth gas station.
   - A second array $Price[1..n]$, where $Price[i]$ is the price of one gallon of gasoline at the ith gas station. (Unlike in real life, these prices do not change over time.)

   You start the trip with a full tank of gas. Whenever you buy gas, you must completely fill your tank. Your car holds exactly 10 gallons of gas and travels exactly 25 miles per gallon; thus, starting with a full tank, you can travel exactly 250 miles before your car dies. Finally, $Dist[i+1] < Dist[i]+250$ for every index i, so the trip is possible.

   Describe and analyze an algorithm to determine the minimum amount of money you must spend on gasoline to guarantee that you can drive the entire route.

5. Suppose you are given a set P of n points in the plane. A point $p \in P$ is *maximal* in P if no other point in P is both above and to the right of P. Intuitively, the maximal points define a "staircase" with all the other points of P below it.



A set of ten points, four of which are maximal.

   Describe and analyze an algorithm to compute the number of maximal points in P in $O(n \log n)$ time. For example, given the ten points shown above, your algorithm should return the integer 4.

This exam lasts 120 minutes.
Write your answers in the separate answer booklet.
Please return this question sheet and your cheat sheet with your answers.

1. Each of these ten questions has one of the following five answers:

   A: $\Theta(1)$ B: $\Theta(\log n)$ C: $\Theta(n)$ D: $\Theta(n \log n)$ E: $\Theta(n^2)$

   (a) What is $\dfrac{n^5 - 3n^3 - 5n + 4}{3n^4 - 2n^2 + n - 7}$?

   (b) What is $\displaystyle\sum_{i=1}^{n} \frac{n}{i}$?

   (c) What is $\displaystyle\sum_{i=1}^{n} \frac{i}{n}$?

   (d) How many bits are required to write the integer 10^n in binary?

   (e) What is the solution to the recurrence $E(n) = E(n/3) + E(n/4) + E(n/5) + n/6$?

   (f) What is the solution to the recurrence $F(n) = 16F(n/4 + 2) + n$?

   (g) What is the solution to the recurrence $G(n) = G(n/2) + 2G(n/4) + n$?

   (h) The *total path length* of a binary tree is the sum of the depths of all nodes. What is the total path length of a perfectly balanced n-node binary tree?

   (i) Consider the following recursive function, defined in terms of two fixed arrays $A[1..n]$ and $B[1..n]$:

   $$WTF(i,j) = \begin{cases} 0 & \text{if } i > j \\ \max\left\{ \begin{array}{c} (A[i] - B[j])^2 + WTF(i+1, j-1) \\ A[i]^2 + WTF(i+1, j) \\ B[i]^2 + WTF(i, j-1) \end{array} \right\} & \text{otherwise} \end{cases}$$

   How long does it take to compute $WTF(1, n)$ using dynamic programming?

   (j) Voyager 1 recently became the first made-made object to reach interstellar space. Currently the spacecraft is about 18 billion kilometers (roughly 60,000 light seconds) from Earth, traveling outward at approximately 17 kilometers per second (approximately 1/18000 of the speed of light). Voyager carries a golden record containing over 100 digital images and approximately one hour of sound recordings. In digital form, the recording would require about 1 gigabyte. Voyager can transmit data back to Earth at approximately 1400 bits per second. Suppose the engineers at JPL sent instructions to Voyager 1 to send the complete contents of the Golden Record back to Earth; how many seconds would they have to wait to receive the entire record?

2. Suppose we are given an array $A[0..n+1]$ with fencepost values $A[0] = A[n+1] = -\infty$. We say that an element $A[x]$ is a *local maximum* if it is less than or equal to its neighbors, or more formally, if $A[x-1] \le A[x]$ and $A[x] \ge A[x+1]$. For example, there are five local maxima in the following array:

$-\infty$	6	⑦	2	1	3	⑦	5	4	⑨	⑨	3	4	⑧	6	$-\infty$

We can obviously find a local maximum in $O(n)$ time by scanning through the array. Describe and analyze an algorithm that returns the index of one local maximum in $O(\log n)$ time. *[Hint: With the given boundary conditions, the array **must** have at least one local maximum. Why?]*

3. ***Prove*** that in any binary tree, the number of nodes with no children (leaves) is exactly one more than the number of nodes with two children. (Remember that a binary tree can have nodes with only one child.)

