CS 473: Undergraduate Algorithms, Spring 2009
Homework 0

Due **in class** at 11:00am, Tuesday, January 27, 2009

- 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 this homework. For all future homeworks, groups of up to three students may submit a single, common solution.
- 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.

  Write the sentence "I understand the course policies." at the top of your solution to problem 1.

---

1. Professor George O'Jungle has a 27-node binary tree, in which every node is labeled with a unique letter of the Roman alphabet or the character **&**. Preorder and postorder traversals of the tree visit the nodes in the following order:

   - Preorder: **I Q J H L E M V O T S B R G Y Z K C A & F P N U D W X**
   - Postorder: **H E M L J V Q S G Y R Z B T C P U D N F W & X A K O I**

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

   (b) Draw George's tree.

2. (a) **[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 your solution requires a particular base case, say so. ***Do not submit proofs***—just a list of five functions—but you should do them anyway, just for practice.

$$A(n) = 10A(n/5) + n$$

$$B(n) = 2B\left(\left\lceil \frac{n+3}{4} \right\rceil\right) + 5n^{6/7} - 8\sqrt{\frac{n}{\log n}} + 9\left\lfloor \log^{10} n \right\rfloor - 11$$

$$C(n) = 3C(n/2) + C(n/3) + 5C(n/6) + n^2$$

$$D(n) = \max_{0<k<n} \left(D(k) + D(n-k) + n\right)$$

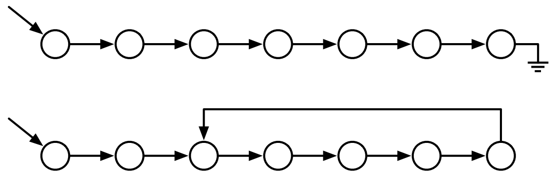
$$E(n) = \frac{E(n-1)\,E(n-3)}{E(n-2)} \qquad \textit{[Hint: Write out the first 20 terms.]}$$

(b) **[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	$\lg n$	$\sqrt{n}$	3^n
$\sqrt{\lg n}$	$\lg \sqrt{n}$	$3^{\sqrt{n}}$	$\sqrt{3^n}$
$3^{\lg n}$	$\lg(3^n)$	$3^{\lg \sqrt{n}}$	$3^{\sqrt{\lg n}}$
$\sqrt{3^{\lg n}}$	$\lg(3^{\sqrt{n}})$	$\lg \sqrt{3^n}$	$\sqrt{\lg(3^n)}$

3. Suppose you are given a pointer to the head of singly linked list. Normally, each node in the list has a pointer to the next element, and the last node's pointer is NULL. Unfortunately, your list might have been corrupted (by a bug in *somebody else's* code, of course), so that some node's pointer leads back to an earlier node in the list.


Top: A standard singly-linked list. Bottom: A corrupted singly linked list.

Describe an algorithm[1] that determines whether the linked list is corrupted or not. Your algorithm must not modify the list. For full credit, your algorithm should run in $O(n)$ time, where n is the number of nodes in the list, and use $O(1)$ extra space (not counting the list itself).

[1]Since you understand the course policies, you know what this phrase means. Right?

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

$$\begin{aligned}42 &= 3^4 - 3^3 - 3^2 - 3^1\\ 25 &= 3^3 - 3^1 + 3^0\\ 17 &= 3^3 - 3^2 - 3^0\end{aligned}$$

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

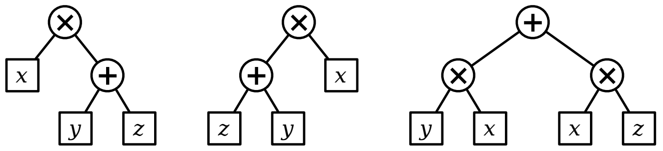
$$\begin{aligned}42 &= (-2)^6 + (-2)^5 + (-2)^4 + (-2)^0\\ 25 &= (-2)^6 + (-2)^5 + (-2)^3 + (-2)^0\\ 17 &= (-2)^4 + (-2)^0\end{aligned}$$

*[Hint: Don't use weak induction. In fact, **never** use weak induction.]*

5. An ***arithmetic expression tree*** is a binary tree where every leaf is labeled with a variable, every internal node is labeled with an arithmetic operation, and every internal node has exactly two children. For this problem, assume that the only allowed operations are $+$ and $\times$. Different leaves may or may not represent different variables.

Every arithmetic expression tree represents a function, transforming input values for the leaf variables into an output value for the root, by following two simple rules: (1) The value of any $+$-node is the sum of the values of its children. (2) The value of any $\times$-node is the product of the values of its children.

Two arithmetic expression trees are ***equivalent*** if they represent the same function; that is, the same input values for the leaf variables always leads to the same output value at both roots. An arithmetic expression tree is in ***normal form*** if the parent of every $+$-node (if any) is another $+$-node.



