CS 473: Undergraduate Algorithms, Spring 2010
Homework 0

Due Tuesday, January 26, 2009 **in class**

- This homework tests 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 early chapters of any algorithms textbook should be sufficient review, but you may also want consult your favorite discrete mathematics and data structures textbooks. If you need help, please ask in office hours and/or on the course newsgroup.
- 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 linked from the course web site. If you have *any* questions, please ask during lecture or office hours, or post your question to the course newsgroup. In particular:
  - Submit five separately stapled solutions, one for each numbered problem, with your name and NetID clearly printed on each page. Please do *not* staple everything together.
  - You may use any source at your disposal—paper, electronic, or human—but you ***must*** write your solutions in your own words, and you ***must*** cite every source that you use.
  - Unless explicitly stated otherwise, ***every*** homework problem requires a proof.
  - Answering "I don't know" to any homework or exam problem (except for extra credit problems) 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 0 points.

---

1. (a) ***Write the sentence "I understand the course policies."***

   (b) **[5 pts]** Solve the following recurrences. State tight asymptotic bounds for each function in the form $\Theta(f(n))$ for some recognizable function $f(n)$. Assume reasonable but nontrivial base cases if none are given. ***Do not submit proofs***—just a list of five functions—but you should do them anyway, just for practice.

   - $A(n) = 3A(n-1) + 1$
   - $B(n) = B(n-5) + 2n - 3$
   - $C(n) = 4\,C(n/2) + \sqrt{n}$
   - $D(n) = 3\,D(n/3) + n^2$
   - $E(n) = E(n-1)^2 - E(n-2)^2$, where $E(0) = 0$ and $E(1) = 1$ *[Hint: This is easy!]*

   (c) **[5 pts]** Sort the following functions from asymptotically smallest to asymptotically largest, indicating ties if there are any. ***Do not submit proofs***—just a sorted list of 16 functions—but you should do them anyway, just for practice.

   Write $f(n) \ll g(n)$ to indicate that $f(n) = o(g(n))$, and write $f(n) \equiv g(n)$ to mean $f(n) = \Theta(g(n))$. We use the notation $\lg n = \log_2 n$.

n	$\lg n$	$\sqrt{n}$	5^n
$\sqrt{\lg n}$	$\lg \sqrt{n}$	$5^{\sqrt{n}}$	$\sqrt{5^n}$
$5^{\lg n}$	$\lg(5^n)$	$5^{\lg \sqrt{n}}$	$5\sqrt{\lg n}$
$\sqrt{5^{\lg n}}$	$\lg(5^{\sqrt{n}})$	$\lg \sqrt{5^n}$	$\sqrt{\lg(5^n)}$

2. **[CS 225 Spring 2009]** Suppose we build up a binary search tree by inserting elements one at a time from the set $\{1, 2, 3, \ldots, n\}$, starting with the empty tree. The structure of the resulting binary search tree depends on the order that these elements are inserted; every insertion order leads to a different n-node binary search tree.

   Recall that the *depth* of a leaf ℓ in a binary search tree is the number of *edges* between ℓ and the root, and the depth of a binary tree is the maximum depth of its leaves.

   (a) What is the maximum possible depth of an n-node binary search tree? Give an *exact* answer, and prove that it is correct.

   (b) *Exactly* how many different insertion orders result in an n-node binary search tree with maximum possible depth? Prove your answer is correct. *[Hint: Set up and solve a recurrence. Don't forget to prove that recurrence counts what you want it to count.]*

3. **[CS 173 Spring 2009]** A *binomial tree of order k* is defined recursively as follows:

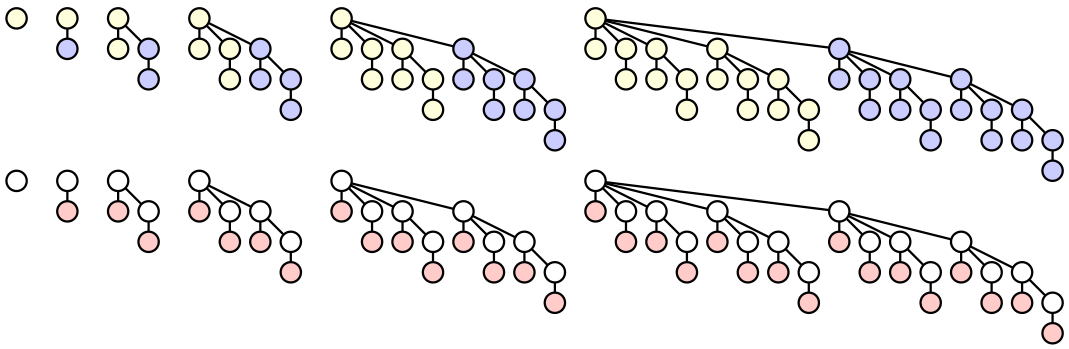
   - A binomial tree of order 0 is a single node.
   - For all $k > 0$, a binomial tree of order k consists of two binomial trees of order $k-1$, with the root of one tree connected as a new child of the root of the other. (See the figure below.)

   Prove the following claims:

   (a) For all non-negative integers k, a binomial tree of order k has exactly 2^k nodes.

   (b) For all positive integers k, attaching a leaf to every node in a binomial tree of order $k-1$ results in a binomial tree of order k.

   (c) For all non-negative integers k and d, a binomial tree of order k has exactly $\binom{k}{d}$ nodes with depth d.



Binomial trees of order 0 through 5.
Top row: the recursive definition. Bottom row: the property claimed in part (b).

4. **[CS 373 Fall 2009]** For any language $L \in \Sigma^*$, let

$$\mathit{Rotate}(L) := \left\{ w \in \Sigma^* \;\middle|\; w = xy \text{ and } yx \in L \text{ for some strings } x, y \in \Sigma^* \right\}$$

   For example, $\mathit{Rotate}(\{\texttt{OOK!}, \texttt{OOKOOK}\}) = \{\texttt{OOK!}, \texttt{OK!O}, \texttt{K!OO}, \texttt{!OOK}, \texttt{OOKOOK}, \texttt{OKOOKO}, \texttt{KOOKOO}\}$.

   Prove that if L is a regular language, then $\mathit{Rotate}(L)$ is also a regular language. *[Hint: Remember the power of nondeterminism.]*

5. Herr Professor Doktor Georg von den Dschungel has a 24-node binary tree, in which every node is labeled with a unique letter of the German alphabet, which is just like the English alphabet with four extra letters: **Ä**, **Ö**, **Ü**, and **ß**. (Don't confuse these with **A**, **O**, **U**, and **B!**) Preorder and postorder traversals of the tree visit the nodes in the following order:

   - Preorder: **B K Ü E H L Z I Ö R C ß T S O A Ä D F M N U G**
   - Postorder: **H I Ö Z R L E C Ü S O T A ß K D M U G N F Ä B**

   (a) List the nodes in George's tree in the order visited by an inorder traversal.

   (b) Draw George's tree.