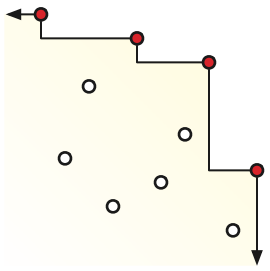
4. A string x is a *supersequence* of a string y if we can obtain x by inserting zero or more letters into y, or equivalently, if y is a subsequence of x. For example, the string **D**YN**AM**IC**P**ROG**RA**MMIN**G** is a supersequence of the string DAMPRAG.

   A *palindrome* is any string that is exactly the same as its reversal, like I, DAD, HANNAH, AIBOHPHOBIA (fear of palindromes), or the empty string.

   Describe and analyze an algorithm to find the length of the shortest supersequence of a given string that is also a palindrome.

   For example, the 11-letter string E**HE**C**ADAC**E**HE** is the shortest palindrome supersequence of HEADACHE, so given the string HEADACHE as input, your algorithm should output the number 11.

5. Suppose you are given a set P of n points in the plane. A point $p \in P$ is *maximal* in P if no other point in P is both above and to the right of P. Intuitively, the maximal points define a "staircase" with all the other points of P below it.



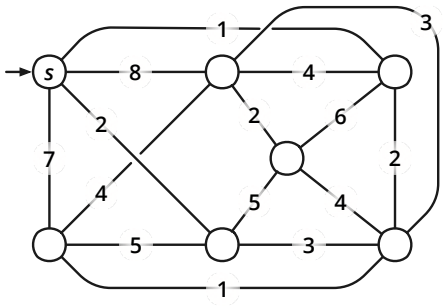
A set of ten points, four of which are maximal.

   Describe and analyze an algorithm to compute the number of maximal points in P in $O(n \log n)$ time. For example, given the ten points shown above, your algorithm should return the integer 4.

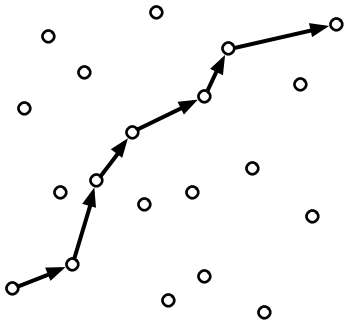
This exam lasts 120 minutes.
Write your answers in the separate answer booklet.
Please return this question sheet and your cheat sheet with your answers.

1. ***Clearly*** indicate the following spanning trees in the weighted graph pictured below. Some of these subproblems have more than one correct answer.

   (a) A depth-first spanning tree rooted at s
   (b) A breadth-first spanning tree rooted at s
   (c) A shortest-path tree rooted at s
   (d) A minimum spanning tree
   (e) A *maximum* spanning tree



2. A ***polygonal path*** is a sequence of line segments joined end-to-end; the endpoints of these line segments are called the ***vertices*** of the path. The ***length*** of a polygonal path is the sum of the lengths of its segments. A polygonal path with vertices $(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)$ is ***monotonically increasing*** if $x_i < x_{i+1}$ and $y_i < y_{i+1}$ for every index i—informally, each vertex of the path is above and to the right of its predecessor.



A monotonically increasing polygonal path with seven vertices through a set of points

Suppose you are given a set S of n points in the plane, represented as two arrays $X[1..n]$ and $Y[1..n]$. Describe and analyze an algorithm to compute the length of the maximum-length monotonically increasing path with vertices in S. Assume you have a subroutine $\textsc{Length}(x, y, x', y')$ that returns the length of the segment from (x, y) to (x', y').