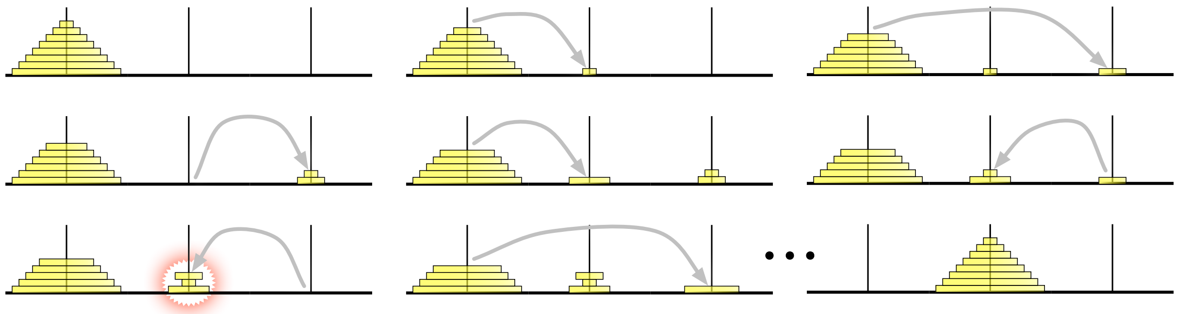
Three equivalent expression trees. Only the third is in normal form.

Prove that for any arithmetic expression tree, there is an equivalent arithmetic expression tree in normal form.

*6. ***[Extra credit]*** You may be familiar with the story behind the famous Tower of Hanoï puzzle:

> At the great temple of Benares, there is a brass plate on which three vertical diamond shafts are fixed. On the shafts are mounted n golden disks of decreasing size. At the time of creation, the god Brahma placed all of the disks on one pin, in order of size with the largest at the bottom. The Hindu priests unceasingly transfer the disks from peg to peg, one at a time, never placing a larger disk on a smaller one. When all of the disks have been transferred to the last pin, the universe will end.

Recently the temple at Benares was relocated to southern California, where the monks are considerably more laid back about their job. At the "Towers of Hollywood", the golden disks have been replaced with painted plywood, and the diamond shafts have been replaced with Plexiglas. More importantly, the restriction on the order of the disks has been relaxed. While the disks are being moved, the bottom disk on any pin must be the *largest* disk on that pin, but disks further up in the stack can be in any order. However, after all the disks have been moved, they must be in sorted order again.



The Towers of Hollywood. The sixth move leaves the disks out of order.

Describe an algorithm that moves a stack of n disks from one pin to the another using the smallest possible number of moves. *Exactly* how many moves does your algorithm perform? *[Hint: The Hollywood monks can bring about the end of the universe considerably faster than their Benaresian counterparts.]*

CS 473: Undergraduate Algorithms, Spring 2009 Homework 1

Due Tuesday, February 3, 2009 at 11:59:59pm.

- Groups of up to three students may submit a single, common solution for this and all future homeworks. Please clearly write every group member's name and NetID on every page of your submission.

1. The traditional Devonian/Cornish drinking song "The Barley Mow" has the following pseudolyrics, where *container*$[i]$ is the name of a container that holds 2^i ounces of beer. One version of the song uses the following containers: nipperkin, gill pot, half-pint, pint, quart, pottle, gallon, half-anker, anker, firkin, half-barrel, barrel, hogshead, pipe, well, river, and ocean. (Every container in this list is twice as big as its predecessor, except that a firkin is actually 2.25 ankers, and the last three units are just silly.)

```
BarleyMow(n):
  "Here's a health to the barley-mow, my brave boys,"
  "Here's a health to the barley-mow!"

  "We'll drink it out of the jolly brown bowl,"
  "Here's a health to the barley-mow!"
  "Here's a health to the barley-mow, my brave boys,"
  "Here's a health to the barley-mow!"

  for $i \leftarrow 1$ to $n$
      "We'll drink it out of the container[$i$], boys,"
      "Here's a health to the barley-mow!"
      for $j \leftarrow i$ downto 1
          "The container[$j$],"
      "And the jolly brown bowl!"
      "Here's a health to the barley-mow!"
      "Here's a health to the barley-mow, my brave boys,"
      "Here's a health to the barley-mow!"
```

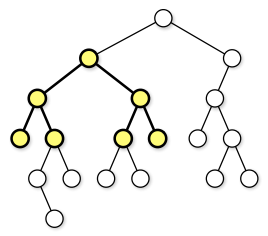
   (a) Suppose each container name *container*$[i]$ is a single word, and you can sing four words a second. How long would it take you to sing BarleyMow(n)? (Give a tight asymptotic bound.) *[Hint: Is 'barley-mow' one word or two? Does it matter?]*

   (b) If you want to sing this song for $n > 20$, you'll have to make up your own container names. To avoid repetition, these names will get progressively longer as n increases[1]. Suppose *container*$[n]$ has $\Theta(\log n)$ syllables, and you can sing six syllables per second. Now how long would it take you to sing BarleyMow(n)? (Give a tight asymptotic bound.)