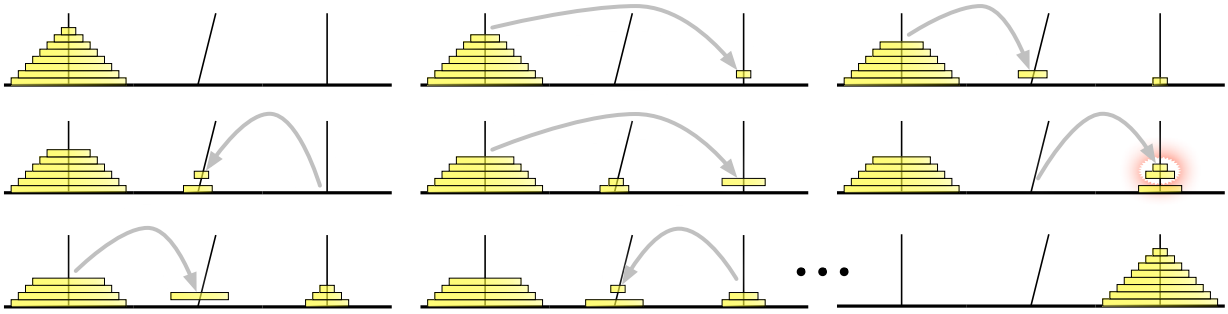
*6. ***[Extra credit]*** You may be familiar with the story behind the famous Tower of Hanoï puzzle, as related by Henri de Parville in 1884:

> In the great temple at Benares beneath the dome which marks the centre of the world, rests a brass plate in which are fixed three diamond needles, each a cubit high and as thick as the body of a bee. On one of these needles, at the creation, God placed sixty-four discs of pure gold, the largest disc resting on the brass plate, and the others getting smaller and smaller up to the top one. This is the Tower of Bramah. Day and night unceasingly the priests transfer the discs from one diamond needle to another according to the fixed and immutable laws of Bramah, which require that the priest on duty must not move more than one disc at a time and that he must place this disc on a needle so that there is no smaller disc below it. When the sixty-four discs shall have been thus transferred from the needle on which at the creation God placed them to one of the other needles, tower, temple, and Brahmins alike will crumble into dust, and with a thunderclap the world will vanish.

A less familiar chapter in the temple's history is its brief relocation to Pisa in the early 13th century. The relocation was organized by the wealthy merchant-mathematician Leonardo Fibonacci, at the request of the Holy Roman Emperor Frederick II, who had heard reports of the temple from soldiers returning from the Crusades. The Towers of Pisa and their attendant monks became famous, helping to establish Pisa as a dominant trading center on the Italian peninsula.

Unfortunately, almost as soon as the temple was moved, one of the diamond needles began to lean to one side. To avoid the possibility of the leaning tower falling over from too much use, Fibonacci convinced the priests to adopt a more relaxed rule: ***Any number of disks on the leaning needle can be moved together to another needle in a single move.*** It was still forbidden to place a larger disk on top of a smaller disk, and disks had to be moved one at a time *onto* the leaning needle or between the two vertical needles.

Thanks to Fibonacci's new rule, the priests could bring about the end of the universe somewhat faster from Pisa then they could than could from Benares. Fortunately, the temple was moved from Pisa back to Benares after the newly crowned Pope Gregory IX excommunicated Frederick II, making the local priests less sympathetic to hosting foreign heretics with strange mathematical habits. Soon afterward, a bell tower was erected on the spot where the temple once stood; it too began to lean almost immediately.



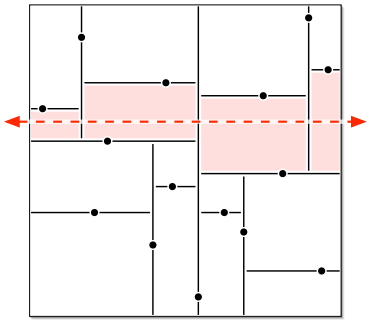
The Towers of Pisa. In the fifth move, two disks are taken off the leaning needle.

Describe an algorithm to transfer a stack of n disks from one *vertical* needle to the other *vertical* needle, using the smallest possible number of moves. *Exactly* how many moves does your algorithm perform?

- For this and all future homeworks, groups of up to three students can submit (or present) a single common solution. Please remember to write the names of all group members on every page.
- **Please fill out the online input survey linked from the course web page no later than Thursday, January 28.** Among other things, this survey asks you to identify the other members of your HW1 group, so that we can partition the class into presentation clusters without breaking up your group. We will announce the presentation clusters on Friday, January 29.
- Students in **Cluster 1** will present their solutions to Jeff or one of the TAs, on Tuesday or Wednesday of the due date (February 2 or February 3), instead of submitting written solutions. **Each homework group in Cluster 1 must sign up for a 30-minute time slot no later than Monday, February 1.** Signup sheets will be posted at 3303 Siebel Center ('The Theory Lab') later this week. Please see the course web page for more details.

---

1. Suppose we have n points scattered inside a two-dimensional box. A *kd-tree* recursively subdivides the points as follows. First we split the box into two smaller boxes with a *vertical* line, then we split each of those boxes with *horizontal* lines, and so on, always alternating between horizontal and vertical splits. Each time we split a box, the splitting line partitions the rest of the interior points *as evenly as possible* by passing through a median point in the interior of the box (*not* on its boundary). If a box doesn't contain any points, we don't split it any more; these final empty boxes are called *cells*.



A kd-tree for 15 points. The dashed line crosses the four shaded cells.

   (a) How many cells are there, as a function of n? Prove your answer is correct.

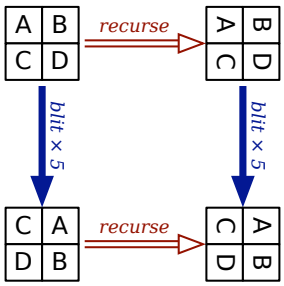
   (b) In the worst case, *exactly* how many cells can a horizontal line cross, as a function of n? Prove your answer is correct. Assume that $n = 2^k - 1$ for some integer k.

   (c) Suppose we have n points stored in a kd-tree. Describe and analyze an algorithm that counts the number of points above a horizontal line (such as the dashed line in the figure) as quickly as possible. *[Hint: Use part (b).]*

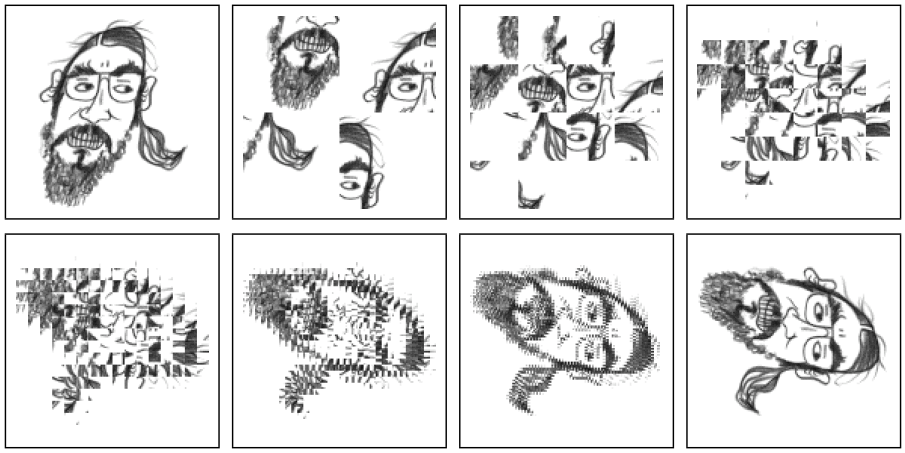
   (d) Describe an analyze an efficient algorithm that counts, given a kd-tree storing n points, the number of points that lie inside a rectangle R with horizontal and vertical sides. *[Hint: Use part (c).]*

2. Most graphics hardware includes support for a low-level operation called *blit*, or **bl**ock **t**ransfer, which quickly copies a rectangular chunk of a pixel map (a two-dimensional array of pixel values) from one location to another. This is a two-dimensional version of the standard C library function `memcpy()`.

   Suppose we want to rotate an $n \times n$ pixel map 90° clockwise. One way to do this, at least when n is a power of two, is to split the pixel map into four $n/2 \times n/2$ blocks, move each block to its proper position using a sequence of five blits, and then recursively rotate each block. Alternately, we could *first* recursively rotate the blocks and *then* blit them into place.



Two algorithms for rotating a pixel map.
Solid arrows indicate blitting the blocks into place; hollow arrows indicate recursively rotating the blocks.



The first rotation algorithm (blit then recurse) in action.

(a) Prove that both versions of the algorithm are correct when n is a power of two.