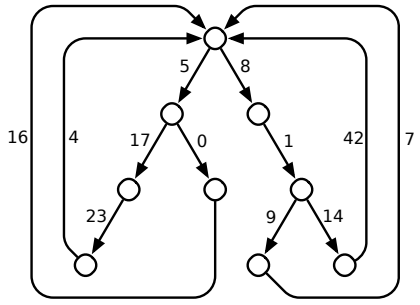
3. Suppose you are maintaining a circular array $X[0..n-1]$ of counters, each taking a value from the set $\{0,1,2\}$. The following algorithm increments one of the counters; if the counter overflows, the algorithm resets it 0 and recursively increments its two neighbors.

```
INCREMENT(i):
  X[i] ← X[i] + 1
  if X[i] = 3
      X[i] ← 0
      INCREMENT((i − 1) mod n)
      INCREMENT((i + 1) mod n)
```

   (a) Suppose $n = 5$ and $X = [2,2,2,2,2]$. What does X contain after we call INCREMENT(3)?

   (b) Suppose all counters are initially 0. ***Prove*** that INCREMENT runs in $O(1)$ amortized time.

4. A *looped tree* is a weighted, directed graph built from a binary tree by adding an edge from every leaf back to the root. Every edge has non-negative weight.



A looped tree.

   (a) How much time would Dijkstra's algorithm require to compute the shortest path from an arbitrary vertex s to another arbitrary vertex t, in a looped tree with n vertices?

   (b) Describe and analyze a faster algorithm. Your algorithm should compute the actual shortest path, not just its length.

5. Consider the following algorithm for finding the smallest element in an unsorted array:

```
RANDOMMIN(A[1..n]):
  min ← ∞
  for i ← 1 to n in random order
      if A[i] < min
          min ← A[i]    (⋆)
  return min
```

   Assume the elements of A are all distinct.

   (a) In the worst case, how many times does RANDOMMIN execute line $(\star)$?

   (b) What is the probability that line $(\star)$ is executed during the *last* iteration of the for loop?

   (c) What is the *exact* expected number of executions of line $(\star)$?

This exam lasts 180 minutes.
Write your answers in the separate answer booklet.
Please return this question handout and your cheat sheets with your answers.

1. Suppose you are given a sorted array of n distinct numbers that has been *rotated* k steps, for some ***unknown*** integer k between 1 and $n-1$. That is, you are given an array $A[1..n]$ such that the prefix $A[1..k]$ is sorted in increasing order, the suffix $A[k+1..n]$ is sorted in increasing order, and $A[n] < A[1]$. For example, you might be given the following 16-element array (where $k = 10$):

9	13	16	18	19	23	28	31	37	42	−4	0	2	5	7	8

   Describe and analyze an algorithm to determine if the given array contains a given number x. For example, given the previous array and the number 17 as input, your algorithm should return FALSE. The index k is ***NOT*** part of the input.

2. You are hired as a cyclist for the Giggle Highway View project, which will provide street-level images along the entire US national highway system. As a pilot project, you are asked to ride the Giggle Highway-View Fixed-Gear Carbon-Fiber Bicycle from "the Giggleplex" in Portland, Oregon to "Giggleburg" in Williamsburg, Brooklyn, New York.

   You are a hopeless caffeine addict, but like most Giggle employees you are also a coffee snob; you only drink independently roasted organic shade-grown single-origin espresso. After each espresso shot, you can bike up to L miles before suffering a caffeine-withdrawal migraine.

   Giggle helpfully provides you with a map of the United States, in the form of an undirected graph G, whose vertices represent coffee shops that sell independently roasted organic shade-grown single-origin espresso, and whose edges represent highway connections between them. Each edge e is labeled with the length $\ell(e)$ of the corresponding stretch of highway. Naturally, there are espresso stands at both Giggle offices, represented by two specific vertices s and t in the graph G.

