CS 473: Undergraduate Algorithms, Fall 2012
Homework 0

Due Tuesday, September 4, 2012 at noon

Quiz 0 (on the course Moodle page)
is also due Tuesday, September 4, 2012 at noon.

- 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. For all future homeworks, groups of up to three students may submit (or present) a single group solution for each problem.
- Please carefully read the course policies on the course web site. If you have *any* questions, please ask in lecture, in headbanging, on Piazza, in office hours, or by email. In particular:
  - Submit separately stapled solutions, one for each numbered problem, with your name and NetID clearly printed on each page, in the corresponding drop boxes outside 1404 Siebel.
  - You may use any source at your disposal—paper, electronic, human, or other—but you ***must*** write your solutions in your own words, and you ***must*** cite every source that you use (except for official course materials). Please see the academic integrity policy for more details.
  - No late homework will be accepted for any reason. However, we may *forgive* quizzes or homeworks in extenuating circumstances; ask Jeff for details.
  - Answering "I don't know" to any (non-extra-credit) problem or subproblem, on any homework or exam, is worth 25% partial credit.
  - Algorithms or proofs containing phrases like "and so on" or "repeat this process for all n", instead of an explicit loop, recursion, or induction, will receive a score of 0.
  - Unless explicitly stated otherwise, ***every*** homework problem requires a proof.

---

1. **[CS 173]** The ***Lucas numbers*** L_n are defined recursively as follows:

$$L_n = \begin{cases} 2 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ L_{n-2} + L_{n-1} & \text{otherwise} \end{cases}$$

You may recognize this as the Fibonacci recurrence, but with a different base case ($L_0 = 2$ instead of $F_0 = 0$). Similarly, the ***anti-Lucas numbers*** Γ_n are defined recursively as follows:

$$\Gamma_n = \begin{cases} 1 & \text{if } n = 0 \\ 2 & \text{if } n = 1 \\ \Gamma_{n-2} - \Gamma_{n-1} & \text{otherwise} \end{cases}$$

Here are the first several Lucas and anti-Lucas numbers:

n	0	1	2	3	4	5	6	7	8	9	10	11	12
L_n	2	1	3	4	7	11	18	29	47	76	123	199	322
Γ_n	1	2	−1	3	−4	7	−11	18	−29	47	−76	123	−199

(a) Prove that $\Gamma_n = (-1)^{n-1} L_{n-1}$ for every positive integer n.

(b) Prove that any non-negative integer can be written as the sum of distinct *non-consecutive* Lucas numbers; that is, if L_i appears in the sum, then L_{i-1} and L_{i+1} cannot. For example:

$$\begin{aligned} 4 &= 4 &&= L_3 \\ 8 &= 7 + 1 &&= L_4 + L_1 \\ 15 &= 11 + 4 &&= L_5 + L_3 \\ 16 &= 11 + 4 + 1 &&= L_5 + L_3 + L_1 \\ 23 &= 18 + 4 + 1 &&= L_6 + L_3 + L_1 \\ 42 &= 29 + 11 + 2 &&= L_7 + L_5 + L_0 \end{aligned}$$

2. **[CS 173 + CS 373]** Consider the language over the alphabet {♠, ♥, ♦, ♣} generated by the following context-free grammar:

$$S \to ♥ \mid ♠S \mid S♣ \mid S♦S$$

Prove that every string in this language has the following properties:

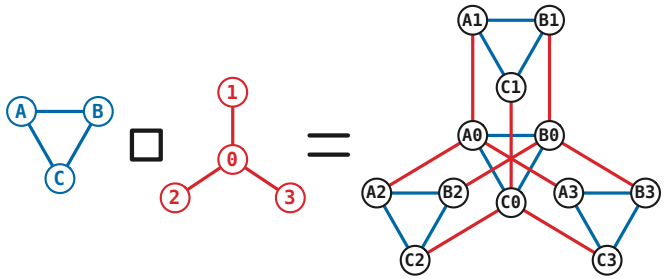
(a) The number of ♥s is exactly one more than the number of ♦s.

(b) There is a ♦ between any two ♥s.

3. **[CS 173 + mathematical maturity]** Given two undirected graphs $G = (V, E)$ and $G' = (V', E')$, we define a new graph $G \square G'$, called the ***box product*** of G and G', as follows:

   - The vertices of $G \square G'$ are all pairs (v, v') where $v \in V$ and $v' \in V'$.
   - Two vertices (v, v') and (w, w') are connected by an edge in $G \square G'$ if and only if either $\big(v = w \text{ and } (v', w') \in E'\big)$ or $\big((v, w) \in E \text{ and } v' = w'\big)$.

   Intuitively, every pair of edges $e \in E$ and $e' \in E'$ define a "box" of four edges in $G \square G'$. For example, if G is a path of length n, then $G \square G$ is an $n \times n$ grid. Another example is shown below.



The box product of two graphs.

   (a) Let I denote the unique connected graph with two vertices. Give a concise *English* description of the following graphs. You do ***not*** have to prove that your answers are correct.

      i. What is $I \square I$?
      ii. What is $I \square I \square I$?
      iii. What is $I \square I \square I \square I$?

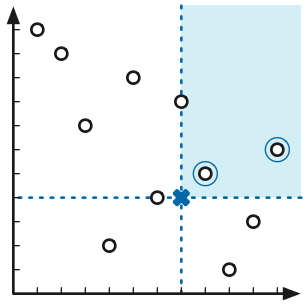
   (b) Recall that a ***Hamiltonian path*** in a graph G is a path in G that visits every vertex of G exactly once. Prove that for any graphs G and G' that both contain Hamiltonian paths, the box product $G \square G'$ also contains a Hamiltonian path. *[Hint: **Don't** use induction.]*

4. **[CS 225]** Describe and analyze a data structure that stores a set S of n points in the plane, each represented by a pair of integer coordinates, and supports queries of the following form:

   SOMETHINGABOVERIGHT(x, y): Return an arbitrary point (a, b) in S such that $a > x$ and $b > y$. If there is no such point in S, return NONE.

   For example, if S is the 11-point set $\{(1, 11), (2, 10), (3, 7), (4, 2), (5, 9), (6, 4), (7, 8), (8, 5), (9, 1), (10, 3), (11, 6)\}$, as illustrated on the next page, then

   - SOMETHINGABOVERIGHT$(0, 0)$ may return any point in S;
   - SOMETHINGABOVERIGHT$(7, 7)$ must return NONE;
   - SOMETHINGABOVERIGHT$(7, 4)$ must return either $(8, 5)$ or $(11, 6)$.



SOMETHINGABOVERIGHT(7,4) returns either (8,5) or (11,6).

A complete solution must (a) describe a data structure, (b) analyze the space it uses, (c) describe a query algorithm, (d) prove that it is correct, and (e) analyze its worst-case running time. You do *not* need to describe how to build your data structure from a given set of points. Smaller and simpler data structures with faster and simpler query algorithms are worth more points. You may assume all points in S have distinct x-coordinates and distinct y-coordinates.

*5. ***[Extra credit]*** A ***Gaussian integer*** is a complex number of the form $x + yi$, where x and y are integers. Prove that any Gaussian integer can be expressed as the sum of distinct powers of the complex number $\alpha = -1 + i$. For example:

$$\begin{aligned}
4 &= 16 + (-8 - 8i) + 8i + (-4) &&= \alpha^8 + \alpha^7 + \alpha^6 + \alpha^4 \\
-8 &= (-8 - 8i) + 8i &&= \alpha^7 + \alpha^6 \\
15i &= (-16 + 16i) + 16 + (-2i) + (-1 + i) + 1 &&= \alpha^9 + \alpha^8 + \alpha^2 + \alpha^1 + \alpha^0 \\
1 + 6i &= (8i) + (-2i) + 1 &&= \alpha^6 + \alpha^2 + \alpha^0 \\
2 - 3i &= (4 - 4i) + (-4) + (2 + 2i) + (-2i) + (-1 + i) + 1 &&= \alpha^5 + \alpha^4 + \alpha^3 + \alpha^2 + \alpha^1 + \alpha^0 \\
-4 + 2i &= (-16 + 16i) + 16 + (-8 - 8i) + (4 - 4i) + (-2i) &&= \alpha^9 + \alpha^8 + \alpha^7 + \alpha^5 + \alpha^2
\end{aligned}$$

The following list of values may be helpful:

$\alpha^0 = 1$	$\alpha^4 = -4$	$\alpha^8 = 16$	$\alpha^{12} = -64$
$\alpha^1 = -1 + i$	$\alpha^5 = 4 - 4i$	$\alpha^9 = -16 + 16i$	$\alpha^{13} = 64 - 64i$
$\alpha^2 = -2i$	$\alpha^6 = 8i$	$\alpha^{10} = -32i$	$\alpha^{14} = 128i$
$\alpha^3 = 2 + 2i$	$\alpha^7 = -8 - 8i$	$\alpha^{11} = 32 + 32i$	$\alpha^{15} = -128 - 128i$

[Hint: How do you write $-2 - i$?]