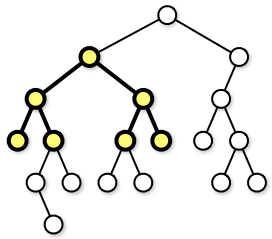
(b) *Exactly* how many blits does the algorithm perform when n is a power of two?

(c) Describe how to modify the algorithm so that it works for arbitrary n, not just powers of two. How many blits does your modified algorithm perform?

(d) What is your algorithm's running time if a $k \times k$ blit takes $O(k^2)$ time?

(e) What if a $k \times k$ blit takes only $O(k)$ time?

3. 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.

CS 473: Undergraduate Algorithms, Spring 2010
Homework 2

Written solutions due Tuesday, February 9, 2010 at noon

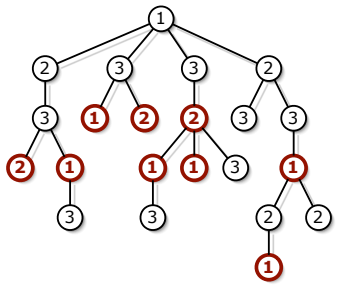
- Roughly 1/3 of the students will give oral presentations of their solutions to the TAs. ***You should have received an email telling you whether you are expected to present this homework.*** Please see the course web page for further details.
- Groups of up to three students may submit a common solution. Please clearly write every group member's name and NetID on every page of your submission. Please start your solution to each numbered problem on a new sheet of paper. Please ***don't*** staple solutions for different problems together.

---

1. A ***palindrome*** is a string that reads the same forwards and backwards, like `x`, `pop`, `noon`, `redivider`, 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 **M**A**H**D**Y**NA**M**ICP**RO**G**R**AMZLET**M**ESHOW**Y**OUT**H**E**M** is MHYMRORMYHM, so given that string as input, your algorithm should output the number 11.

2. Oh, no! You have been appointed as the gift czar for Giggle, Inc.'s annual mandatory holiday party! The president of the company, who is certifiably insane, has declared that every Giggle employee must receive one of three gifts: (1) an all-expenses-paid six-week vacation anywhere in the world, (2) an all-the-pancakes-you-can-eat breakfast for two at Jumping Jack Flash's Flapjack Stack Shack, or (3) a burning paper bag full of dog poop. Corporate regulations prohibit any employee from receiving the same gift as his/her direct supervisor. Any employee who receives a better gift than his/her direct supervisor will almost certainly be fired in a fit of jealousy. How do you decide what gifts everyone gets if you want to minimize the number of people that get fired?

   More formally, suppose you are given a rooted tree T, representing the company hierarchy. You want to label each node in T with an integer 1, 2, or 3, such that every node has a different label from its parent.. The *cost* of an labeling is the number of nodes that have smaller labels than their parents. Describe and analyze an algorithm to compute the minimum cost of any labeling of the given tree T. (Your algorithm does *not* have to compute the actual best labeling—just its cost.)



A tree labeling with cost 9. Bold nodes have smaller labels than their parents.
This is ***not*** the optimal labeling for this tree.

3. After graduating from UIUC, you have decided to join the Wall Street Bank ***Boole Long Live***. The managing director of the bank, Eloob Egroeg, is a genius mathematician who worships George Boole[1] every morning before leaving for the office. The first day of every hired employee is a 'solve-or-die' day where s/he has to solve one of the problems posed by Eloob within 24 hours. Those who fail to solve the problem are fired immediately!

   Entering into the bank for the first time, you notice that the offices of the employees are organized in a straight row, with a large "T" or "F" written on the door of each office. Furthermore, between each adjacent pair of offices, there is a board marked by one of the symbols $\wedge, \vee$, or $\oplus$. When you ask about these arcane symbols, Eloob confirms that T and F represent the boolean values 'true' and 'false', and the symbols on the boards represent the standard boolean operators AND, OR, and XOR. He also explains that these letters and symbols describe whether certain combinations of employees can work together successfully. At the start of any new project, Eloob hierarchically clusters his employees by adding parentheses to the sequence of symbols, to obtain an unambiguous boolean expression. The project is successful if this parenthesized boolean expression evaluates to T.

   For example, if the bank has three employees, and the sequence of symbols on and between their doors is $T \wedge F \oplus T$, there is exactly one successful parenthesization scheme: $(T \wedge (F \oplus T))$. However, if the list of door symbols is $F \wedge T \oplus F$, there is no way to add parentheses to make the project successful.

   Eloob finally poses your solve-or-die question: Describe and algorithm to decide whether a given sequence of symbols can be parenthesized so that the resulting boolean expression evaluates to T. The input to your algorithm is an array $S[0..2n]$, where $S[i] \in \{T, F\}$ when i is even, and $S[i] \in \{\vee, \wedge, \oplus\}$ when i is odd.

[1] 1815-1864, The inventor of Boolean Logic

- For this and all future homeworks, groups of up to three students can submit (or present) a single common solution. Please remember to write the names of all group members on every page.
- Students in **Cluster 3** will present their solutions to Jeff or one of the TAs, on Tuesday or Wednesday of the due date (February 16 or February 17), instead of submitting written solutions. **Each homework group in Cluster 3 must sign up for a 30-minute time slot no later than Monday, February 15.** Signup sheets will be posted at 3304 Siebel Center ('The Theory Lab') later this week. Please see the course web page for more details.

---

1. You saw in class a correct greedy algorithm for finding the maximum number of non-conflicting courses from a given set of possible courses. This algorithm repeatedly selects the class with the earliest completion time that does not conflict with any previously selected class.

   Below are four alternative greedy algorithms. For each algorithm, either prove that the algorithm constructs an optimal schedule, or give a concrete counterexample showing that the algorithm is suboptimal.

   (a) Choose the course that *ends latest*, discard all conflicting classes, and recurse.

   (b) Choose the course that *starts first*, discard all conflicting classes, and recurse.

   (c) Choose the course with *shortest duration*, discard all conflicting classes, and recurse.

   (d) Choose a course that *conflicts with the fewest other courses* (breaking ties arbitrarily), discard all conflicting classes, and recurse.

2. You have been given the task of designing an algorithm for vending machines that computes the smallest number of coins worth any given amount of money. Your supervisors at The Area 51 Soda Company are anticipating a hostile takeover of earth by an advanced alien race that uses an unknown system of currency. So your algorithm must be as general as possible so that it will work with the alien system, whatever it turns out to be.

   Given a quantity of money x, and a set of coin denominations $b_1, \ldots, b_k$, your algorithm should compute how to make change for x with the fewest number of coins. For example, if you use the US coin denominations (1¢, 5¢, 10¢, 25¢, 50¢, and 100¢), the optimal way to make 17¢ in change uses 4 coins: one dime (10¢), one nickel (5¢), and two pennies (1¢).

   (a) Show that the following greedy algorithm does *not* work for all currency systems: If $x = 0$, do nothing. Otherwise, find the largest denomination $c \leq x$, issue one c-cent coin, and recursively give $x - c$ cents in change.

   (b) Now suppose that the system of currency you are concerned with only has coins in powers of some base b. That is, the coin denominations are $b^0, b^1, b^2, \ldots, b^k$. Show that the greedy algorithm described in part (a) does make optimal change in this currency system.

   (c) Describe and analyze an algorithm that computes optimal change for *any* set of coin denominations. (You may assume the aliens' currency system includes a 1-cent coin, so that making change is always possible.)

3. Suppose you have just purchased a new type of hybrid car that uses fuel extremely efficiently, but can only travel 100 miles on a single battery. The car's fuel is stored in a single-use battery, which must be replaced after at most 100 miles. The actual fuel is virtually free, but the batteries are expensive and can only be installed by licensed battery-replacement technicians. Thus, even if you decide to replace your battery early, you must still pay full price for the new battery to be installed. Moreover, because these batteries are in high demand, no one can afford to own more than one battery at a time.