   (a) Describe and analyze an algorithm to determine whether it is possible to bike from the Giggleplex to Giggleburg without suffering a caffeine-withdrawal migraine.

   (b) You discover that by wearing a more expensive fedora, you can increase the distance L that you can bike between espresso shots. Describe and analyze and algorithm to find the minimum value of L that allows you to bike from the Giggleplex to Giggleburg without suffering a caffeine-withdrawal migraine.

3. Suppose you are given a collection of up-trees representing a partition of the set $\{1, 2, \ldots, n\}$ into subsets. **You have no idea how these trees were constructed.** You are also given an array $node[1..n]$, where $node[i]$ is a pointer to the up-tree node containing element i. Your task is to create a new array $label[1..n]$ using the following algorithm:

```
LABELEVERYTHING:
  for i ← 1 to n
      label[i] ← FIND(node[i])
```

Recall that there are two natural ways to implement FIND: simple pointer-chasing and pointer-chasing with path compression. Pseudocode for both methods is shown below.

```
FIND(x):
  while x ≠ parent(x)
      x ← parent(x)
  return x
```

Without path compression

```
FIND(x):
  if x ≠ parent(x)
      parent(x) ← FIND(parent(x))
  return parent(x)
```

With path compression

(a) What is the worst-case running time of LABELEVERYTHING if we implement FIND *without* path compression?

(b) ***Prove*** that if we implement FIND using path compression, LABELEVERYTHING runs in $O(n)$ time in the worst case.

4. Congratulations! You have successfully conquered Camelot, transforming the former battle-scarred kingdom with an anarcho-syndicalist commune, where citizens take turns to act as a sort of executive-officer-for-the-week, but with all the decisions of that officer ratified at a special bi-weekly meeting, by a simple majority in the case of purely internal affairs, but by a two-thirds majority, in the case of more major....

As a final symbolic act, you order the Round Table (surprisingly, an actual circular table) to be split into pizza-like wedges and distributed to the citizens of Camelot as trophies. Each citizen has submitted a request for an angular wedge of the table, specified by two angles—for example, Sir Robin the Brave might request the wedge from 23.17° to 42°. Each citizen will be happy if and only if they receive *precisely* the wedge that they requested. Unfortunately, some of these ranges overlap, so satisfying *all* the citizens' requests is simply impossible. Welcome to politics.

Describe and analyze an algorithm to find the maximum number of requests that can be satisfied.

5. The NSA has established several monitoring stations around the country, each one conveniently hidden in the back of a Starbucks. Each station can monitor up to 42 cell-phone towers, but can only monitor cell-phone towers within a 20-mile radius. To ensure that every cell-phone call is recorded even if some stations malfunction, the NSA requires each cell-phone tower to be monitored by at least 3 different stations.

   Suppose you know that there are n cell-phone towers and m monitoring stations, and you are given a function DISTANCE(i, j) that returns the distance between the ith tower and the jth station in $O(1)$ time. Describe and analyze an algorithm that either computes a valid assignment of cell-phone towers to monitoring stations, or reports correctly that there is no such assignment (in which case the NSA will build another Starbucks).

6. Consider the following closely related problems:

   - HAMILTONIANPATH: Given an undirected graph G, determine whether G contains a ***path*** that visits every vertex of G exactly once.
   - HAMILTONIANCYCLE: Given an undirected graph G, determine whether G contains a ***cycle*** that visits every vertex of G exactly once.

   Describe a polynomial-time reduction from HAMILTONIANPATH to HAMILTONIANCYCLE. ***Prove*** your reduction is correct. *[Hint: A polynomial-time reduction is allowed to call the black-box subroutine more than once.]*

7. An array $X[1..n]$ of distinct integers is ***wobbly*** if it alternates between increasing and decreasing: $X[i] < X[i+1]$ for every odd index i, and $X[i] > X[i+1]$ for every even index i. For example, the following 16-element array is wobbly:

12	13	0	16	13	31	5	7	−1	23	8	10	−4	37	17	42

   Describe and analyze an algorithm that permutes the elements of a given array to make the array wobbly.