   (c) Suppose each time you mention the name of a container, you actually drink the corresponding amount of beer: one ounce for the jolly brown bowl, and 2^i ounces for each *container*$[i]$. Assuming for purposes of this problem that you are at least 21 years old, *exactly* how many ounces of beer would you drink if you sang BarleyMow(n)? (Give an *exact* answer, not just an asymptotic bound.)

[1]"We'll drink it out of the hemisemidemiyottapint, boys!"

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



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

3. (a) Describe and analyze a recursive algorithm to reconstruct a binary tree, given its preorder and postorder node sequences (as in Homework 0, problem 1).

   (b) Describe and analyze a recursive algorithm to reconstruct a binary tree, given its preorder and *inorder* node sequences.

CS 473: Undergraduate Algorithms, Spring 2009
Homework 10

Due Tuesday, May 5, 2009 at 11:59:59pm

- Groups of up to three students may submit a single, common solution. Please clearly write every group member's name and NetID on every page of your submission.
- ***This homework is optional.*** If you submit solutions, they will be graded, and your overall homework grade will be the average of ten homeworks (Homeworks 0–10, dropping the lowest). If you do not submit solutions, your overall homework grade will be the average of *nine* homeworks (Homeworks 0–9, dropping the lowest).

---

1. Suppose you are given a magic black box that can determine ***in polynomial time***, given an arbitrary graph G, the number of vertices in the largest complete subgraph of G. Describe and analyze a ***polynomial-time*** algorithm that computes, given an arbitrary graph G, a complete subgraph of G of maximum size, using this magic black box as a subroutine.

2. PlanarCircuitSAT is a special case of CircuitSAT where the input circuit is drawn 'nicely' in the plane — no two wires cross, no two gates touch, and each wire touches only the gates it connects. (Not every circuit can be drawn this way!) As in the general CircuitSAT problem, we want to determine if there is an input that makes the circuit output True?

   Prove that PlanarCircuitSAT is NP-complete. *[Hint:* Xor.*]*

3. For each problem below, either describe a polynomial-time algorithm or prove that the problem is NP-complete.

   (a) A *double-Eulerian* circuit in an undirected graph G is a closed walk that traverses every edge in G exactly twice. Given a graph G, does G have a *double-Eulerian* circuit?

   (b) A *double-Hamiltonian* circuit in an undirected graph G is a closed walk that visits every vertex in G exactly twice. Given a graph G, does G have a *double-Hamiltonian* circuit?

CS 473: Undergraduate Algorithms, Spring 2009
Homework 2

Written solutions due Tuesday, February 10, 2009 at 11:59:59pm.

- Roughly 1/3 of the students will give oral presentations of their solutions to the TAs. ***Please check Compass to check whether you are supposed give an oral presentation for 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.
- ***For this homework only:*** These homework problems ask you to describe recursive backtracking algorithms for various problems. ***Don't*** use memoization or dynamic programming to make your algorithms more efficient; you'll get to do that on HW3. ***Don't*** analyze the running times of your algorithms. The ***only*** things you should submit for each problem are (1) a description of your recursive algorithm, and (2) a *brief* justification for its correctness.

---

1. A ***basic arithmetic expression*** is composed of characters from the set $\{1, +, \times\}$ and parentheses. Almost every integer can be represented by more than one basic arithmetic expression. For example, all of the following basic arithmetic expression represent the integer 14:

$$1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1$$
$$((1+1)\times(1+1+1+1+1))+((1+1)\times(1+1))$$
$$(1+1)\times(1+1+1+1+1+1+1)$$
$$(1+1)\times(((1+1+1)\times(1+1))+1)$$

Describe a recursive algorithm to compute, given an integer n as input, the minimum number of 1's in a basic arithmetic expression whose value is n. The number of parentheses doesn't matter, just the number of 1's. For example, when $n = 14$, your algorithm should return 8, for the final expression above.

2. A sequence $A = \langle a_1, a_2, \ldots, a_n\rangle$ is ***bitonic*** if there is an index i with $1 < i < n$, such that the prefix $\langle a_1, a_2, \ldots, a_i\rangle$ is strictly increasing and the suffix $\langle a_i, a_{i+1}, \ldots, a_n\rangle$ is strictly decreasing. In particular, a bitonic sequence must contain at least three elements.