   Suppose you are trying to get from San Francisco to New York City on the new Inter-Continental Super-Highway, which runs in a direct line between these two cities. There are several fueling stations along the way; each station charges a different price for installing a new battery. Before you start your trip, you carefully print the Wikipedia page listing the locations and prices of every fueling station on the ICSH. Given this information, how do you decide the best places to stop for fuel?

   More formally, suppose you are given two arrays $D[1..n]$ and $C[1..n]$, where $D[i]$ is the distance from the start of the highway to the ith station, and $C[i]$ is the cost to replace your battery at the ith station. Assume that your trip starts and ends at fueling stations (so $D[1] = 0$ and $D[n]$ is the total length of your trip), and that your car starts with an empty battery (so you must install a new battery at station 1).

   (a) Describe and analyze a greedy algorithm to find the minimum number of refueling stops needed to complete your trip. Don't forget to prove that your algorithm is correct.

   (b) But what you really want to minimize is the total *cost* of travel. Show that your greedy algorithm in part (a) does *not* produce an optimal solution when extended to this setting.

   (c) Describe a dynamic programming algorithm to compute the locations of the fuel stations you should stop at to minimize the cost of travel.

1. Suppose we want to write an efficient function SHUFFLE(n) that returns a permutation of the set $\{1, 2, \ldots, n\}$ chosen uniformly at random.

   (a) Prove that the following algorithm is ***not*** correct. *[Hint: Consider the case $n = 3$.]*

```
SHUFFLE(n):
  for i ← 1 to n
      π[i] ← i
  for i ← 1 to n
      swap π[i] ↔ π[RANDOM(n)]
  return π[1..n]
```

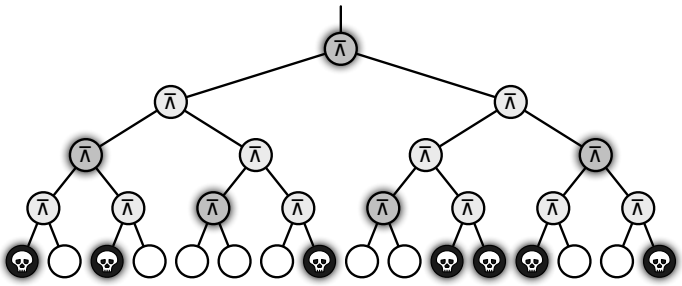
   (b) Consider the following implementation of SHUFFLE.

```
SHUFFLE(n):
  for i ← 1 to n
      π[i] ← NULL
  for i ← 1 to n
      j ← RANDOM(n)
      while (π[j] != NULL)
          j ← RANDOM(n)
      π[j] ← i
  return π[1..n]
```

   Prove that this algorithm is correct. What is its expected running time?

   (c) Describe and analyze an implementation of SHUFFLE that runs in $O(n)$ time. (An algorithm that runs in $O(n)$ *expected* time is fine, but $O(n)$ *worst-case* time is possible.)

2. Death knocks on your door one cold blustery morning and challenges you to a game. Death knows you are an algorithms student, so instead of the traditional game of chess, Death presents you with a complete binary tree with 4^n leaves, each colored either black or white. There is a token at the root of the tree. To play the game, you and Death will take turns moving the token from its current node to one of its children. The game will end after $2n$ moves, when the token lands on a leaf. If the final leaf is black, you die; if it's white, you will live forever. You move first, so Death gets the last turn.



   You can decide whether it's worth playing or not as follows. Imagine that the tree is a Boolean circuit whose inputs are specified at the leaves: white and black represent TRUE and FALSE inputs, respectively. Each internal node in the tree is a NAND gate that gets its input from its children and passes its output to its parent. (Recall that a NAND gate outputs FALSE if and only if both its inputs are TRUE.) If the output at the top of the tree is TRUE, then you can win and live forever! If the output at the top of the tree is FALSE, you should challenge Death to a game of Twister instead. Or maybe Battleship.

(a) Describe and analyze a deterministic algorithm to determine whether or not you can win. *[Hint: This is easy!]*

(b) Unfortunately, Death won't give you enough time to look at every node in the tree. Describe a *randomized* algorithm that determines whether you can win in $O(3^n)$ expected time. *[Hint: Consider the case $n = 1$.]*

*(c) **[Extra credit]** Describe and analyze a randomized algorithm that determines whether you can win in $O(c^n)$ expected time, for some constant $c < 3$. *[Hint: You may not need to change your algorithm from part (b) at all!]*

3. 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?]*

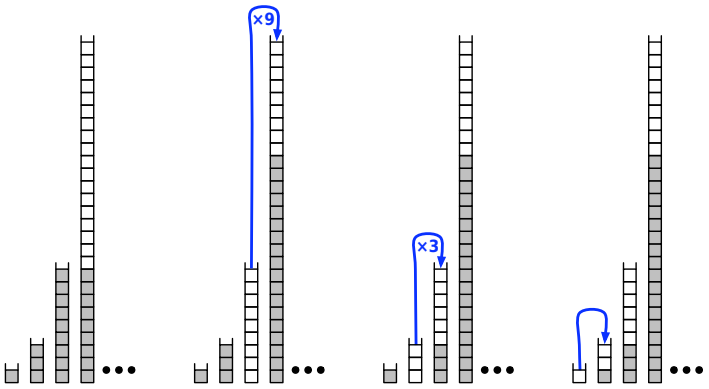
(b) Show that each of the other meldable priority queue operations can be implemented with at most one call to MELD and $O(1)$ additional time. (This implies that every operation takes $O(\log n)$ expected time.)

1. 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. The user always pushes and pops elements from the smallest stack S_0. However, before any element can be pushed 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}.) Similarly, before any element can be popped from any empty stack S_i, we first pop 3^i elements from S_{i+1} and push them onto S_i to make room. (Thus, if S_{i+1} is already empty, we first recursively fill it by popping elements from S_{i+2}.) Moving a single element from one stack to another takes $O(1)$ time.

   Here is pseudocode for the multistack operations MSPush and MSPop. The internal stacks are managed with the subroutines Push and Pop.

```
MPush(x):
  i ← 0
  while S_i is full
      i ← i + 1
  while i > 0
      i ← i − 1
      for j ← 1 to 3^i
          Push(S_{i+1}, Pop(S_i))
  Push(S_0, x)
```

```
MPop(x):
  i ← 0
  while S_i is empty
      i ← i + 1
  while i > 0
      i ← i − 1
      for j ← 1 to 3^i
          Push(S_i, Pop(S_{i+1}))
  return Pop(S_0)
```



Making room in a multistack, just before pushing on a new element.

   (a) In the worst case, how long does it take to push one more element onto a multistack containing n elements?

   (b) Prove that if the user never pops anything from the multistack, the amortized cost of a push operation is $O(\log n)$, where n is the maximum number of elements in the multistack during its lifetime.

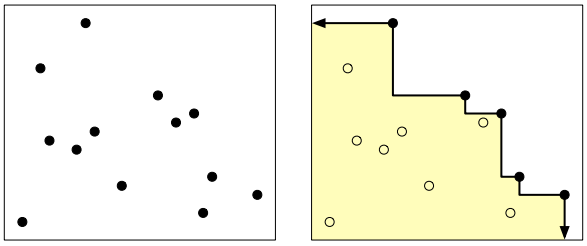
   (c) Prove that in any intermixed sequence of pushes and pops, each push or pop operation takes $O(\log n)$ amortized time, where n is the maximum number of elements in the multistack during its lifetime.

2. Design and analyze a simple data structure that maintains a list of integers and supports the following operations.

   - CREATE( ) creates and returns a new list
   - PUSH(L, x) appends x to the end of L
   - POP(L) deletes the last entry of L and returns it
   - LOOKUP(L, k) returns the kth entry of L

   Your solution may use these primitive data structures: arrays, balanced binary search trees, heaps, queues, single or doubly linked lists, and stacks. If your algorithm uses *anything* fancier, you must give an explicit implementation. Your data structure must support all operations in amortized constant time. In addition, your data structure must support each LOOKUP in *worst-case* $O(1)$ time. At all times, the size of your data structure must be linear in the number of objects it stores.