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

---

1. Describe and analyze an algorithm to reconstruct a binary search tree T, given the sequence of keys visited by a postorder traversal of T (as in Quiz 0 problem 3).

   Assume that all the input keys are distinct. Don't worry about detecting invalid inputs; the input sequence is guaranteed to be the postorder traversal of some binary search tree.

2. An array $A[0..n-1]$ of n distinct numbers is ***bitonic*** if there are unique indices i and j such that $A[(i-1) \bmod n] < A[i] > A[(i+1) \bmod n]$ and $A[(j-1) \bmod n] > A[j] < A[(j+1) \bmod n]$. In other words, a bitonic sequence either consists of an increasing sequence followed by a decreasing sequence, or can be circularly shifted to become so. For example,

4	6	9	8	7	5	1	2	3

   is bitonic, but

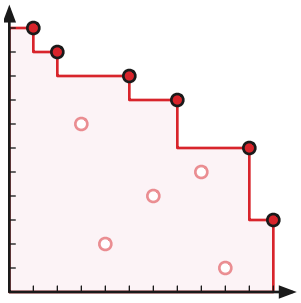
3	6	9	8	7	5	1	2	4

   is *not* bitonic.

   Describe and analyze an algorithm to find the *smallest* element in an n-element bitonic array in $O(\log n)$ time. You may assume that the numbers in the input array are distinct.

3. Let S be a set of n points in the plane. A point $p \in S$ ***maximal*** (or *Pareto-optimal*) if no point in S is both above and to the right of p. The maximal points in S intuitively define a *staircase* with all the other points in S below and to the left.

   Describe and analyze a divide-and-conquer algorithm to find all the maximal points in a given n-point set in $O(n \log n)$ time. You may assume all the input points have distinct x-coordinates and distinct y-coordinates.



Maximal points define a staircase.

*4. ***[Extra Credit]*** Describe and analyze an algorithm to find all the maximal points in a given n-point set in $\boldsymbol{O(n \log m)}$ time, where m is the number of maximal points. In particular, your algorithm should run in $O(n)$ time if the input set contains only one maximal point, and in $O(n \log n)$ time in the worst case. *[Hint: I know of at least two different ways to do this.]*

1. Describe and analyze an algorithm to determine whether a given set of positive integers can be partitioned into three disjoint subsets whose sums are equal. That is, given a set X whose elements sum to S, your algorithm should determine whether there are three disjoint subsets $A, B, C \subseteq X$ such that $A \cup B \cup C = X$ and

$$\sum_{a \in A} a = \sum_{b \in B} b = \sum_{c \in C} c = \frac{S}{3}.$$

For full credit, your algorithm should run in time polynomial in S and n.

For example, the set $\{2, 3, 4, 6, 7, 8, 9, 12\}$, can be partitioned into the subsets $\{2, 3, 12\}$, $\{4, 6, 7\}$, and $\{8, 9\}$, each of whose elements sum to 17. On the other hand, there is no balanced partition of the set $\{4, 8, 15, 16, 23, 42\}$ into three subsets.

2. Suppose Scrooge McDuck wants to split a gold chain into $n + 1$ smaller chains to distribute to his numerous nephews and nieces as their inheritance. Scrooge has carefully marked n locations where he wants the chain to be cut.

McDuck's jeweler, Fenton Crackshell, agrees to cut any chain of length ℓ inches into two smaller pieces, at any specified location, for a fee of ℓ dollars. To cut a chain into more pieces, Scrooge must pay Crackshell separately for each individual cut. As a result, the cost of breaking the chain into multiple pieces depends on the order that Crackshell makes his cuts. Obviously, Scrooge wants to pay Crackshell as little as possible.

For example, suppose the chain is 12 inches long, and the cut locations are 3 inches, 7 inches, and 9 inches from one end of the chain. If Crackshell cuts first at the 3-inch mark, then at the 7-inch mark, and then at the 9-inch mark, Scrooge must pay $12 + 9 + 5 = 26$ dollars. If Crackshell cuts the chain first at the 9-inch mark, then at the 7-inch mark, and then at the 3-inch mark, Scrooge must pay $12 + 9 + 7 = 28$ dollars. Finally, if Crackshell makes his first cut at the 7-inch mark, Scrooge only owes him $12 + 7 + 5 = 24$ dollars.

Describe and analyze an efficient algorithm that computes the minimum cost of partitioning the chain. The input to your algorithm is a sorted array $C[1..n+1]$, where $C[1]$ through $C[n]$ are the n cut positions, and $C[n+1]$ is the total length of the chain.

Rubric: For *all* dynamic programming problems in this class:

- 60% for a correct recurrence, including base cases and a plain-English description of the function. No credit for anything else if this is wrong.
- 10% for describing a suitable memoization data structure.
- 20% for describing a correct evaluation order. A clear picture is sufficient.
- 10% for analyzing the resulting algorithm's running time. It is not necessary to state a space bound.

Official solutions will always include pseudocode for the final iterative dynamic programming algorithm, but this is ***not*** required for full credit. On the other hand, if you do provide correct pseudocode for the iterative algorithm, you do not need to ***separately*** describe the recurrence, the memoization structure, or the evaluation order.

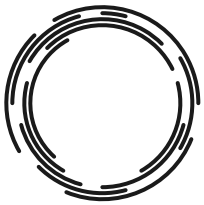
In this and all future homeworks, please clearly print the day and meeting time of *one* discussion section at the top of each page, together with the homework number, problem number, your names, and your NetIDs. We will return your graded homeworks at the discussion section you indicate. For example:

CS 473 Homework 3 Problem 2

Kyle Jao (fjao2), Nirman Kumar (nkumar5), Ernie Pan (erniepan)
Wednesday 3pm section

---

1. You are given a set A of n arcs on the unit circle, each specified by the angular coordinates of its endpoints. Describe an efficient algorithm for finding the largest possible subset X of A such that no two arcs in X intersect. Assume that all $2n$ endpoints are distinct.



2. Suppose you are given an $m \times n$ bitmap, represented by an array $M[1..m, 1..n]$ whose entries are all 0 or 1. A *checkered block* is a subarray of the form $M[i..i', j..j']$ in which no pair of adjacent entries is equal. Describe an efficient algorithm for finding the number of elements in the largest checkered block(s) in M.

$$\begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

Two checkered blocks in a 6×6 bitmap

3. A company is planning a party for its employees. The employees in the company are organized in a strict hierarchy, that is, a tree with the company president at the root. The organizers of the party have assigned a real number to each employee indicating the *awkwardness* of inviting both that employee and their immediate supervisor; a negative value indicates that the employee and their supervisor actually like each other. The organizers want to selectively invite employees to the party so that the the total awkwardness is as small as possible. For example, if the guest list does not include both an employee and their supervisor, the total awkwardness is zero.

   However, the president of the company insists on inviting exactly k employees to the party, including herself. Moreover, everyone who is invited is required to attend. Yeah, that'll be fun.

   Describe an algorithm that computes the total awkwardness of the least awkward subset of k employees to invite to the party. The input to your algorithm is an n-node rooted tree T representing the company hierarchy, an integer $awk(x)$ for each node x in T, and an integer $0 \le k \le n$. Your algorithm should return a single integer.

   For full credit, you may assume that the input is a binary tree. A complete solution for arbitrary rooted trees is worth 10 points extra credit.

4. [**Extra credit**] Two players A and B play a turn-based game on a rooted tree T. Each node v is labeled with a real number $\ell(v)$, which could be positive, negative, or zero.

   The game starts with three tokens at the root of T. In each turn, the current player moves one of the tokens from its current node down to one of its children, and the current player's score is increased by $\ell(u) \cdot \ell(v)$, where u and v are the locations of the two tokens that did *not* move. At most one token can be on any node (except the root) at any time. The game ends when the current player is unable to move, for example, if all three tokens are at leaves. The player with the higher score at the end of the game is the winner.

   Assuming that both players play optimally, describe an efficient algorithm to determine who wins on a given labeled tree. Do not assume that T is a *binary* tree.

1. Consider an n-node treap T. As in the lecture notes, we identify nodes in T by the ranks of their search keys; for example, 'node 5' means the node with the 5th smallest search key. Let i, j, k be integers such that $1 \le i \le j \le k \le n$.

   (a) What is the *exact* probability that node j is a common ancestor of node i and node k?

   (b) What is the *exact* expected length of the unique path in T from node i to node k?

   Don't forget to prove that your answers are correct!

2. A *meldable priority queue* stores a set of keys 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 element x into Q, if it is not already there.
   - DECREASEKEY(Q, x, y): Replace an element $x \in Q$ with a smaller key 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 element $x \in Q$. 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, where each node stores a key, along with pointers to its parent and 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 key(Q1) > key(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 with equal probability?]*

   (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.)