You may use the following algorithms as black boxes:

RANDOM(k): Given any positive integer k, return an integer chosen independently and uniformly at random from the set $\{1, 2, \ldots, k\}$ in $O(1)$ time.

ORLINMAXFLOW(V, E, c, s, t): Given a directed graph $G = (V, E)$, a capacity function $c : E \to \mathbb{R}^+$, and vertices s and t, return a maximum (s, t)-flow in G in $O(VE)$ time. If the capacities are integral, so is the returned maximum flow.

Any other algorithm that we described in class.

You may assume the following problems are NP-hard:

CIRCUITSAT: Given a boolean circuit, are there any input values that make the circuit output TRUE?

PLANARCIRCUITSAT: Given a boolean circuit drawn in the plane so that no two wires cross, are there any input values that make the circuit output TRUE?

3SAT: Given a boolean formula in conjunctive normal form, with exactly three literals per clause, does the formula have a satisfying assignment?

MAXINDEPENDENTSET: Given an undirected graph G, what is the size of the largest subset of vertices in G that have no edges among them?

MAXCLIQUE: Given an undirected graph G, what is the size of the largest complete subgraph of G?

MINVERTEXCOVER: Given an undirected graph G, what is the size of the smallest subset of vertices that touch every edge in G?

MINSETCOVER: Given a collection of subsets $S_1, S_2, \ldots, S_m$ of a set S, what is the size of the smallest subcollection whose union is S?

MINHITTINGSET: Given a collection of subsets $S_1, S_2, \ldots, S_m$ of a set S, what is the size of the smallest subset of S that intersects every subset S_i?

3COLOR: Given an undirected graph G, can its vertices be colored with three colors, so that every edge touches vertices with two different colors?

HAMILTONIANCYCLE: Given a graph G, is there a cycle in G that visits every vertex exactly once?

HAMILTONIANPATH: Given a graph G, is there a path in G that visits every vertex exactly once?

TRAVELINGSALESMAN: Given a graph G with weighted edges, what is the minimum total weight of any Hamiltonian path/cycle in G?

STEINERTREE: Given an undirected graph G with some of the vertices marked, what is the minimum number of edges in a subtree of G that contains every marked vertex?

SUBSETSUM: Given a set X of positive integers and an integer k, does X have a subset whose elements sum to k?

PARTITION: Given a set X of positive integers, can X be partitioned into two subsets with the same sum?

3PARTITION: Given a set X of $3n$ positive integers, can X be partitioned into n three-element subsets, all with the same sum?

DRAUGHTS: Given an $n \times n$ international draughts configuration, what is the largest number of pieces that can (and therefore must) be captured in a single move?

DOGE: Such N. Many P. Wow.

This exam lasts 180 minutes.
Write your answers in the separate answer booklet.
Please return this question handout and your cheat sheets with your answers.

1. Suppose you are given a sorted array of n distinct numbers that has been *rotated* k steps, for some ***unknown*** integer k between 1 and $n-1$. That is, you are given an array $A[1..n]$ such that the prefix $A[1..k]$ is sorted in increasing order, the suffix $A[k+1..n]$ is sorted in increasing order, and $A[n] < A[1]$. Describe and analyze an algorithm to compute the unknown integer k.

   For example, given the following array as input, your algorithm should output the integer 10.

9	13	16	18	19	23	28	31	37	42	−4	0	2	5	7	8

2. You are hired as a cyclist for the Giggle Highway View project, which will provide street-level images along the entire US national highway system. As a pilot project, you are asked to ride the Giggle Highway-View Fixed-Gear Carbon-Fiber Bicycle from "the Giggleplex" in Portland, Oregon to "Giggleburg" in Williamsburg, Brooklyn, New York.