   Describe a recursive algorithm to compute, given a sequence A, the length of the longest bitonic subsequence of A.

3. A ***palindrome*** is a string that reads the same forwards and backwards, like x, pop, noon, redivider, or amanaplanacatahamayakayamahatacanalpanama. Any string can be broken into sequence of palindromes. For example, the string bubbaseesabanana ('Bubba sees a banana.') can be broken into palindromes in several different ways; for example:

   bub + baseesab + anana
   b + u + bb + a + sees + aba + nan + a
   b + u + bb + a + sees + a + b + anana
   b + u + b + b + a + s + e + e + s + a + b + a + n + a + n + a

   Describe a recursive algorithm to compute the minimum number of palindromes that make up a given input string. For example, given the input string bubbaseesabanana, your algorithm would return the integer 3.

CS 473: Undergraduate Algorithms, Spring 2009 Homework 3

Written solutions due Tuesday, February 17, 2009 at 11:59:59pm.

---

1. Redo Homework 2, but now with dynamic programming!

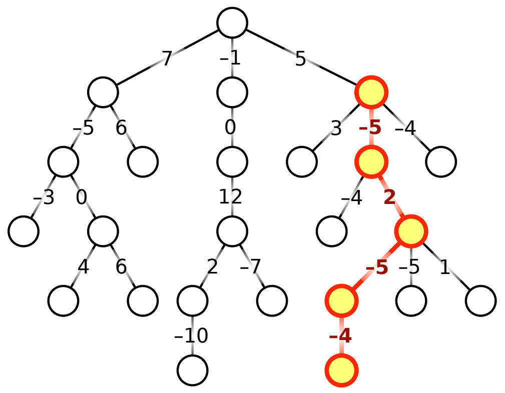
   (a) Describe and analyze an efficient algorithm to compute the minimum number of 1's in a basic arithmetic expression whose value is a given positive integer.

   (b) Describe and analyze an efficient algorithm to compute the length of the longest bitonic subsequence of a given input sequence.

   (c) Describe and analyze an efficient algorithm to compute the minimum number of palindromes that make up a given input string.

   Please see Homework 2 for more detailed descriptions of each problem. ***Solutions for Homework 2 will be posted Friday, after the HW2 oral presentations.*** You may (and should!) use anything from those solutions without justification.

2. Let T be a rooted tree with integer weights on its edges, which could be positive, negative, or zero. Design an algorithm to find the minimum-length path from a node in T down to one of its descendants. The length of a path is the sum of the weights of its edges. For example, given the tree shown below, your algorithm should return the number -12. For full credit, your algorithm should run in $O(n)$ time.



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

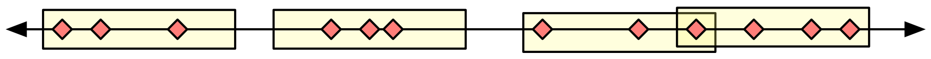
3. Describe and analyze an efficient algorithm to compute the longest common subsequence of *three* given strings. For example, given the input strings **E**P**I**D**E**M**I**OL**O**GIST, R**E**FR**I**G**E**RAT**IO**N, and SUP**E**RCALIFRAGIL**I**STIC**E**XP**I**ALOD**O**CIOUS, your algorithm should return the number 5, because the longest common subsequence is **EIEIO**.

CS 473: Undergraduate Algorithms, Spring 2009
Homework 3½

Practice only

1. After graduating from UIUC, you are hired by a mobile phone company to plan the placement of new cell towers along a long, straight, nearly-deserted highway out west. Each cell tower can transmit the same fixed distance from its location. Federal law requires that any building along the highway must be within the broadcast range of at least one tower. On the other hand, your company wants to build as few towers as possble. Given the locations of the buildings, where should you build the towers?

   More formally, suppose you are given a set $X = \{x_1, x_2, \ldots, x_n\}$ of points on the real number line. Describe an algorithm to compute the minimum number of intervals of length 1 that can cover all the points in X. For full credit, your algorithm should run in $O(n \log n)$ time.

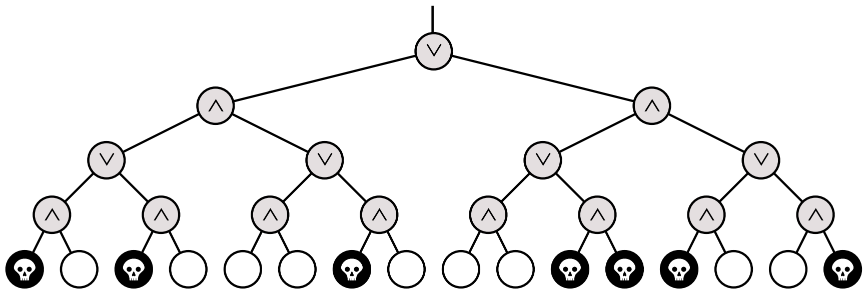

A set of points that can be covered by four unit intervals.