1. Suppose we can insert or delete an element into a hash table in $O(1)$ time. In order to ensure that our hash table is always big enough, without wasting a lot of memory, we will use the following global rebuilding rules:

   - After an insertion, if the table is more than 3/4 full, we allocate a new table twice as big as our current table, insert everything into the new table, and then free the old table.
   - After a deletion, if the table is less than 1/4 full, we allocate a new table half as big as our current table, insert everything into the new table, and then free the old table.

   Show that for any sequence of insertions and deletions, the amortized time per operation is still $O(1)$. *[Hint: Do not use potential functions.]*

2. Recall that a standard FIFO queue supports the following operations:

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

   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 per operation is $O(1)$.

   (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 we use the obvious algorithm to implement MULTIPULL:

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

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

   (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 we use the obvious algorithm to implement MULTIPUUSH:

```
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) 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 should use only $O(1)$ space per item.

3. (a) Describe how to implement a queue using two stacks and $O(1)$ additional memory, so that the amortized time for any enqueue or dequeue operation is $O(1)$. The ***only*** access you have to the stacks is through the standard subroutines PUSH and POP. You may not implement your own nontrivial data structure or examine the internal contents of the stacks.

   (b) A *quack* is a data structure combining properties of both stacks and queues. It can be viewed as a list of elements written left to right such that three operations are possible:

   - QUACKPUSH(x): add a new item x to the left end of the list
   - QUACKPOP(): remove and return the item on the left end of the list;
   - QUACKPULL(): remove the item on the right end of the list.

   Implement a quack using *three* stacks and $O(1)$ additional memory, so that the amortized time for any QUACKPUSH, QUACKPOP, or QUACKPULL operation is $O(1)$. In particular, each element in the quack must be stored in *exactly one* of the three stacks. Again, you are ***only*** allowed to access the component stacks through the interface functions PUSH and POP.

In both problems in this homework, your goal is to design a data structure that efficiently maintains a collection of strings (sequences of characters) subject to some or all of the following operations:

- NEWSTRING(a) creates a new string of length 1 containing only the character a and returns a pointer to that string.
- CONCAT(S, T) removes the strings S and T (given by pointers) from the data structure, adds the concatenated string ST to the data structure, and returns a pointer to the new string.
- SPLIT(S, k) removes the string S (given by a pointer) from the data structure, adds the substrings $S[1..k]$ and $S[k+1..length(S)]$ to the data structure, and returns pointers to the two new strings. You can safely assume that $1 \le k \le length(S) - 1$.
- REVERSE(S) removes the string S (given by a pointer) from the data structure, adds the reversal of S to the data structure, and returns a pointer to the new string.
- LOOKUP(S, k) returns the kth character in string S (given by a pointer), or NULL if the length of S is less than k.

For example, we can build the strings `SPLAYTREE` and `UNIONFIND` with 18 calls to NEWSTRING and 16 calls to CONCAT. Further operations modify the collection of strings as follows:

operation	result	stored strings
REVERSE(`SPLAYTREE`)	`EERTYALPS`	{**`EERTYALPS`**, `UNIONFIND`}
SPLIT(`EERTYALPS`, 5)	`EERTY`, `ALPS`	{**`EERTY`**, **`ALPS`**, `UNIONFIND`}
SPLIT(`UNIONFIND`, 3)	`UNI`, `ONFIND`	{`EERTY`, `ALPS`, **`UNI`**, **`ONFIND`**}
REVERSE(`ONFIND`)	`DNIFNO`	{`EERTY`, `ALPS`, `UNI`, **`DNIFNO`**}
CONCAT(`UNI`, `EERTY`)	`UNIEERTY`	{**`UNIEERTY`**, `ALPS`, `DNIFNO`}
SPLIT(`UNIEERTY`, 5)	`UNIEE`, `RTY`	{**`UNIEE`**, **`RTY`**, `ALPS`, `DNIFNO`}
NEWSTRING(LOOKUP(`UNIEE`, 5))	`E`	{**`E`**, `UNIEE`, `RTY`, `ALPS`, `DNIFNO`}

Except for NEWSTRING and LOOKUP, these operations are destructive; at the end of the sequence above, the string `ONFIND` is no longer stored in the data structure.

In both of the following problems, you need to describe a data structure that uses $O(n)$ space, where n is the sum of the stored string lengths, describe your algorithms for the various operations, prove that your algorithms are correct, and prove that they have the necessary running times. (Most of these steps should be very easy.)

---

1. Describe a data structure that supports the following operations:
   - SPLIT, CONCAT, and LOOKUP in $O(\log n)$ time (either worst-case, expected, or amortized).
   - NEWSTRING in $O(1)$ worst-case and amortized time.
   - ***5 points extra credit:*** REVERSE in $O(1)$ worst-case and amortized time.

2. Describe a data structure that supports the following operations:
   - CONCAT in $O(\log n)$ amortized time.
   - NEWSTRING and LOOKUP in $O(1)$ worst-case and amortized time.
   - ***5 points extra credit:*** REVERSE in $O(1)$ worst-case and amortized time.

   This data structure does not need to support SPLIT at all.

---

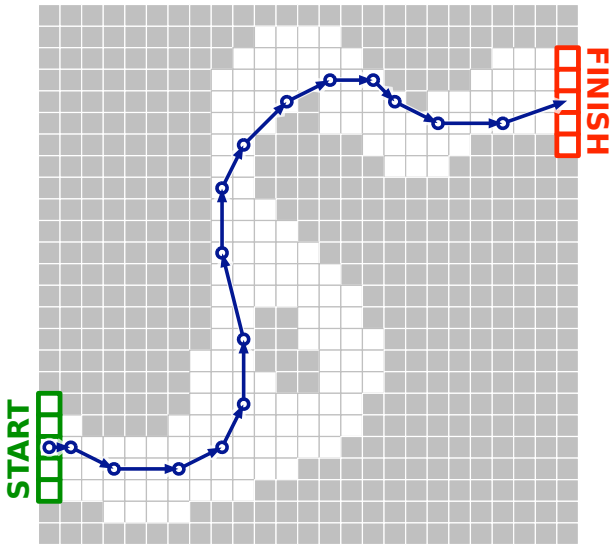
1. ***Racetrack*** (also known as *Graph Racers* and *Vector Rally*) is a two-player paper-and-pencil racing game that Jeff played on the bus in 5th grade.[1] The game is played with a track drawn on a sheet of graph paper. The players alternately choose a sequence of grid points that represent the motion of a car around the track, subject to certain constraints explained below.

   Each car has a *position* and a *velocity*, both with integer x- and y-coordinates. A subset of grid squares is marked as the *starting area*, and another subset is marked as the *finishing area*. The initial position of each car is chosen by the player somewhere in the starting area; the initial velocity of each car is always $(0, 0)$. At each step, the player optionally increments or decrements either or both coordinates of the car's velocity; in other words, each component of the velocity can change by at most 1 in a single step. The car's new position is then determined by adding the new velocity to the car's previous position. The new position must be inside the track; otherwise, the car crashes and that player loses the race. The race ends when the first car reaches a position inside the finishing area.

   Suppose the racetrack is represented by an $n \times n$ array of bits, where each 0 bit represents a grid point inside the track, each 1 bit represents a grid point outside the track, the 'starting area' is the first column, and the 'finishing area' is the last column.

   Describe and analyze an algorithm to find the minimum number of steps required to move a car from the starting area to the finishing area of a given racetrack.

velocity	position
$(0,0)$	$(1,5)$
$(1,0)$	$(2,5)$
$(2,-1)$	$(4,4)$
$(3,0)$	$(7,4)$
$(2,1)$	$(9,5)$
$(1,2)$	$(10,7)$
$(0,3)$	$(10,10)$
$(-1,4)$	$(9,14)$
$(0,3)$	$(9,17)$
$(1,2)$	$(10,19)$
$(2,2)$	$(12,21)$
$(2,1)$	$(14,22)$
$(2,0)$	$(16,22)$
$(1,-1)$	$(17,21)$
$(2,-1)$	$(19,20)$
$(3,0)$	$(22,20)$
$(3,1)$	$(25,21)$



A 16-step Racetrack run, on a 25×25 track. This is *not* the shortest run on this track.

[1] The actual game is a bit more complicated than the version described here. See http://harmmade.com/vectorracer/ for an excellent online version.

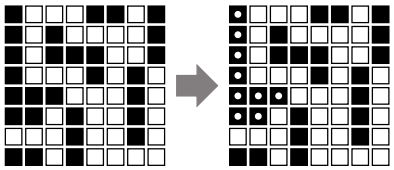
2. An ***Euler tour*** of a graph G is a walk that starts and ends at the same vertex and traverses every edge of G exactly once.

   (a) Prove that a connected undirected graph G has an Euler tour if and only if every vertex in G has even degree.

   (b) Describe and analyze an algorithm that constructs an Euler tour of a given graph G, or correctly reports that no such tour exists.

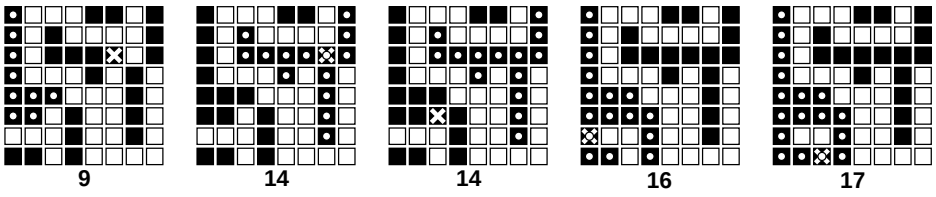
3. (a) Describe and analyze an algorithm to compute the size of the largest connected component of black pixels in a given $n \times n$ bitmap $B[1..n, 1..n]$.

   For example, given the bitmap below as input, your algorithm should return the number 9, because the largest conected black component (marked with white dots on the right) contains nine pixels.



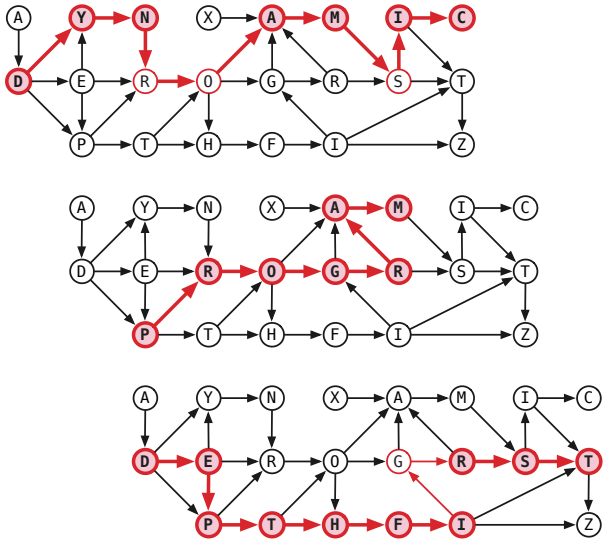
   (b) Design and analyze an algorithm BLACKEN(i, j) that colors the pixel $B[i, j]$ black and returns the size of the largest black component in the bitmap. For full credit, the *amortized* running time of your algorithm (starting with an all-white bitmap) must be as small as possible.

   For example, at each step in the sequence below, we blacken the pixel marked with an X. The largest black component is marked with white dots; the number underneath shows the correct output of the BLACKEN algorithm.



   (c) What is the *worst-case* running time of your BLACKEN algorithm?

1. Let G be a directed acyclic graph where each node has a label from some finite alphabet; different nodes may have the same label. Any directed path in G has a ***signature***, which is the string defined by concatenating the labels of its vertices. A ***subsequence*** of G is a subsequence of the signature of some directed path in G. For example, the strings `DYNAMIC`, `PROGRAM`, and `DETPHFIRST` are all subsequences of the dag shown below; in fact, `PROGRAM` is the signature of a path.



   Describe and analyze an algorithm to compute the length of the longest common subsequence of two given directed acyclic graphs, that is, the longest string that is a subsequence of both dags.

2. Suppose you are given a graph G with weighted edges and a minimum spanning tree T of G.

   (a) Describe an algorithm to update the minimum spanning tree when the weight of a single edge e is decreased.