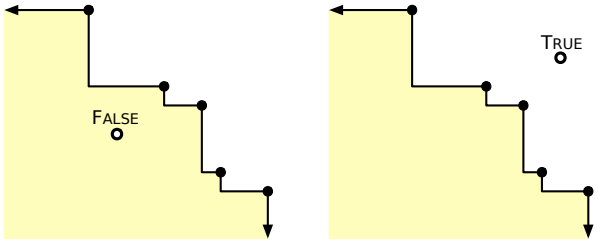
3. Let P be a set of n points in the plane. The *staircase* of P is the set of all points in the plane that have at least one point in P both above and to the right.



A set of points in the plane and its staircase (shaded).

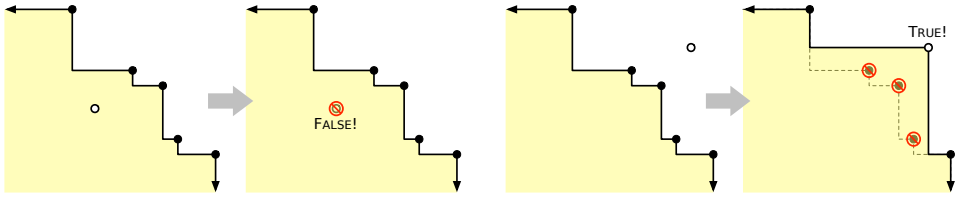
(a) Describe an algorithm to compute the staircase of a set of n points in $O(n \log n)$ time.

(b) 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.



Two staircase queries.

(c) Describe and analyze a data structure that maintains a staircase 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.



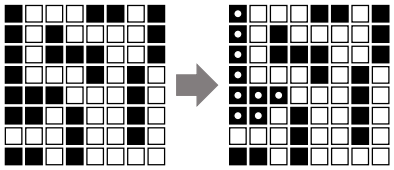
Two staircase insertions.

CS 473: Undergraduate Algorithms, Spring 2010
Homework 6

Written solutions due Tuesday, March 16, 2010 at noon

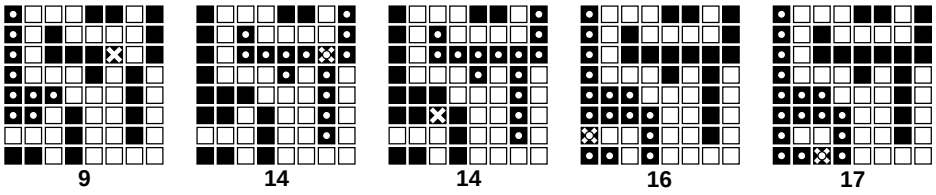
---

1. (a) Describe and analyze an algorithm to compute the size of the largest connected component of black pixels in an $n \times n$ 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?

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 *increased*.

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

   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.]*

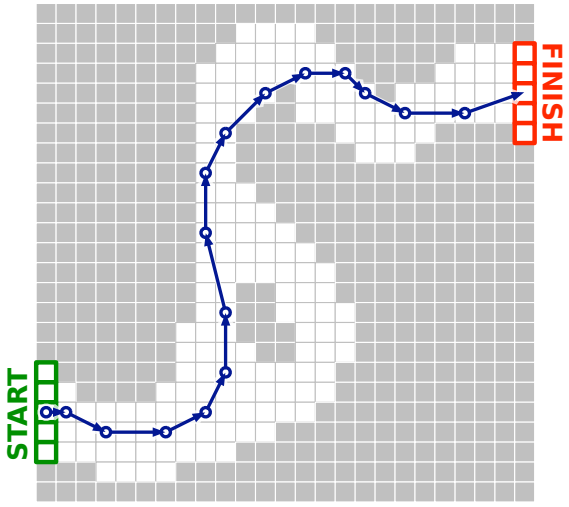
3. ***Racetrack*** (also known as *Graph Racers* and *Vector Rally*) is a two-player paper-and-pencil racing game of uncertain origin that Jeff played on the bus in 5th grade.[1] The game is played using a racetrack drawn on a sheet of graph paper. The players alternately choose a sequence of grid points 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. The initial position is an arbitrary point on the starting line, chosen by the player; the initial velocity 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 lie inside the track; otherwise, the car crashes and that player immediately loses the race. The first car that reaches a position ***on*** the finish line is the winner.

   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 line' is the first column, and the 'finish line' is the last column.

   Describe and analyze an algorithm to find *the minimum number of steps* required to move a car from the starting line to the finish line according to these rules, given a racetrack bitmap as input. *[Hint: Build a graph. What are the vertices? What are the edges? What problem is this?]*

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 Jeff played was a bit more complicated than the version described in this problem. In particular, the track was a freeform curve, and by default, the entire line segment traversed by a car in a single step had to lie entirely inside the track. If a car did run off the track, it started its next turn with velocity zero, at the legal grid point closest to where it first crossed the track boundary.

1. On an overnight camping trip in Sunnydale National Park, you are woken from a restless sleep by a scream. As you crawl out of your tent to investigate, a terrified park ranger runs out of the woods, covered in blood and clutching a crumpled piece of paper to his chest. As he reaches your tent, he gasps, "Get out... while... you...", thrusts the paper into your hands, and falls to the ground. Checking his pulse, you discover that the ranger is stone dead.

   You look down at the paper and recognize a map of the park, drawn as an undirected graph, where vertices represent landmarks in the park, and edges represent trails between those landmarks. (Trails start and end at landmarks and do not cross.) You recognize one of the vertices as your current location; several vertices on the boundary of the map are labeled EXIT.

   On closer examination, you notice that someone (perhaps the poor dead park ranger) has written a real number between 0 and 1 next to each vertex and each edge. A scrawled note on the back of the map indicates that a number next to an edge is the probability of encountering a vampire along the corresponding trail, and a number next to a vertex is the probability of encountering a vampire at the corresponding landmark. (Vampires can't stand each other's company, so you'll never see more than one vampire on the same trail or at the same landmark.) The note warns you that stepping off the marked trails will result in a slow and painful death.

   You glance down at the corpse at your feet. Yes, his death certainly looked painful. Wait, was that a twitch? Are his teeth getting longer? After driving a tent stake through the undead ranger's heart, you wisely decide to leave the park immediately.

   Describe and analyze an efficient algorithm to find a path from your current location to an arbitrary EXIT node, such that the total *expected number* of vampires encountered along the path is as small as possible. *Be sure to account for* ***both*** *the vertex probabilities* ***and*** *the edge probabilities!*

2. In this problem we will discover how you, too, can be employed by Wall Street and cause a major economic collapse! The *arbitrage* business is a money-making scheme that takes advantage of differences in currency exchange. In particular, suppose that 1 US dollar buys 120 Japanese yen; 1 yen buys 0.01 euros; and 1 euro buys 1.2 US dollars. Then, a trader starting with $1 can convert his money from dollars to yen, then from yen to euros, and finally from euros back to dollars, ending with $1.44! The cycle of currencies $\$ \to ¥ \to € \to \$$ is called an ***arbitrage cycle***. Of course, finding and exploiting arbitrage cycles before the prices are corrected requires extremely fast algorithms.

   Suppose n different currencies are traded in your currency market. You are given the matrix $R[1..n, 1..n]$ of exchange rates between every pair of currencies; for each i and j, one unit of currency i can be traded for $R[i, j]$ units of currency j. (Do *not* assume that $R[i, j] \cdot R[j, i] = 1$.)

   (a) Describe an algorithm that returns an array $V[1..n]$, where $V[i]$ is the maximum amount of currency i that you can obtain by trading, starting with one unit of currency 1, assuming there are no arbitrage cycles.