2. (a) The *left spine* of a binary tree is a path starting at the root and following only left-child pointers down to a leaf. What is the expected number of nodes in the left spine of an n-node treap?

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

   (c) Prove that the expected number of proper descendants of any node in a treap is exactly equal to the expected depth of that node.

3. Death knocks on your door one cold blustery morning and challenges you to a game. Death knows that 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 nodes at even levels (where it's your turn) are OR gates, the nodes at odd levels (where it's Death's turn) are AND gates. Each gate gets its input from its children and passes its output to its parent. White and black stand for TRUE and FALSE. 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.

(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) Describe 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!]*

CS 473: Undergraduate Algorithms, Spring 2009
Homework 3

Written solutions due Tuesday, March 2, 2009 at 11:59:59pm.

1. 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($Q_1, Q_2$):
  if $Q_1$ is empty return $Q_2$
  if $Q_2$ is empty return $Q_1$
  if $key(Q_1) > key(Q_2)$
      swap $Q_1 \leftrightarrow Q_2$
  with probability 1/2
      $left(Q_1) \leftarrow$ Meld($left(Q_1), Q_2$)
  else
      $right(Q_1) \leftarrow$ Meld($right(Q_1), Q_2$)
  return $Q_1$
```

   (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 is the total number of nodes in both trees. *[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.)

2. Recall that a *priority search tree* is a binary tree in which every node has both a *search key* and a *priority*, arranged so that the tree is simultaneously a binary search tree for the keys and a min-heap for the priorities. A *heater* is a priority search tree in which the priorities are given by the user, and the search keys are distributed uniformly and independently at random in the real interval $[0,1]$. Intuitively, a heater is the 'opposite' of a treap.

   The following problems consider an n-node heater T whose node priorities are the integers from 1 to n. We identify nodes in T by their priorities; thus, 'node 5' means the node in T with priority 5. The min-heap property implies that node 1 is the root of T. Finally, let i and j be integers with $1 \le i < j \le n$.

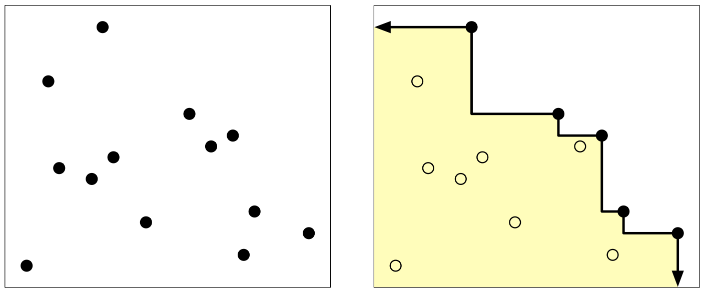
   (a) ***Prove*** that in a random permutation of the $(i+1)$-element set $\{1, 2, \ldots, i, j\}$, elements i and j are adjacent with probability $2/(i+1)$.

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

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

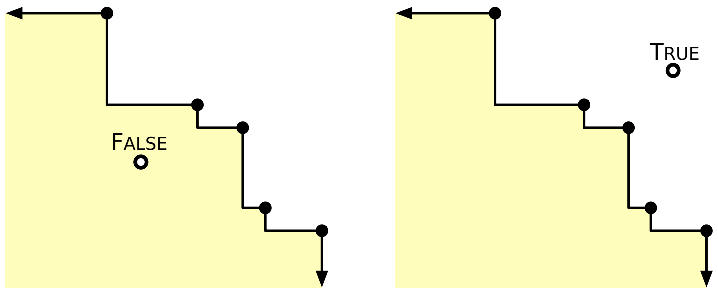
   (d) What is the *exact* expected depth of node j?

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.

CS 473: Undergraduate Algorithms, Spring 2009
Homework 5

Written solutions due Tuesday, March 9, 2009 at 11:59:59pm.

1. Remember the difference between stacks and queues? Good.

   (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 methods PUSH and POP.

   (b) A *quack* is an abstract data type that combines 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:

   - **Push:** add a new item to the left end of the list;
   - **Pop:** remove the item on the left end of the list;
   - **Pull:** 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 push, pop, or pull operation is $O(1)$. Again, you are *only* allowed to access the stacks through the standard methods PUSH and POP.

2. In a *dirty* binary search tree, each node is labeled either *clean* or *dirty*. The lazy deletion scheme used for scapegoat trees requires us to *purge* the search tree, keeping all the clean nodes and deleting all the dirty nodes, as soon as half the nodes become dirty. In addition, the purged tree should be perfectly balanced.

   Describe and analyze an algorithm to purge an *arbitrary* n-node dirty binary search tree in $O(n)$ time, using at most $O(\log n)$ space (in addition to the tree itself). Don't forget to include the recursion stack in your space bound. An algorithm that uses $\Theta(n)$ additional space in the worst case is worth half credit.