   (b) Describe an algorithm to update the minimum spanning tree when the weight of a single edge e is increased.

   In both cases, the input to your algorithm is the edge e and its new weight; your algorithms should modify T so that it is still a minimum spanning tree. *[Hint: Consider the cases $e \in T$ and $e \notin T$ separately.]*

3. When there is more than one shortest paths from one node s to another node t, it is often most convenient to choose the shortest path with the fewest edges; call this the ***best path*** from s to t. For instance, if nodes represent cities and edge lengths represent costs of flying between cities, there could be many ways to fly from city s to city t for the same cost; the most desirable these schedules is the one with the fewest flights.

   Suppose we are given a directed graph G with positive edge weights and a source vertex s in G. Describe and analyze an algorithm to compute *best* paths in G from s to every other vertex. *[Hint: What is the actual output of your algorithm? If possible, use one of the standard shortest-path algorithms as a black box.]*

Orlin's algorithm computes maximum flows in $O(VE)$ time.
You may use this algorithm as a black box.

1. Suppose you are given a flow network G with ***integer*** edge capacities and an ***integer*** maximum flow f^* in G. Describe algorithms for the following operations:

   (a) INCREMENT(e): Increase the capacity of edge e by 1 and update the maximum flow.

   (b) DECREMENT(e): Decrease the capacity of edge e by 1 and update the maximum flow.

   Both algorithms should modify f^* so that it is still a maximum flow, more quickly than recomputing a maximum flow from scratch.

2. Suppose we are given a set of boxes, each specified by its height, width, and depth in centimeters. All three dimensions of each box lie strictly between 25cm and 50cm; box dimensions are *not* necessarily integers. As you might expect, one box can be placed inside another box if, possibly after rotating one or both boxes, each dimension of the first box is smaller than the corresponding dimension of the second box. Boxes can be nested recursively.

   Call a box is *visible* if it is not inside another box. Describe and analyze an algorithm to nest the boxes so that the number of visible boxes is as small as possible.

3. The University of Southern North Dakota at Hoople has hired you to write an algorithm to schedule their final exams. Each semester, USNDH offers n different classes. There are r different rooms on campus and t different time slots in which exams can be offered. You are given two arrays $E[1..n]$ and $S[1..r]$, where $E[i]$ is the number of students enrolled in the ith class, and $S[j]$ is the number of seats in the jth room. At most one final exam can be held in each room during each time slot. Class i can hold its final exam in room j only if $E[i] < S[j]$.

   Describe and analyze an efficient algorithm to assign a room and a time slot to each class (or report correctly that no such assignment is possible).

This homework is optional. Any problem that you do not submit will be automatically forgiven.

A useful list of NP-hard problems appears on the next page.

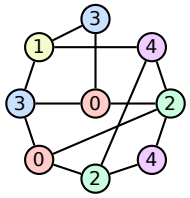
---

1. In the task scheduling problem, we are given n tasks, identified by the integers 1 through n, and a set of precedence constraints of the form "Task i must be executed before task j." A ***feasible schedule*** is an ordering of the n tasks that satisfies all the given precedence constraints.

   (a) Given a set of tasks and precedence constraints, describe and analyze a polynomial-time algorithm to determine whether a feasible schedule exists.

   (b) Suppose we are given a set of precedence constraints for which there is no feasible schedule. In this case, we would like a schedule that violates the minimum number of precedence constraints. Prove that finding such a schedule is NP-hard.

2. Recall that a 5-coloring of a graph G is a function that assigns each vertex of G a 'color' from the set $\{0, 1, 2, 3, 4\}$, such that for any edge uv, vertices u and v are assigned different 'colors'. A 5-coloring is *careful* if the colors assigned to adjacent vertices are not only distinct, but differ by more than 1 (mod 5). Prove that deciding whether a given graph has a careful 5-coloring is NP-hard. *[Hint: Reduce from the standard* 5COLOR *problem.]*



A careful 5-coloring.

3. (a) A ***tonian path*** in a graph G is a path that goes through at least half of the vertices of G. Show that determining whether a given graph has a tonian path is NP-hard.

   (b) A ***tonian cycle*** in a graph G is a cycle that goes through at least half of the vertices of G. Show that determining whether a given graph has a tonian cycle is NP-hard. *[Hint: Use part (a).]*

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?

MAX2SAT: Given a boolean formula in conjunctive normal form, with exactly two literals per clause, what is the largest number of clauses that can be satisfied by an 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?

MAXCUT: Given a graph G, what is the size (number of edges) of the largest bipartite subgraph of G?

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?

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 n positive integers, can X be partitioned into $n/3$ three-element subsets, all with the same sum?

CS 473: Undergraduate Algorithms, Fall 2012

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 any integer can be represented as the sum of distinct powers of -2.

3. Write ***four different*** proofs that any n-node tree has exactly $n - 1$ edges.

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.

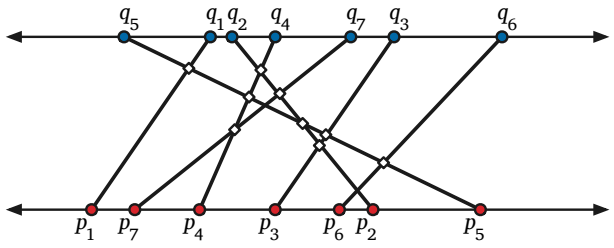
Khelm is Warsaw. Warsaw is Khelm. Khelm is Warsaw. Zay gezunt!
Warsaw is Khelm. Khelm is Warsaw. Warsaw is Khelm. For gezunt!
— Golem (feat. Amanda Palmer), "Warsaw is Khelm", *Fresh Off Boat* (2006)

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.]*

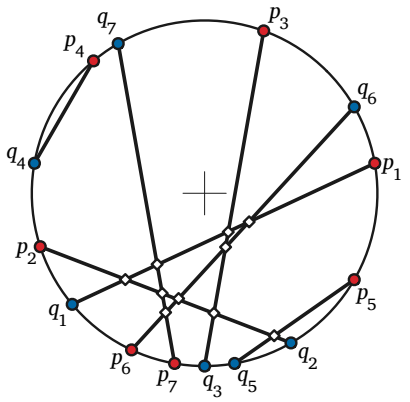
   Assume a reasonable representation for the input points, and assume the x-coordinates of the input points are distinct. For example, for the input shown below, your algorithm should return the number 9.



Nine 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 the previous problem in $O(n \log n)$ time.

A subsequence is anything obtained from a sequence by deleting a subset of elements; the elements of the subsequence need not be contiguous in the original sequence. For example, the strings `I`, `PRO`, `DAMMM`, `NROAIG`, and `DYNAMICPROGRAMMING` are all subsequences of the string `DYNAMICPROGRAMMING`.

1. Suppose we are given two arrays $A[1..m]$ and $B[1..m]$. A *common subsequence* of A and B is any subsequence of A that is also a subsequence of B. For example, `AMI` is a common subsequence of `DYNAMIC` and `PROGRAMMING`. Describe and analyze an efficient algorithm to compute the length of the *longest* common subsequence of A and B.

2. Describe and analyze an efficient algorithm to compute the length of the longest common subsequence of *three* given arrays $A[1..l]$, $B[1..m]$, and $C[1..n]$.

3. A *shuffle* of two strings X and Y is formed by interspersing the characters into a new string, such that the characters of X and Y remain in the same order. For example, given the strings `'dynamic'` and `'programming'`, the string `'prodgyrnamammiincg'` is indeed a shuffle of the two:

$$\texttt{pro}^{\texttt{d}}\texttt{g}^{\texttt{y}}\texttt{r}^{\texttt{nam}}\texttt{ammi}^{\texttt{i}}\texttt{n}^{\texttt{c}}\texttt{g}$$

   Given three strings $A[1..m], B[1..n]$ and $C[1..m+n]$, describe an algorithm to determine whether C is a shuffle of A and B.

In all cases, first describe a recursive algorithm and only then transform it into an iterative algorithm.

1. A set of vectors A is said to be linearly independent if no $v \in A$ can be expressed as a linear combination of the vectors in $A - \{v\}$. Given a set of vectors S, describe an efficient algorithm for finding a linearly independent subset of S with the maximum possible size. Assume you are given a function that can check if n vectors are linearly independent in $O(n^2)$ time.

2. You live in a country with n different types of coins, with values $1, 2, 2^2, ..., 2^{n-1}$. Describe an efficient algorithm for determining how to make change for a given value W using the least possible number of coins.

3. Let X be a set of n intervals on the real line. A *proper coloring* of X assigns a color to each interval, so that any two overlapping intervals are assigned different colors. Describe an efficient algorithm to compute the minimum number of colors needed to properly color X. Assume that your input consists of two array $L[1..n]$ and $R[1..n]$, where $L[i]$ and $R[i]$ are the left and right endpoints of the ith interval.

1. Any connected graph with n nodes and n edges has exactly one cycle.

2. Any n-node binary tree can be transformed into any other n-node binary tree by a sequence of at most $2n - 2$ rotations.

3. If $d_1, \ldots, d_n$ are positive integers such that $\sum_{i=1}^{n} d_i = 2n - 2$, then there is a tree having $d_1, \ldots, d_n$ as its vertex degrees. For examples, $\{1, 1, 1, 1, 1, 5\}$ has sum $2 \cdot 6 - 2$, and so the hypothesis is satisfied. The tree that is the star with five leaves has vertex degrees $\{1, 1, 1, 1, 1, 5\}$. Also, $\{1, 1, 1, 1, 2, 3, 3\}$ has sum $2 \cdot 7 - 2$, and the perfect binary three with depth 2 has vertex degrees $\{1, 1, 1, 1, 2, 3, 3\}$.

1. Prove that the expected space requirement of a skip list constructed on n numbers is $O(n)$.

2. Let S be a set of n points in the plane. A point p in S is called ***maximal*** (or *Pareto-optimal*) if no other point in S is both above and to the right of p. If each point in S is chosen independently and uniformly at random from the unit square $[0,1] \times [0,1]$ what is the *exact* expected number of Pareto-optimal points in S.

3. A *data stream* is an extremely long sequence of items that you can read only once. A data stream algorithm looks roughly like this:

```
DoSomethingInteresting(stream S) :
repeat
  x ← next item in S
  ⟨⟨ do something fast with x ⟩⟩
until S ends
return ⟨⟨ something ⟩⟩
```

   Describe and analyze an algorithm that chooses one element uniformly at random from a data stream, *without knowing the length of the stream in advance*. Your algorithm should spend $O(1)$ time per stream element and use $O(1)$ space (not counting the stream itself).

1. An ***extendable array*** is a data structure that stores a sequence of items and supports the following operations.

   - AddToFront(x) adds x to the *beginning* of the sequence.
   - AddToEnd(x) adds x to the *end* of the sequence.
   - Lookup(k) returns the kth item in the sequence, or Null if the current length of the sequence is less than k.

   Describe a ***simple*** data structure that implements an extendable array. Your AddToFront and AddToBack algorithms should take $O(1)$ *amortized* time, and your Lookup algorithm should take $O(1)$ *worst-case* time. The data structure should use $O(n)$ space, where n is the ***current*** length of the sequence.

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. Chicago has many tall buildings, but only some of them have a clear view of Lake Michigan. Suppose we are given an array $A[1..n]$ that stores the height of n buildings on a city block, indexed from west to east. Building i has a good view of Lake Michigan if and only if every building to the east of i is shorter than i.

   Here is an algorithm that computes which buildings have a good view of Lake Michigan. What is the running time of this algorithm?

```
GoodView(A[1..n]):
  initialize a stack S
  for i ← 1 to n
      while (S not empty and A[i] > A[Top(S)])
          Pop(S)
      Push(S, i)
  return S
```

1. Suppose we want to maintain a dynamic set of numbers, subject to the following operations:

   - INSERT(x): Add x to the set. (Assume x is not already in the set.)
   - PRINT&DELETEBETWEEN(a, b): Print every element x in the range $a \le x \le b$ in increasing order, and then delete those elements from the set.

   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 $\{1, 3, 8\}$.
   - PRINT&DELETEBETWEEN$(6, 7)$ prints nothing and does not change the set.
   - PRINT&DELETEBETWEEN$(0, 10)$ prints the sequence $1, 3, 4, 5, 8$ and deletes everything.

   Describe a data structure that supports both operations in $O(\log n)$ amortized time, where n is the *current* number of elements in the set.

   [Hint: As warmup, argue that the obvious implementation of PRINT&DELETEBETWEEN*—while the successor of a is less than or equal to b, print it and delete it—runs in $O(\log N)$ amortized time, where N is the* ***maximum*** *number of elements that are ever in the set.]*

2. Describe a data structure that stores a set of numbers (which is initially empty) and supports the following operations in $O(1)$ amortized time:

   - INSERT(x): Insert x into the set. (You can safely assume that x is not already in the set.)
   - FINDMIN: Return the smallest element of the set (or NULL if the set is empty).
   - DELETEBOTTOMHALF: Remove the smallest $\lceil n/2 \rceil$ elements the set. (That's smallest by *value*, not smallest by *insertion time*.)

3. Consider the following solution for the union-find problem, called *union-by-weight*. Each set leader $\overline{x}$ stores the number of elements of its set in the field $weight(\overline{x})$. 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).

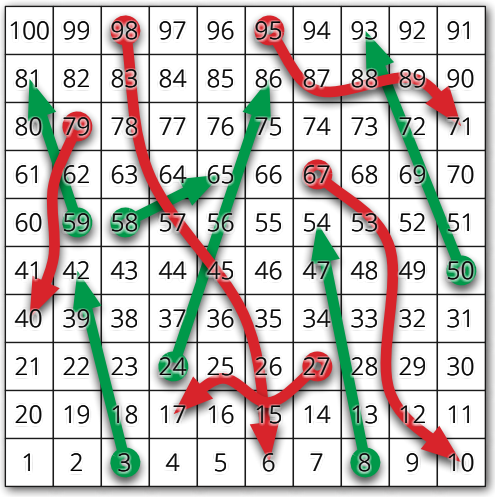
```
MAKESET(x):
  parent(x) ← x
  weight(x) ← 1
```

```
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 always 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. ***Snakes and Ladders*** is a classic board game, originating in India no later than the 16th century. The board consists of an $n \times n$ grid of squares, numbered consecutively from 1 to n^2, starting in the bottom left corner and proceeding row by row from bottom to top, with rows alternating to the left and right. Certain pairs of squares in this grid, always in different rows, are connected by either "snakes" (leading down) or "ladders" (leading up). Each square can be an endpoint of at most one snake or ladder.



A typical Snakes and Ladders board.
Upward straight arrows are ladders; downward wavy arrows are snakes.

   You start with a token in cell 1, in the bottom left corner. In each move, you advance your token up to k positions, for some fixed constant k. If the token ends the move at the *top* end of a snake, it slides down to the bottom of that snake. Similarly, if the token ends the move at the *bottom* end of a ladder, it climbs up to the top of that ladder.