   (b) Describe an algorithm to determine whether the given matrix of currency exchange rates creates an arbitrage cycle.

   (c) Modify your algorithm from part (b) to actually return an arbitrage cycle, if it exists.

3. Let $G = (V, E)$ be a directed graph with weighted edges; edge weights could be positive, negative, or zero. In this problem, you will develop an algorithm to compute shortest paths between *every* pair of vertices. The output from this algorithm is a two-dimensional array $dist[1..V, 1..V]$, where $dist[i, j]$ is the length of the shortest path from vertex i to vertex j.

   (a) How could we delete some node v from this graph, without changing the shortest-path distance between any other pair of nodes? Describe an algorithm that constructs a directed graph $G' = (V', E')$ with weighted edges, where $V' = V \setminus \{v\}$, and the shortest-path distance between any two nodes in G' is equal to the shortest-path distance between the same two nodes in G. For full credit, your algorithm should run in $O(V^2)$ time.

   (b) Now suppose we have already computed all shortest-path distances in G'. Describe an algorithm to compute the shortest-path distances from v to every other node, and from every other node to v, in the original graph G. For full credit, your algorithm should run in $O(V^2)$ time.

   (c) Combine parts (a) and (b) into an algorithm that finds the shortest paths between *every* pair of vertices in the graph. For full credit, your algorithm should run in $O(V^3)$ time.

   The lecture notes (along with most algorithms textbooks and Wikipedia) describe a dynamic programming algorithm due to Floyd and Warshall that computes all shortest paths in $O(V^3)$ time. This is *not* that algorithm.

CS 473: Undergraduate Algorithms, Spring 2010
Homework 8

Written solutions due Tuesday, April 20, 2010 in class.

---

1. Suppose you have already computed a maximum (s,t)-flow f in a flow network G with integer capacities. Let k be an arbitrary positive integer, and let e be an arbitrary edge in G whose capacity is at least k.

   (a) Suppose we *increase* the capacity of e by k units. Describe and analyze an algorithm to update the maximum flow.

   (b) Now suppose we *decrease* the capacity of e by k units. Describe and analyze an algorithm to update the maximum flow.

   For full credit, both algorithms should run in $O(Ek)$ time. *[Hint: First consider the case $k = 1$.]*

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

3. A *cycle cover* of a given directed graph $G = (V, E)$ is a set of vertex-disjoint cycles that cover all the vertices. Describe and analyze an efficient algorithm to find a cycle cover for a given graph, or correctly report that none exists. *[Hint: Use ipartite atching!]*

1. We say that an array $A[1..n]$ is *k-sorted* if it can be divided into k blocks, each of size n/k, such that the elements in each block are larger than the elements in earlier blocks, and smaller than elements in later blocks. The elements within each block need not be sorted.

   For example, the following array is 4-sorted:

1	2	4	3	7	6	8	5	10	11	9	12	15	13	16	14

   (a) Describe an algorithm that k-sorts an arbitrary array in time $O(n \log k)$.

   (b) Prove that any comparison-based k-sorting algorithm requires $\Omega(n \log k)$ comparisons in the worst case.

   (c) Describe an algorithm that completely sorts an already k-sorted array in time $O(n \log(n/k))$.

   (d) Prove that any comparison-based algorithm to completely sort a k-sorted array requires $\Omega(n \log(n/k))$ comparisons in the worst case.

   In all cases, you can assume that n/k is an integer and that $n! \approx \left(\frac{n}{e}\right)^n$.

2. Recall the nuts and bolts problem from the first randomized algorithms lecture. You are given n nuts and n bolts of different sizes. Each nut matches exactly one bolt and vice versa. The nuts and bolts are all almost exactly the same size, so we can't tell if one bolt is bigger than the other, or if one nut is bigger than the other. If we try to match a nut with a bolt, however, we will discover either that the nut is too big, the nut is too small, or the nut is just right for the bolt. The goal was to find the matching nut for every bolt.

   Now consider a relaxed version of the problem where the goal is to find the matching nuts for *half* of the bolts, or equivalently, to find $n/2$ matched nut-bolt pairs. (It doesn't matter *which* $n/2$ nuts and bolts are matched.) Prove that any deterministic algorithm to solve this problem must perform $\Omega(n \log n)$ nut-bolt tests in the worst case.

3. UIUC has just finished constructing the new Reingold Building, the tallest dormitory on campus. In order to determine how much insurance to buy, the university administration needs to determine the highest safe floor in the building. A floor is consdered *safe* if ~~a drunk student~~ ***an egg*** can fall from a window on that floor and land without breaking; if the egg breaks, the floor is considered *unsafe*. Any floor that is higher than an unsafe floor is also considered unsafe. The only way to determine whether a floor is safe is to drop an egg from a window on that floor.

   You would like to find the lowest unsafe floor L by performing as few tests as possible; unfortunately, you have only a very limited supply of eggs.

   (a) Prove that if you have only one egg, you can find the lowest unsafe floor with L tests. *[Hint: Yes, this is trivial.]*

   (b) Prove that if you have only one egg, you must perform at least L tests in the worst case. In other words, prove that your algorithm from part (a) is optimal. *[Hint: Use an adversary argument.]*

   (c) Describe an algorithm to find the lowest unsafe floor using *two* eggs and only $O(\sqrt{L})$ tests. *[Hint: Ideally, each egg should be dropped the same number of times. How many floors can you test with n drops?]*

   (d) Prove that if you start with two eggs, you must perform at least $\Omega(\sqrt{L})$ tests in the worst case. In other words, prove that your algorithm from part (c) is optimal.

This homework is practice only. However, there will be at least one NP-hardness problem on the final exam, so working through this homework is *strongly* recommended. Students/groups are welcome to submit solutions for feedback (but not credit) in class on May 4, after which we will publish official solutions.

---

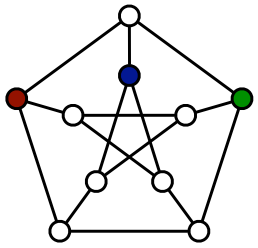
1. Recall that 3SAT asks whether a given boolean formula in conjunctive normal form, with exactly three literals in each clause, is satisfiable. In class we proved that 3SAT is NP-complete, using a reduction from CIRCUITSAT.

   Now consider the related problem **2SAT**: Given a boolean formula in conjunctive normal form, with exactly *two* literals in each clause, is the formula satisfiable? For example, the following boolean formula is a valid input to 2SAT:

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

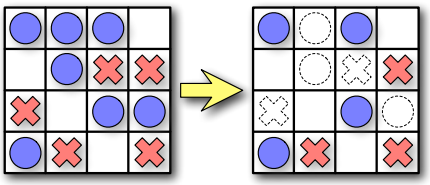
   Either prove that 2SAT is NP-hard or describe a polynomial-time algorithm to solve it. *[Hint: Recall that $(x \vee y) \equiv (\overline{x} \to y)$, and build a graph.]*

2. Let $G = (V, E)$ be a graph. A *dominating set* in G is a subset S of the vertices such that every vertex in G is either in S or adjacent to a vertex in S. The DOMINATINGSET problem asks, given a graph G and an integer k as input, whether G contains a dominating set of size k. Either prove that this problem is NP-hard or describe a polynomial-time algorithm to solve it.

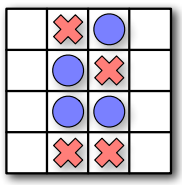


A dominating set of size 3 in the Peterson graph.

3. Consider the following solitaire game. The puzzle consists of an $n \times m$ grid of squares, where each square may be empty, occupied by a red stone, or occupied by a blue stone. The goal of the puzzle is to remove some of the given stones so that the remaining stones satisfy two conditions: (1) every row contains at least one stone, and (2) no column contains stones of both colors. For some initial configurations of stones, reaching this goal is impossible.



A solvable puzzle and one of its many solutions.