   You are a hopeless caffeine addict, but like most Giggle employees you are also a coffee snob; you only drink independently roasted organic shade-grown single-origin espresso. After each espresso shot, you can bike up to L miles before suffering a caffeine-withdrawal migraine.

   Giggle helpfully provides you with a map of the United States, in the form of an undirected graph G, whose vertices represent coffee shops that sell independently roasted organic shade-grown single-origin espresso, and whose edges represent highway connections between them. Each edge e is labeled with the length $\ell(e)$ of the corresponding stretch of highway. Naturally, there are espresso stands at both Giggle offices, represented by two specific vertices s and t in the graph G.

   (a) Describe and analyze an algorithm to determine whether it is possible to bike from the Giggleplex to Giggleburg without suffering a caffeine-withdrawal migraine.

   (b) When you report to your supervisor (whom Giggle recently hired away from competitor Yippee!) that the ride is impossible, she demands to look at your map. "Oh, I see the problem; there are no *Starbucks* on this map!" As you look on in horror, she hands you an updated graph G' that includes a vertex for every Starbucks location in the United States, helpfully marked in Starbucks Green (Pantone® 3425 C).

   Describe and analyze an algorithm to find the minimum number of Starbucks locations you must visit to bike from the Giggleplex to Giggleburg without suffering a caffeine-withdrawal migraine. More formally, your algorithm should find the minimum number of green vertices on any path in G' from s to t that uses only edges of length at most L.

3. Suppose you are given a collection of up-trees representing a partition of the set $\{1, 2, \ldots, n\}$ into subsets. **You have no idea how these trees were constructed.** You are also given an array $node[1..n]$, where $node[i]$ is a pointer to the up-tree node containing element i. Your task is to create a new array $label[1..n]$ using the following algorithm:

```
LABELEVERYTHING:
  for i ← 1 to n
      label[i] ← FIND(node[i])
```

Recall that there are two natural ways to implement FIND: simple pointer-chasing and pointer-chasing with path compression. Pseudocode for both methods is shown below.

```
FIND(x):
  while x ≠ parent(x)
      x ← parent(x)
  return x
```

Without path compression

```
FIND(x):
  if x ≠ parent(x)
      parent(x) ← FIND(parent(x))
  return parent(x)
```

With path compression

(a) What is the worst-case running time of LABELEVERYTHING if we implement FIND *without* path compression?

(b) ***Prove*** that if we implement FIND using path compression, LABELEVERYTHING runs in $O(n)$ time in the worst case.

4. Congratulations! You have successfully conquered Camelot, transforming the former battle-scarred kingdom with an anarcho-syndicalist commune, where citizens take turns to act as a sort of executive-officer-for-the-week, but with all the decisions of that officer ratified at a special bi-weekly meeting, by a simple majority in the case of purely internal affairs, but by a two-thirds majority, in the case of more major....

As a final symbolic act, you order the Round Table (surprisingly, an actual circular table) to be split into pizza-like wedges and distributed to the citizens of Camelot as trophies. Each citizen has submitted a request for an angular wedge of the table, specified by two angles—for example, Sir Robin the Brave might request the wedge from 17° to 42°. Each citizen will be happy if and only if they receive *precisely* the wedge that they requested. Unfortunately, some of these ranges overlap, so satisfying *all* the citizens' requests is simply impossible. Welcome to politics.

Describe and analyze an algorithm to find the maximum number of requests that can be satisfied.

5. The NSA has established several monitoring stations around the country, each one conveniently hidden in the back of a Starbucks. Each station can monitor up to 42 cell-phone towers, but can only monitor cell-phone towers within a 20-mile radius. To ensure that every cell-phone call is recorded even if some stations malfunction, the NSA requires each cell-phone tower to be monitored by at least 3 different stations.