   Describe and analyze an algorithm to compute the smallest number of moves required for the token to reach the last square of the grid.

2. Suppose you are given a set of n jobs, indexed from 1 to n, together with a list of *precedence constraints*. Each precedence constraint is a pair (i, j), indicating that job i must be finished before job j begins. Describe and analyze an algorithm that either finds an schedule for executing all n jobs on a single processor, such that all precedence constraints are satisfied, or correctly reports that no such schedule is possible.

3. For the previous problem, describe and analyze an algorithm to determine whether there is a *unique* schedule for a given number of jobs that satisfies a given set of precedence constraints.

1. Let $G = (V, E)$ be a directed graph. If the indedgree of each vertex is at least 1 prove that G contains a cycle. Give an algorithm to find a cycle in such a graph. How fast is your algorithm?

2. At a party with n people $P_1, \ldots, P_n$, certain pairs of individuals cannot stand each other. Given a list of such pairs, determine if we can divide the n people into two groups such that all the people in both group are amicable, that is, they can stand each other.

3. Given a graph $G = (V, E)$ prove that one can orient each edge so that after the orientation, the indegree and outdegree of each vertex differ by at most 1. How fast can you compute this orientation? **[Hint: Can you change the graph so that it has an Euler tour?]**

1. Describe and analyze an algorithm that finds a *maximum* spanning tree of a graph, that is, the spanning tree with largest total weight.

2. During your CS 473 final exam, you are given a specific undirected graph $G = (V, E)$ with non-negative edge weights, and you are asked to compute both the minimum spanning tree of G and the tree of shortest paths in G rooted at some fixed source vertex s.

   Two and a half hours into the exam, Jeff announces that there was a mistake in the exam; every edge weight should be increased by 1. Well, that's just great. Now what?!

   (a) Do you need to recompute the minimum spanning tree? Either prove that increasing all edge weights by 1 cannot change the minimum spanning tree, or give an example where the minimum spanning tree changes.

   (b) Do you need to recompute the shortest path tree rooted at s? Either prove that increasing all edge weights by 1 cannot change the shortest path tree, or give an example where the shortest path tree changes.

3. After graduating you accept a job with Aerophobes-Я-Us, the leading traveling agency for people who hate to fly. Your job is to build a system to help customers plan airplane trips from one city to another. All of your customers are afraid of flying (and by extension, airports), so any trip you plan needs to be as short as possible. You know all the departure and arrival times of every flight on the planet.

   Suppose one of your customers wants to fly from city X to city Y. Describe an algorithm to find a sequence of flights that minimizes the *total time in transit*—the length of time from the initial departure to the final arrival, including time at intermediate airports waiting for connecting flights. *[Hint: Modify the input data and apply Dijkstra's algorithm.]*

1. Suppose you are given the following information:

   - A directed graph $G = (V, E)$.
   - Two vertices s and t in V.
   - A positive edge capacity function $c : E \to \mathbb{R}^+$.
   - Another function $f : E \to \mathbb{R}$

   Describe and analyze an algorithm to determine whether f is a maximum (s, t)-flow in G.

2. Describe an efficient algorithm to determine whether a given flow network contains a *unique* maximum flow.

3. Suppose that a flow network has vertex capacities in addition to edge capacities. That is, the total amount of flow into or out of any vertex v is at most the capacity of v:

$$\sum_u f(u \to v) \;=\; \sum_w f(v \to w) \;\le\; c(v)$$

   Describe and analyze an algorithm to compute maximum flows with this additional constraint.

1. The UIUC Faculty Senate has decided to convene a committee to determine whether Chief Illiniwek should become the official ~~mascot~~ *symbol* of the University of Illinois Global Campus.[1] Exactly one faculty member must be chosen from each academic department to serve on this committee. Some faculty members have appointments in multiple departments, but each committee member will represent only one department. For example, if Prof. Blagojevich is affiliated with both the Department of Corruption and the Department of Stupidity, and he is chosen as the Stupidity representative, then someone else must represent Corruption. Finally, University policy requires that any committee on virtual ~~mascots~~ *symbols* must contain the same number of assistant professors, associate professors, and full professors. Fortunately, the number of departments is a multiple of 3.

   Describe an efficient algorithm to select the membership of the Global Illiniwek Committee. Your input is a list of all UIUC faculty members, their ranks (assistant, associate, or full), and their departmental affiliation(s). There are n faculty members and $3k$ departments.

2. *Ad-hoc networks* are made up of low-powered wireless devices. In principle[2], these networks can be used on battlefields, in regions that have recently suffered from natural disasters, and in other hard-to-reach areas. The idea is that a large collection of cheap, simple devices could be distributed through the area of interest (for example, by dropping them from an airplane); the devices would then automatically configure themselves into a functioning wireless network.

   These devices can communicate only within a limited range. We assume all the devices are identical; there is a distance D such that two devices can communicate if and only if the distance between them is at most D.

   We would like our ad-hoc network to be reliable, but because the devices are cheap and low-powered, they frequently fail. If a device detects that it is likely to fail, it should transmit its information to some other *backup* device within its communication range. We require each device x to have k potential backup devices, all within distance D of x; we call these k devices the ***backup set*** of x. Also, we do not want any device to be in the backup set of too many other devices; otherwise, a single failure might affect a large fraction of the network.

   So suppose we are given the communication radius D, parameters b and k, and an array $d[1..n, 1..n]$ of distances, where $d[i, j]$ is the distance between device i and device j. Describe an algorithm that either computes a backup set of size k for each of the n devices, such that no device appears in more than b backup sets, or reports (correctly) that no good collection of backup sets exists.

3. 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.

[1] Thankfully, the Global Campus has faded into well-deserved obscurity, thanks in part to the 2009 admissions scandal. Imagine MOOCs, but with the same business model and faculty oversight as the University of Phoenix.

[2] but not really in practice

A useful list of NP-hard problems appears on the next page.

The KNAPSACK problem is the following. We are given a set of n objects, each with a positive integer ***size*** and a positive integer *value*; we are also given a positive integer B. The problem is to choose a subset of the n objects with maximum total value, whose total size is at most B. Let V denote the sum of the values of all objects.

1. Describe an algorithm to solve KNAPSACK in time polynomial in n and V.

2. Prove that the KNAPSACK problem is NP-hard.

   Given the algorithm from problem 1, why doesn't this immediately imply that P=NP?

---

3. ***Facility location*** is a family of problems that require choosing a subset of facilities (for example, gas stations, cell towers, garbage dumps, Starbuckses, ...) to serve a given set of locations cheaply. In its most abstract formulation, the input to the facility location problem is a pair of arrays $Open[1..n]$ and $Connect[1..n, 1..m]$, where

   - $Open[i]$ is the cost of opening a facility i, and
   - $Connect[i, j]$ is the cost of connecting facility i to location j.

   Given these two arrays, the problem is to compute a subset $I \subseteq \{1, 2, \ldots, n\}$ of the n facilities to open and a function $\phi : \{1, 2, \ldots, m\} \to I$ that assigns an open facility to each of the m locations, so that the total cost

   $$\sum_{i \in I} Open[i] + \sum_{j=1}^{m} Connect[\phi(j), j]$$

   is minimized. Prove that this problem is NP-hard.

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?

MAX2SAT: Given a boolean formula in conjunctive normal form, with exactly two literals per clause, what is the largest number of clauses that can be satisfied by an 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?

MAXCUT: Given a graph G, what is the size (number of edges) of the largest bipartite subgraph of G?

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?

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 n positive integers, can X be partitioned into $n/3$ three-element subsets, all with the same sum?

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 $\dfrac{n^3 + 3n^2 - 5n + 1}{4n^2 - 2n + \sqrt{3}}$?

   (b) What is $\displaystyle\sum_{i=1}^{n}\sum_{j=1}^{i} 5$?

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

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

   (e) What is the solution to the recurrence $E(n) = E(n-3) + 2n - 1$?

   (f) What is the solution to the recurrence $F(n) = 2F(n/3) + 2F(n/6) + n$?

   (g) What is the solution to the recurrence $G(n) = 12\,G(n/4) + n^2$?

   (h) What is the worst-case depth of an n-node binary tree?

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

   $$WTF(i,j) = \begin{cases} 0 & \text{if } i \le 0 \text{ or } j \le 0 \\ X[j] + WTF(i-1,j) + WTF(i,\lfloor j/2 \rfloor) & \text{otherwise} \end{cases}$$

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

   (j) How many seconds does it take for a 10-megapixel image taken by the Curiosity Rover to be encoded, transmitted from Mars to Earth, decoded, and tweeted?

2. A palindrome is any string that is exactly the same as its reversal, like `I`, or `DEED`, or `HANNAH`, or `SATOR AREPO TENET OPERA ROTAS`. Describe and analyze an algorithm to find the length of the longest subsequence of a given string that is also a palindrome.