3. Some applications of binary search trees attach a *secondary data structure* to each node in the tree, to allow for more complicated searches. Maintaining these secondary structures usually complicates algorithms for keeping the top-level search tree balanced.

   Let T be an arbitrary binary tree. Suppose every node v in T stores a secondary structure of size $O(size(v))$, which can be built in $O(size(v))$ time, where $size(v)$ denotes the number of descendants of v. Performing a rotation at any node v now requires $O(size(v))$ time, because we have to rebuild one of the secondary structures.

   (a) **[1 pt]** Overall, how much space does this data structure use *in the worst case*?

   (b) **[1 pt]** How much space does this structure use if the primary search tree T is perfectly balanced?

   (c) **[2 pts]** Suppose T is a splay tree. Prove that the *amortized* cost of a splay (and therefore of a search, insertion, or deletion) is $\Omega(n)$. *[Hint: This is easy!]*

(d) **[3 pts]** Now suppose T is a scapegoat tree, and that rebuilding the subtree rooted at v requires $\Theta(\mathit{size}(v) \log \mathit{size}(v))$ time (because we also have to rebuild the secondary structures at every descendant of v). What is the *amortized* cost of inserting a new element into T?

(e) **[3 pts]** Finally, suppose T is a treap. What's the worst-case *expected* time for inserting a new element into T?

CS 473: Undergraduate Algorithms, Spring 2009
Homework 6

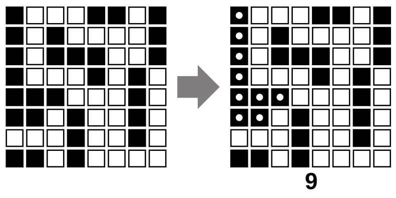
Written solutions due Tuesday, March 17, 2009 at 11:59:59pm.

1. Let G be an undirected graph with n nodes. Suppose that G contains two nodes s and t, such that every path from s to t contains more than $n/2$ edges.
   (a) Prove that G must contain a vertex v that lies on *every* path from s to t.
   (b) Describe an algorithm that finds such a vertex v in $O(V + E)$ time.

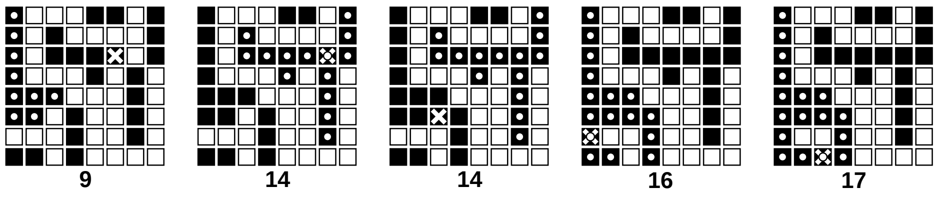
2. Suppose you are given a graph G with weighted edges and a minimum spanning tree T of G.
   (a) Describe an algorithm to update the minimum spanning tree when the weight of a single edge e is decreased.
   (b) Describe an algorithm to update the minimum spanning tree when the weight of a single edge 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. (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?

CS 473: Undergraduate Algorithms, Spring 2009
Homework 6½

Practice only—do not submit solutions

1. In class last Tuesday, we discussed Ford's generic shortest-path algorithm—relax arbitrary tense edges until no edge is tense. This problem asks you to fill in part of the proof that this algorithm is correct.

   (a) Prove that after *every* call to RELAX, for every vertex v, either $dist(v) = \infty$ or $dist(v)$ is the total weight of some path from s to v.

   (b) Prove that for every vertex v, when the generic algorithm halts, either $pred(v) = \text{NULL}$ and $dist(v) = \infty$, or $dist(v)$ is the total weight of the predecessor chain ending at v:

   $$s \to \cdots \to pred(pred(v)) \to pred(v) \to v.$$

2. Describe a modification of Shimbel's shortest-path algorithm that actually computes a negative-weight cycle if any such cycle is reachable from s, or a shortest-path tree rooted at s if there is no such cycle. Your modified algorithm should still run in $O(VE)$ time.

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 all the flights on the planet.

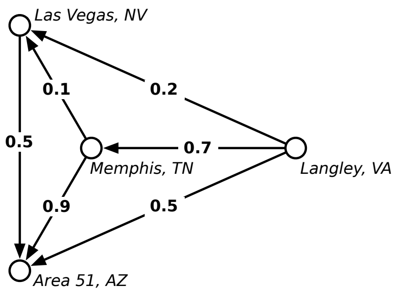
   Suppose one of your customers wants to fly from city X to city Y. Describe an algorithm to find a sequence of flights that minimizes the *total time in transit*—the length of time from the initial departure to the final arrival, including time at intermediate airports waiting for connecting flights.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

CS 473: Undergraduate Algorithms, Spring 2009 Homework 6¾

Practice only—do not submit solutions

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

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

   For example, with the probabilities shown above, if Mulder tries to drive directly from Langley to Area 51, he has a 50% chance of getting there without being abducted. If he stops in Memphis, he has a $0.7 \times 0.9 = 63\%$ chance of arriving safely. If he stops first in Memphis and then in Las Vegas, he has a $1 - 0.7 \times 0.1 \times 0.5 = 96.5\%$ chance of being abducted! (That's how they got Elvis, you know.)