An unsolvable puzzle.

Prove that it is NP-hard to determine, given an initial configuration of red and blue stones, whether the puzzle can be solved.

This exam lasts 120 minutes.
Write your answers in the separate answer booklet.
Please return this question 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: $\Theta(n^2)$

   Choose the correct answer for each question. Each correct answer is worth $+1$ point; each incorrect answer is worth $-1/2$ point; and each "I don't know" is worth $+1/4$ point. Negative scores will be recorded as 0.

   (a) What is $\dfrac{3}{n} + \dfrac{n}{3}$?

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

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

   (d) How many bits are required to write the number $n!$ (the factorial of n) in binary?

   (e) What is the solution to the recurrence $E(n) = E(n-3) + 17n$?

   (f) What is the solution to the recurrence $F(n) = 2F(n/4) + 6n$?

   (g) What is the solution to the recurrence $G(n) = 9G(n/9) + 9n$?

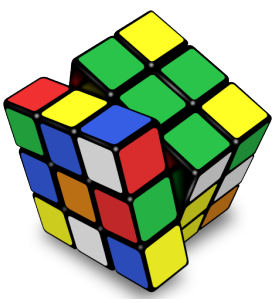
   (h) What is the worst-case running time of quicksort?

   (i) Let $X[1..n, 1..n]$ be a fixed array of numbers. Consider the following recursive function:

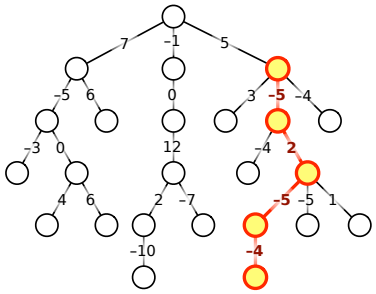
$$WTF(i,j) = \begin{cases} 0 & \text{if } \min\{i,j\} \le 0 \\ -\infty & \text{if } \max\{i,j\} > n \\ X[i,j] + \max\left\{ \begin{array}{l} WTF(i-2, j+1) \\ WTF(i-2, j-1) \\ WTF(i-1, j-2) \\ WTF(i+1, j-2) \end{array} \right\} & \text{otherwise} \end{cases}$$

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

   (j) The Rubik's Cube is a mechanical puzzle invented in 1974 by Ernő Rubik, a Hungarian professor of architecture. The puzzle consists of a $3 \times 3 \times 3$ grid of 'cubelets', whose faces are covered with stickers in six different colors. In the puzzle's solved state, each face of the puzzle is one solid color. A mechanism inside the puzzle allows any face of the cube to be freely turned (as shown on the right). The puzzle can be scrambled by repeated turns. Given a scrambled Rubik's Cube, how long does it take to find the *shortest* sequence of turns that returns the cube to its solved state?



2. 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.



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

3. 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 $a + b = c$?

4. A *common supersequence* of two strings A and B is another string that includes both the characters of A in order and the characters of B in order. Describe and analyze and algorithm to compute the length of the *shortest* common supersequence of two strings $A[1..m]$ and $B[1..n]$. You do not need to compute an actual supersequence, just its length.

   For example, if the input strings are `ANTHROHOPOBIOLOGICAL` and `PRETERDIPLOMATICALLY`, your algorithm should output 31, because a shortest common supersequence of those two strings is `PREANTHEROHODPOBIOPLOMATGICALLY`.

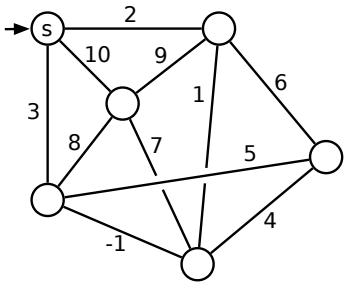
5. *[Taken directly from HBS0.]* Recall that the *Fibonacci numbers* F_n are recursively defined as follows: $F_0 = 0$, $F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$ for every integer $n \geq 2$. The first few Fibonacci numbers are $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$.

   Prove that any non-negative integer can be written as the sum of distinct *non-consecutive* Fibonacci numbers. That is, if any Fibonacci number F_n appears in the sum, then its neighbors F_{n-1} and F_{n+1} do not. For example:

$$\begin{aligned} 88 = \quad & 55 + 21 + 8 + 3 + 1 \quad && = F_{10} + F_8 + F_6 + F_4 + F_2 \\ 42 = \quad & 34 + 8 \quad && = F_9 + F_6 \\ 17 = \quad & 13 + 3 + 1 \quad && = F_7 + F_4 + F_2 \end{aligned}$$

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

1. Find the following spanning trees for the weighted graph shown below.



   (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.~~ ***Oops!***
   (d) A minimum spanning tree.

   You do *not* need to justify your answers; just clearly indicate the edges of each spanning tree in your answer booklet. Yes, one of the edges has negative weight.

2. *[Taken directly from HBS 6.]* 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. ***Prove*** that a connected undirected graph G has an Euler tour if and only if every vertex in G has even degree.

3. You saw in class that the standard algorithm to INCREMENT a binary counter runs in $O(1)$ amortized time. Now suppose we also want to support a second function called RESET, which resets all bits in the counter to zero.

   Here are the INCREMENT and RESET algorithms. In addition to the array $B[\ldots]$ of bits, we now also maintain the index of the most significant bit, in an integer variable msb.

```
INCREMENT(B[0..∞], msb):
  i ← 0
  while B[i] = 1
      B[i] ← 0
      i ← i + 1
  B[i] ← 1
  if i > msb
      msb ← i
```

```
RESET(B[0..∞], msb):
  for i ← 0 to msb
      B[i] ← 0
  msb ← 0
```

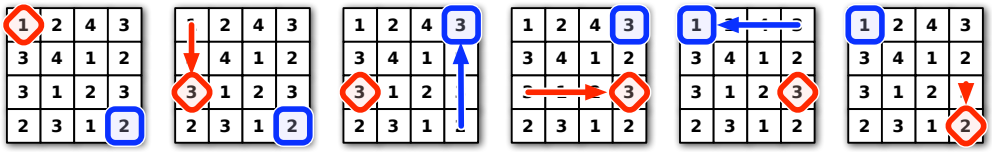
   In parts (a) and (b), let n denote the number currently stored in the counter.

   (a) What is the worst-case running time of INCREMENT, as a function of n?
   (b) What is the worst-case running time of RESET, as a function of n?
   (c) ***Prove*** that in an *arbitrary* intermixed sequence of INCREMENT and RESET operations, the amortized time for each operation is $O(1)$.

4. The following puzzle was invented by the infamous Mongolian puzzle-warrior Vidrach Itky Leda in the year 1473. The puzzle consists of an $n \times n$ grid of squares, where each square is labeled with a positive integer, and two tokens, one red and the other blue. The tokens always lie on distinct squares of the grid. The tokens start in the top left and bottom right corners of the grid; the goal of the puzzle is to swap the tokens.

   In a single turn, you may move either token up, right, down, or left *by a distance determined by the **other** token*. For example, if the red token is on a square labeled 3, then you may move the blue token 3 steps up, 3 steps left, 3 steps right, or 3 steps down. However, you may not move a token off the grid or to the same square as the other token.

   Describe and analyze an efficient algorithm that either returns the minimum number of moves required to solve a given Vidrach Itky Leda puzzle, or correctly reports that the puzzle has no solution. For example, given the puzzle below, your algorithm would return the number 5.



A five-move solution for a 4×4 Vidrach Itky Leda puzzle.

5. Suppose you are given an array $X[1..n]$ of real numbers chosen independently and uniformly at random from the interval $[0,1]$. An array entry $X[i]$ is called a *local maximum* if it is larger than its neighbors $X[i-1]$ and $X[i+1]$ (if they exist).