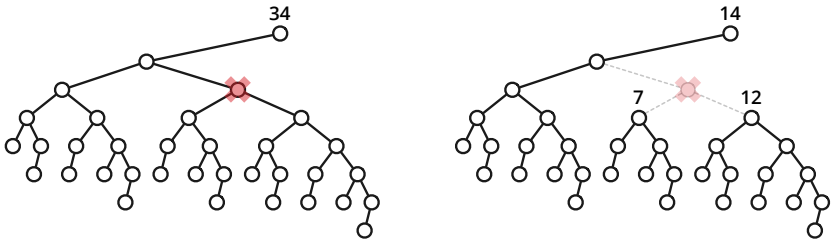
   For example, the longest palindrome subsequence of `MAHDYNAMICPROGRAMZLETMESHOWYOUTHEM` is `MHYMRORMYHM`, so given that string as input, your algorithm should return the integer 11.

3. ***Prove*** that any integer—positive, negative, or zero—can be represented as the sum of distinct powers of -2. For example:

$$\begin{aligned}
4 &= 4 &&= (-2)^2 \\
8 &= 16-8 &&= (-2)^4+(-2)^3 \\
15 &= 16-2+1 &&= (-2)^4+(-2)^1+(-2)^0 \\
-16 &= -32+16 &&= (-2)^5+(-2)^4 \\
23 &= 64-32-8-2+1 &&= (-2)^6+(-2)^5+(-2)^3+(-2)^1+(-2)^0 \\
-42 &= -32-8-2 &&= (-2)^5+(-2)^3+(-2)^1
\end{aligned}$$

4. Let T be a binary tree with n vertices. Deleting any vertex v splits T into at most three subtrees, containing the left child of v (if any), the right child of v (if any), and the parent of v (if any). We call v a ***central*** vertex if each of these smaller trees has at most $n/2$ vertices.

   Describe and analyze an algorithm to find a central vertex in a given binary tree.



Deleting a central vertex in a 34-node binary tree,
leaving subtrees with 14 nodes, 7 nodes, and 12 nodes.

5. Let $n = 2^\ell - 1$ for some positive integer ℓ. Suppose someone claims to hold an array $A[1..n]$ of *distinct* ℓ-bit strings; thus, exactly one ℓ-bit string does *not* appear in A. Suppose further that the ***only*** way we can access A is by calling the function FETCHBIT(i, j), which returns the jth bit of the string $A[i]$ in $O(1)$ time.

   Describe an algorithm to find the missing string in A using only $O(n)$ calls to FETCHBIT.

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.

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

---

1. (a) The *left spine* of a binary tree is the path from the root to the leftmost node. For example, if the root has no left child, the left spine contains only the root. What is the expected number of nodes in the left spine of an n-node treap? *[Hint: What is the probability that the node with kth smallest search key is in the left spine?]*

   (b) What is the expected number of leaves in an n-node treap? *[Hint: What is the probability that the node with kth smallest search key is a leaf?]*

   You do ***not*** need to prove that your answers are correct.

2. Recall that a *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 support all three operations in $O(1)$ worst-case time, using a doubly-linked list and a counter. Now 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 $O(1)$.

3. 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.

4. Consider the following modification of the standard algorithm for incrementing a binary counter.

```
INCREMENT(B[0..∞]):
  i ← 0
  while B[i] = 1
      B[i] ← 0
      i ← i + 1
  B[i] ← 1
  SOMETHINGELSE(i)
```

The only difference from the standard algorithm is the function call at the end, to a black-box subroutine called SOMETHINGELSE.

Suppose we call INCREMENT n times, starting with a counter with value 0. The amortized time of each INCREMENT clearly depends on the running time of SOMETHINGELSE. Let $T(i)$ denote the worst-case running time of SOMETHINGELSE(i). For example, we proved in class that INCREMENT algorithm runs in $O(1)$ amortized time when $T(i) = 0$.

(a) What is the amortized time per INCREMENT if $T(i) = 42$?

(b) What is the amortized time per INCREMENT if $T(i) = 2^i$?

(c) What is the amortized time per INCREMENT if $T(i) = 4^i$?

(d) What is the amortized time per INCREMENT if $T(i) = \sqrt{2}^i$?

(e) What is the amortized time per INCREMENT if $T(i) = 2^i/(i+1)$?

You do ***not*** need to prove your answers are correct.

5. A *data stream* is a long sequence of items that you can only read only once, in order. Every data-stream algorithm looks roughly like this:

```
DOSOMETHINGINTERESTING(stream S):
  repeat
      x ← next item in S
      ⟨⟨do something fast with x⟩⟩
  until S ends
  return ⟨⟨something⟩⟩
```

Describe and analyze an algorithm that chooses one element uniformly at random from a data stream.

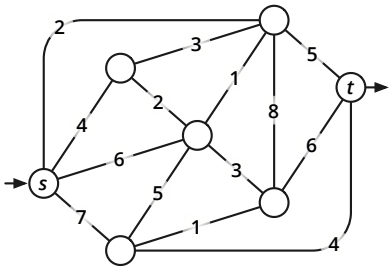
Your algorithm should spend $O(1)$ time per stream element and use only $O(1)$ space (excluding the stream itself). ***You do not know the length of the stream in advance***; your algorithm learns that the stream has ended only when a request for the next item fails.

You do ***not*** need to prove your algorithm is correct.

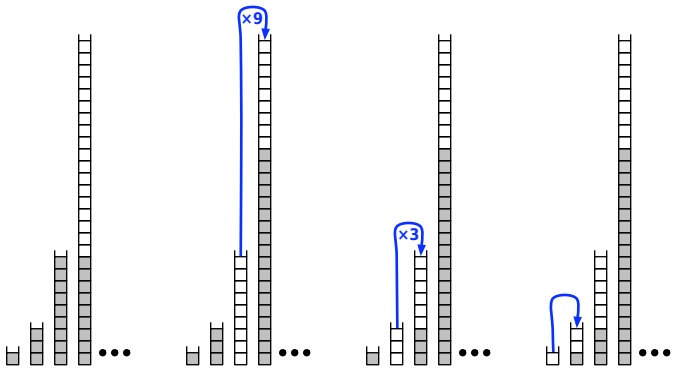
This exam lasts 180 minutes.
Write your answers in the separate answer booklet.
Please return this question sheet and your cheat sheets with your answers.

1. Clearly indicate the following structures 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 minimum (s,t)-cut



2. A *multistack* consists of an infinite series of stacks $S_0, S_1, S_2, \ldots$, where the ith stack S_i can hold up to 3^i elements. Whenever a user attempts to push an element onto any full stack S_i, we first pop all the elements off S_i and push them onto stack S_{i+1} to make room. (Thus, if S_{i+1} is already full, we first recursively move all its members to S_{i+2}.) Moving a single element from one stack to the next takes $O(1)$ time.



Making room for one new element in a multistack.

   (a) In the worst case, how long does it take to push one more element onto a multistack containing n elements?
   (b) ***Prove*** that the amortized cost of a push operation is $O(\log n)$, where n is the maximum number of elements in the multistack.

3. Suppose we are given an array $A[1..n]$ of numbers with the special property that $A[1] \geq A[2]$ and $A[n-1] \leq A[n]$. A ***local minimum*** is an element $A[i]$ such that $A[i-1] \geq A[i]$ and $A[i] \leq A[i+1]$. For example, there are six local minima in the following array:

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

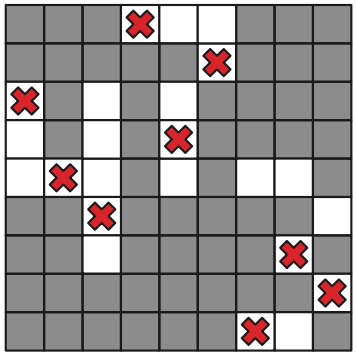
   Describe and analyze an algorithm that finds a local minimum in the array A in $O(\log n)$ time.

4. Suppose we are given an n-digit integer X. Repeatedly remove one digit from either end of X (your choice) until no digits are left. The *square-depth* of X is the maximum number of perfect squares that you can see during this process. For example, the number 32492 has square-depth 3, by the following sequence of removals:

$$32492 \to \underline{3249}\not{2} \to \underline{324}\not{9} \to \not{3}24 \to \not{2}\underline{4} \to \not{4}.$$

Describe and analyze an algorithm to compute the square-depth of a given integer X, represented as an array $X[1..n]$ of n decimal digits. Assume you have access to a subroutine IsSquare that determines whether a given k-digit number (represented by an array of digits) is a perfect square ***in $O(k^2)$ time***.

5. Suppose we are given an $n \times n$ square grid, some of whose squares are colored black and the rest white. Describe and analyze an algorithm to determine whether tokens can be placed on the grid so that



- every token is on a white square;
- every row of the grid contains exactly one token; and
- every column of the grid contains exactly one token.

Your input is a two dimensional array $IsWhite[1..n, 1..n]$ of booleans, indicating which squares are white. (You solved an instance of this problem in the last quiz.)