2. Let $G = (V, E)$ be a directed graph with weighted edges; edge weights could be positive, negative, or zero. Suppose the vertices of G are partitioned into k disjoint subsets $V_1, V_2, \ldots, V_k$; that is, every vertex of G belongs to exactly one subset V_i. For each i and j, let $\delta(i, j)$ denote the minimum shortest-path distance between any vertex in V_i and any vertex in V_j:

   $$\delta(i, j) = \min\{dist(u, v) \mid u \in V_i \text{ and } v \in V_j\}.$$

   Describe an algorithm to compute $\delta(i, j)$ for *all* i and j in time $O(VE + kE \log E)$. The output from your algorithm is a $k \times k$ array.

3. Recall[1] that a deterministic finite automaton (DFA) is formally defined as a tuple $M = (\Sigma, Q, q_0, F, \delta)$, where the finite set Σ is the input alphabet, the finite set Q is the set of states, $q_0 \in Q$ is the start state, $F \subseteq Q$ is the set of final (accepting) states, and $\delta\colon Q \times \Sigma \to Q$ is the transition function. Equivalently, a DFA is a directed (multi-)graph with labeled edges, such that each symbol in Σ is the label of exactly one edge leaving any vertex. There is a special 'start' vertex q_0, and a subset of the vertices are marked as 'accepting states'. Any string in Σ^* describes a unique walk starting at q_0.

   Stephen Kleene[2] proved that the language accepted by any DFA is identical to the language described by some regular expression. This problem asks you to develop a variant of the Floyd-Warshall all-pairs shortest path algorithm that computes a regular expression that is equivalent to the language accepted by a given DFA.

   Suppose the input DFA M has n states, numbered from 1 to n, where (without loss of generality) the start state is state 1. Let $L(i, j, r)$ denote the set of all words that describe walks in M from state i to state j, where every intermediate state lies in the subset $\{1, 2, \ldots, r\}$; thus, the language accepted by the DFA is exactly
$$\bigcup_{q \in F} L(1, q, n).$$
   Let $R(i, j, r)$ be a regular expression that describes the language $L(i, j, r)$.

   (a) What is the regular expression $R(i, j, 0)$?

   (b) Write a recurrence for the regular expression $R(i, j, r)$ in terms of regular expressions of the form $R(i', j', r - 1)$.

   (c) Describe a polynomial-time algorithm to compute $R(i, j, n)$ for all states i and j. (Assume that you can concatenate two regular expressions in $O(1)$ time.)

[1]No, really, you saw this in CS 273/373.

[2]Pronounced 'clay knee', not 'clean' or 'clean-ee' or 'clay-nuh' or 'dimaggio'.

CS 473: Undergraduate Algorithms, Spring 2009
Homework 7

Due Tuesday, April 14, 2009 at 11:59:59pm.

- Groups of up to three students may submit a single, common solution for this and all future homeworks. Please clearly write every group member's name and NetID on every page of your submission.

---

1. A graph is *bipartite* if its vertices can be colored black or white such that every edge joins vertices of two different colors. A graph is *d-regular* if every vertex has degree d. A *matching* in a graph is a subset of the edges with no common endpoints; a matching is *perfect* if it touches every vertex.

   (a) Prove that every regular bipartite graph contains a perfect matching.

   (b) Prove that every d-regular bipartite graph is the union of d perfect matchings.

2. Let $G = (V, E)$ be a directed graph where for each vertex v, the in-degree of v and out-degree of v are equal. Let u and v be two vertices G, and suppose G contains k edge-disjoint paths from u to v. Under these conditions, must G also contain k edge-disjoint paths from v to u? Give a proof or a counterexample with explanation.

3. A flow f is called ***acyclic*** if the subgraph of directed edges with positive flow contains no directed cycles. A flow is *positive* if its value is greater than 0.

   (a) A *path flow* assigns positive values only to the edges of one simple directed path from s to t. Prove that every positive acyclic flow can be written as the sum of a finite number of path flows.

   (b) Describe a flow in a directed graph that *cannot* be written as the sum of path flows.

   (c) A *cycle flow* assigns positive values only to the edges of one simple directed cycle. Prove that every flow can be written as the sum of a finite number of path flows and cycle flows.