   Suppose you know that there are n cell-phone towers and m monitoring stations, and you are given a function DISTANCE(i, j) that returns the distance between the ith tower and the jth station in $O(1)$ time. Describe and analyze an algorithm that either computes a valid assignment of cell-phone towers to monitoring stations, or reports correctly that there is no such assignment (in which case the NSA will build another Starbucks).

6. Consider the following closely related problems:

   - HAMILTONIANPATH: Given an undirected graph G, determine whether G contains a ***path*** that visits every vertex of G exactly once.
   - HAMILTONIANCYCLE: Given an undirected graph G, determine whether G contains a ***cycle*** that visits every vertex of G exactly once.

   Describe a polynomial-time reduction from HAMILTONIANCYCLE to HAMILTONIANPATH. ***Prove*** your reduction is correct. *[Hint: A polynomial-time reduction is allowed to call the black-box subroutine more than once.]*

7. An array $X[1..n]$ of distinct integers is ***wobbly*** if it alternates between increasing and decreasing: $X[i] < X[i+1]$ for every odd index i, and $X[i] > X[i+1]$ for every even index i. For example, the following 16-element array is wobbly:

12	13	0	16	13	31	5	7	−1	23	8	10	−4	37	17	42

   Describe and analyze an algorithm that permutes the elements of a given array to make the array wobbly.

You may use the following algorithms as black boxes:

RANDOM(k): Given any positive integer k, return an integer chosen independently and uniformly at random from the set $\{1, 2, \ldots, k\}$ in $O(1)$ time.

ORLINMAXFLOW(V, E, c, s, t): Given a directed graph $G = (V, E)$, a capacity function $c : E \to \mathbb{R}^+$, and vertices s and t, return a maximum (s, t)-flow in G in $O(VE)$ time. If the capacities are integral, so is the returned maximum flow.

Any other algorithm that we described in class.

You may assume the following problems are NP-hard:

CIRCUITSAT: Given a boolean circuit, are there any input values that make the circuit output TRUE?

PLANARCIRCUITSAT: Given a boolean circuit drawn in the plane so that no two wires cross, are there any input values that make the circuit output TRUE?

3SAT: Given a boolean formula in conjunctive normal form, with exactly three literals per clause, does the formula have a satisfying assignment?

MAXINDEPENDENTSET: Given an undirected graph G, what is the size of the largest subset of vertices in G that have no edges among them?

MAXCLIQUE: Given an undirected graph G, what is the size of the largest complete subgraph of G?

MINVERTEXCOVER: Given an undirected graph G, what is the size of the smallest subset of vertices that touch every edge in G?

MINSETCOVER: Given a collection of subsets $S_1, S_2, \ldots, S_m$ of a set S, what is the size of the smallest subcollection whose union is S?

MINHITTINGSET: Given a collection of subsets $S_1, S_2, \ldots, S_m$ of a set S, what is the size of the smallest subset of S that intersects every subset S_i?

3COLOR: Given an undirected graph G, can its vertices be colored with three colors, so that every edge touches vertices with two different colors?

HAMILTONIANCYCLE: Given a graph G, is there a cycle in G that visits every vertex exactly once?

HAMILTONIANPATH: Given a graph G, is there a path in G that visits every vertex exactly once?

TRAVELINGSALESMAN: Given a graph G with weighted edges, what is the minimum total weight of any Hamiltonian path/cycle in G?

STEINERTREE: Given an undirected graph G with some of the vertices marked, what is the minimum number of edges in a subtree of G that contains every marked vertex?

SUBSETSUM: Given a set X of positive integers and an integer k, does X have a subset whose elements sum to k?

PARTITION: Given a set X of positive integers, can X be partitioned into two subsets with the same sum?

3PARTITION: Given a set X of $3n$ positive integers, can X be partitioned into n three-element subsets, all with the same sum?

DRAUGHTS: Given an $n \times n$ international draughts configuration, what is the largest number of pieces that can (and therefore must) be captured in a single move?

DOGE: Such N. Many P. Wow.