   What is the *exact* expected number of local maxima in X? ***Prove*** that your answer is correct. *[Hint: Consider the special case $n = 3$.]*

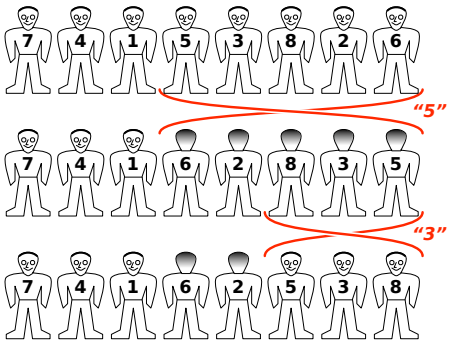
0.7	0.3	**1.0**	0.1	0.0	0.5	**0.6**	0.2	0.4	**0.9**	0.8

A randomly filled array with 4 local maxima.

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

1. Choreographer Michael Flatley has hired a new dance company to perform his latest Irish step-dancing extravaganza. At their first practice session, the new dancers line up in a row on stage and practice a movement called the *Flatley Flip*: Whenever Mr. Flatley calls out any positive integer k, the k rightmost dancers rotate 180 degrees as a group, so that their order in the line is reversed.

   Each dancer wears a shirt with a positive integer printed on the front and back; different dancers have different numbers. Mr. Flatley wants to rearrange the dancers, using only a sequence of Flatley Flips, so that these numbers are sorted from left to right in increasing order.



Two Flatley flips.

   (a) Describe an algorithm to sort an arbitrary row of n numbered dancers, using $O(n)$ Flatley flips. (After sorting, the dancers may face forward, backward, or some of each.) *Exactly* how many flips does your algorithm perform in the worst case?[1]

   (b) Describe an algorithm that sorts an arbitrary row of n numbered dancers *and ensures that all dancers are facing forward*, using $O(n)$ Flatley flips. *Exactly* how many flips does your algorithm perform in the worst case?[2]

2. You're in charge of choreographing a musical for your local community theater, and it's time to figure out the final pose of the big show-stopping number at the end. ("Streetcar!") You've decided that each of the n cast members in the show will be positioned in a big line when the song finishes, all with their arms extended and showing off their best spirit fingers.

   The director has declared that during the final flourish, each cast member must either point both their arms up or point both their arms down; it's your job to figure out who points up and who points down. Moreover, in a fit of unchecked power, the director has also given you a list of arrangements that will upset his delicate artistic temperament. Each forbidden arrangement is a subset of cast members paired with arm positions; for example: "Marge may not point her arms up while Ned and Apu point their arms down."

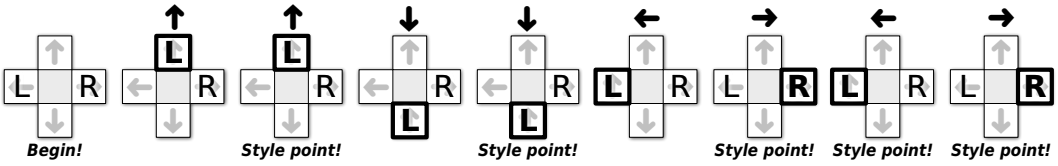
   Prove that finding an acceptable arrangement of arm positions is NP-hard. *[Hint: Describe a reduction from 3SAT.]*

[1] This is really a question about networking.
[2] This is really a question about mutating DNA.

3. ***Dance Dance Revolution*** is a dance video game, first introduced in Japan by Konami in 1998. Players stand on a platform marked with four arrows, pointing forward, back, left, and right, arranged in a cross pattern. During play, the game plays a song and scrolls a sequence of n arrows (←, ↑, ↓, or →) from the bottom to the top of the screen. At the precise moment each arrow reaches the top of the screen, the player must step on the corresponding arrow on the dance platform. (The arrows are timed so that you'll step with the beat of the song.)

   You are playing a variant of this game called "Vogue Vogue Revolution", where the goal is to play perfectly but move as little as possible. When an arrow reaches the top of the screen, if one of your feet is already on the correct arrow, you are awarded one style point for maintaining your current pose. If neither foot is on the right arrow, you must move one (and *only* one) of your feet from its current location to the correct arrow on the platform. If you ever step on the wrong arrow, or fail to step on the correct arrow, or move more than one foot at a time, all your style points are taken away and the games ends.

   How should you move your feet to maximize your total number of style points? For purposes of this problem, assume you always start with you left foot on ← and you right foot on →, and that you've memorized the entire sequence of arrows. For example, if the sequence is ↑↑↓↓←→←→, you can earn 5 style points by moving you feet as shown below:



   (a) ***Prove*** that for *any* sequence of n arrows, it is possible to earn at least $n/4 - 1$ style points.

   (b) Describe an efficient algorithm to find the maximum number of style points you can earn during a given VVR routine.[3] The input to your algorithm is an array $Arrow[1..n]$ containing the sequence of arrows. *[Hint: Build a graph!]*

4. It's almost time to show off your flippin' sweet dancing skills! Tomorrow is the big dance contest you've been training for your entire life, except for that summer you spent with your uncle in Alaska hunting wolverines. You've obtained an advance copy of the the list of n songs that the judges will play during the contest, in chronological order.

   You know all the songs, all the judges, and your own dancing ability extremely well. For each integer k, you know that if you dance to the kth song on the schedule, you will be awarded exactly $Score[k]$ points, but then you will be physically unable to dance for the next $Wait[k]$ songs (that is, you cannot dance to songs $k+1$ through $k+Wait[k]$). The dancer with the highest total score at the end of the night wins the contest, so you want your total score to be as high as possible.

   Describe and analyze an efficient algorithm to compute the maximum total score you can achieve. The input to your sweet algorithm is the pair of arrays $Score[1..n]$ and $Wait[1..n]$.[4]

[3] This is really a question about paging.
[4] This is really a question about processor scheduling.

5. 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.

6. You've been put in charge of putting together a team for the "Dancing with the Computer Scientists" international competition. Good teams in this competition must be capable of performing a wide variety of dance styles. You are auditioning a set of n dancing computer scientists, each of whom specializes in a particular style of dance.

   Describe an algorithm to determine in $O(n)$ time if *more than half* of the n dancers specialize exactly in the same dance style. The input to your algorithm is an array of n positive integers, where each integer identifies a style: 1 = ballroom, 2 = latin, 3 = swing, 4 = b-boy, 42 = contact improv, 101 = peanut butter jelly time, and so on. *[Hint: Remember the* SELECT *algorithm!]*

7. The party you are attending is going great, but now it's time to line up for ***The Algorithm March (アルゴリズムこうしん)***! This dance was originally developed by the Japanese comedy duo Itsumo Kokokara (いつもここから) for the children's television show PythagoraSwitch (ピタゴラスイッチ). The Algorithm March is performed by a line of people; each person in line starts a specific sequence of movements one measure later than the person directly in front of them. Thus, the march is the dance equivalent of a musical round or canon, like "Row Row Row Your Boat".

   Proper etiquette dictates that each marcher must know the person directly in front of them in line, lest a minor mistake during lead to horrible embarrassment between strangers. Suppose you are given a complete list of which people at your party know each other. ***Prove*** that it is NP-hard to determine the largest number of party-goers that can participate in the Algorithm March.[5]

[5] This is really a question about ninjas.

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?

MinDominatingSet: Given an undirected graph G, what is the size of the smallest subset S of vertices such that every vertex in G is either in S or adjacent to a vertex in S?

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?

ChromaticNumber: Given an undirected graph G, what is the minimum number of colors needed to color its vertices, 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?

Minesweeper: Given a Minesweeper configuration and a particular square x, is it safe to click on x?

Tetris: Given a sequence of N Tetris pieces and a partially filled $n \times k$ board, is it possible to play every piece in the sequence without overflowing the board?

Sudoku: Given an $n \times n$ Sudoku puzzle, does it have a solution?

KenKen: Given an $n \times n$ Ken-Ken puzzle, does it have a solution?