6. Recall the following problem from Homework 2:

- 3WayPartition: Given a set X of positive integers, determine whether there are three disjoint subsets $A, B, C \subseteq X$ such that $A \cup B \cup C = X$ and

$$\sum_{a \in A} a = \sum_{b \in B} b = \sum_{c \in C} c.$$

(a) ***Prove*** that 3WayPartition is NP-hard. *[Hint: Don't try to reduce from 3SAT or 3COLOR; in this rare instance, the 3 is just a coincidence.]*

(b) In Homework 2, you described an algorithm to solve 3WayPartition in $O(nS^2)$ time, where S is the sum of all elements of X. Why doesn't this algorithm imply that P=NP?

7. Describe and analyze efficient algorithms to solve the following problems:

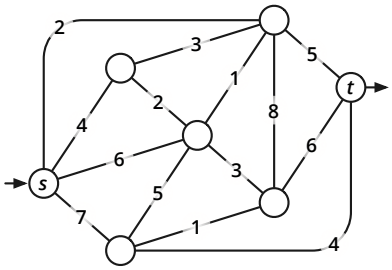
(a) Given an array of n integers, does it contain two elements a, b such that $a + b = 0$?

(b) Given an array of n integers, does it contain three elements a, b, c such that $a + b + c = 0$?

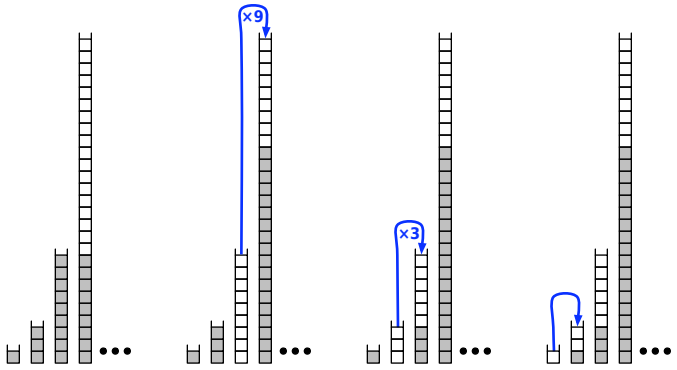
This exam lasts 180 minutes.
Write your answers in the separate answer booklet.
Please return this question sheet and your cheat sheets with your answers.

1. Clearly indicate the following structures 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 minimum (s, t)-cut



2. A *multistack* consists of an infinite series of stacks $S_0, S_1, S_2, \ldots$, where the ith stack S_i can hold up to 3^i elements. Whenever a user attempts to push an element onto any full stack S_i, we first pop all the elements off S_i and push them onto stack S_{i+1} to make room. (Thus, if S_{i+1} is already full, we first recursively move all its members to S_{i+2}.) Moving a single element from one stack to the next takes $O(1)$ time.



Making room for one new element in a multistack.

   (a) In the worst case, how long does it take to push one more element onto a multistack containing n elements?
   (b) ***Prove*** that the amortized cost of a push operation is $O(\log n)$, where n is the maximum number of elements in the multistack.

3. Describe and analyze an algorithm to determine, given an undirected graph $G = (V, E)$ and three vertices $u, v, w \in V$ as input, whether G contains a simple path from u to w that passes through v. You do ***not*** need to prove your algorithm is correct.

4. Suppose we are given an n-digit integer X. Repeatedly remove one digit from either end of X (your choice) until no digits are left. The *square-depth* of X is the maximum number of perfect squares that you can see during this process. For example, the number 32492 has square-depth 3, by the following sequence of removals:

$$32492 \to \underline{3249}\not{2} \to \underline{324}\not{9} \to \not{3}24 \to \not{2}\underline{4} \to \not{4}.$$

Describe and analyze an algorithm to compute the square-depth of a given integer X, represented as an array $X[1..n]$ of n decimal digits. Assume you have access to a subroutine IsSQUARE that determines whether a given k-digit number (represented by an array of digits) is a perfect square ***in $O(k^2)$ time***.

5. Suppose we are given two *sorted* arrays $A[1..n]$ and $B[1..n]$ containing $2n$ distinct numbers. Describe and analyze an algorithm that finds the nth smallest element in the union $A \cup B$ in $O(\log n)$ time.

6. Recall the following problem from Homework 2:

   - 3WAYPARTITION: Given a set X of positive integers, determine whether there are three disjoint subsets $A, B, C \subseteq X$ such that $A \cup B \cup C = X$ and

$$\sum_{a \in A} a = \sum_{b \in B} b = \sum_{c \in C} c.$$

   (a) ***Prove*** that 3WAYPARTITION is NP-hard. *[Hint: Don't try to reduce from 3SAT or 3COLOR; in this rare instance, the 3 is just a coincidence.]*

   (b) In Homework 2, you described an algorithm to solve 3WAYPARTITION in $O(nS^2)$ time, where S is the sum of all elements of X. Why doesn't this algorithm imply that P=NP?

7. Describe and analyze efficient algorithms to solve the following problems:

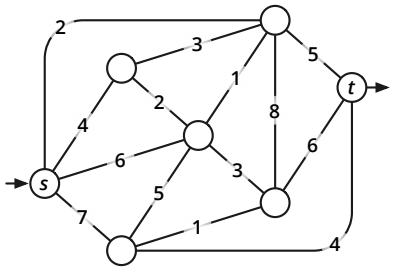
   (a) Given an array of n integers, does it contain two elements a, b such that $a + b = 0$?

   (b) Given an array of n integers, does it contain three elements a, b, c such that $a + b + c = 0$?

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

1. Clearly indicate the following structures 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 minimum (s, t)-cut



2. Describe a data structure that stores a set of numbers (which is initially empty) and supports the following operations in $O(1)$ amortized time:

   - INSERT(x): Insert x into the set. (You can safely assume that x is not already in the set.)
   - FINDMIN: Return the smallest element of the set (or NULL if the set is empty).
   - DELETEBOTTOMHALF: Remove the smallest $\lceil n/2 \rceil$ elements the set. (That's smallest by *value*, not smallest by *insertion time*.)

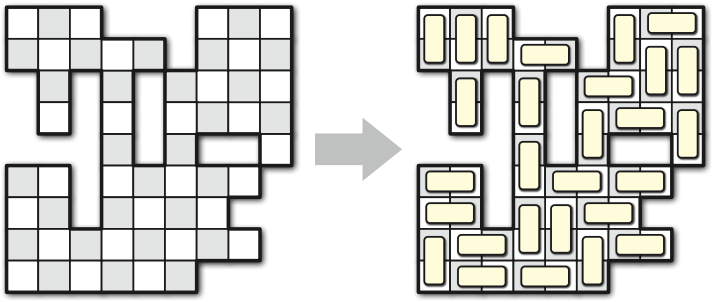
3. Suppose we are given two *sorted* arrays $A[1..n]$ and $B[1..n]$ containing $2n$ distinct numbers. Describe and analyze an algorithm that finds the nth smallest element in the union $A \cup B$ in $O(\log n)$ time.

4. Suppose you have a black-box subroutine QUALITY that can compute the 'quality' of any given string $A[1..k]$ in $O(k)$ time. For example, the quality of a string might be 1 if the string is a Québecois curse word, and 0 otherwise.

   Given an arbitrary input string $T[1..n]$, we would like to break it into contiguous substrings, such that the total quality of all the substrings is as large as possible. For example, the string `SAINTCIBOIREDESACRAMENTDECRISSE` can be decomposed into the substrings `SAINT + CIBOIRE + DE + SACRAMENT + DE + CRISSE`, of which three (or possibly four) are *sacres*.

   Describe an algorithm that breaks a string into substrings of maximum total quality, using the QUALITY subroutine.

5. Suppose you are given an $n \times n$ checkerboard with some of the squares removed. 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 place dominos on the board so that each domino covers exactly two squares (meaning squares that have *not* been removed) and each square is covered by exactly one domino. Your input is a two-dimensional array $Removed[1..n, 1..n]$ of booleans, where $Removed[i, j] = \text{TRUE}$ if and only if the square in row i and column j has been removed. For example, for the board shown below, your algorithm should return TRUE.



6. Recall the following problem from Homework 2:

   - 3WAYPARTITION: Given a set X of positive integers, determine whether there are three disjoint subsets $A, B, C \subseteq X$ such that $A \cup B \cup C = X$ and

$$\sum_{a \in A} a = \sum_{b \in B} b = \sum_{c \in C} c.$$

   (a) ***Prove*** that 3WAYPARTITION is NP-hard. *[Hint: Don't try to reduce from* 3SAT *or* 3COLOR*; in this rare instance, the 3 is just a coincidence.]*

   (b) In Homework 2, you described an algorithm to solve 3WAYPARTITION in $O(nS^2)$ time, where S is the sum of all elements of X. Why doesn't this algorithm imply that P=NP?

7. Describe and analyze efficient algorithms to solve the following problems:

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

   (b) Given an array of n integers, does it contain three elements a, b, c such that $a + b + c = 0$?