   (d) Prove that for any flow f, there is an acyclic flow with the same value as f. (In particular, this implies that some maximum flow is acyclic.)

CS 473: Undergraduate Algorithms, Spring 2009 Homework 8

Due Tuesday, April 21, 2009 at 11:59:59pm.

- Groups of up to three students may submit a single, common solution for this and all future homeworks. Please clearly write every group member's name and NetID on every page of your submission.

1. 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 non exists. *[Hint: Use bipartite matching!]*

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. *Ad-hoc networks* are made up of cheap, low-powered wireless devices. In principle[1], 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 several simple devices could be distributed randomly in the area of interest (for example, dropped from an airplane), and then they would somehow automatically configure themselves into an efficiently functioning wireless network.

   The 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 all its information to some other *backup* device within its communication range. To improve reliability, 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.

   Suppose we are given the communication distance 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 and analyze an algorithm that either computes a backup set of size k for each of the n devices, such that that no device appears in more than b backup sets, or correctly reports that no good collection of backup sets exists.

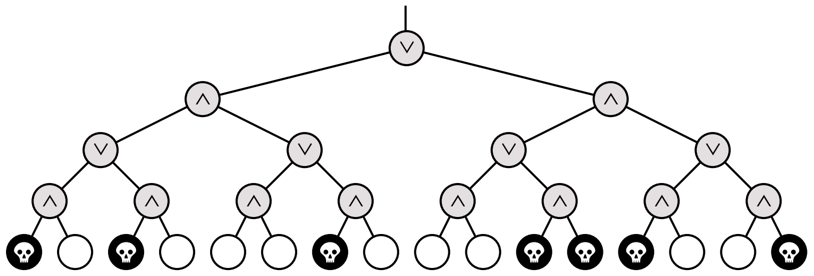
[1]but not so much in practice

CS 473: Undergraduate Algorithms, Spring 2009 Homework 9

Due Tuesday, April 28, 2009 at 11:59:59pm.

- Groups of up to three students may submit a single, common solution for this and all future homeworks. Please clearly write every group member's name and NetID on every page of your submission.

1. Death knocks on your door one cold blustery morning and challenges you to a game. Death knows that 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 is white, you will live forever. You move first, so Death gets the last turn.



You can decide whether it is worth playing or not as follows. Imagine that the nodes at even levels (where it is your turn) are OR gates, the nodes at odd levels (where it is Death's turn) are AND gates. Each gate gets its input from its children and passes its output to its parent. White and black leaves stand represent TRUE and FALSE inputs, respectively. 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.

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

(b) Prove that *every* deterministic algorithm must examine *every* leaf of the tree in the worst case. Since there are 4^n leaves, this implies that any deterministic algorithm must take $\Omega(4^n)$ time in the worst case. Use an adversary argument; in other words, assume that Death cheats.

(c) ***[Extra credit]*** Describe a *randomized* algorithm that runs in $O(3^n)$ expected time.

2. 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.

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.

   *(e) **[Extra credit!]** Describe an algorithm to find the lowest unsafe floor using k eggs, using as few tests as possible, and prove your algorithm is optimal for all values of k.

CS 473: Undergraduate Algorithms, Spring 2009
Head Banging Session 0

January 20 and 21, 2009

---

1. Solve the following recurrences. If base cases are provided, find an *exact* closed-form solution. Otherwise, find a solution of the form $\Theta(f(n))$ for some function f.

   - **Warmup:** You should be able to solve these almost as fast as you can write down the answers.
     (a) $A(n) = A(n-1) + 1$, where $A(0) = 0$.
     (b) $B(n) = B(n-5) + 2$, where $B(0) = 17$.
     (c) $C(n) = C(n-1) + n^2$
     (d) $D(n) = 3\,D(n/2) + n^2$
     (e) $E(n) = 4\,E(n/2) + n^2$
     (f) $F(n) = 5\,F(n/2) + n^2$

   - **Real practice:**
     (a) $A(n) = A(n/3) + 3A(n/5) + A(n/15) + n$
     (b) $B(n) = \min_{0<k<n} (B(k) + B(n-k) + n)$
     (c) $C(n) = \max_{n/4<k<3n/4} (C(k) + C(n-k) + n)$
     (d) $D(n) = \max_{0<k<n} \big(D(k) + D(n-k) + k(n-k)\big)$, where $D(1) = 0$
     (e) $E(n) = 2E(n-1) + E(n-2)$, where $E(0) = 1$ and $E(1) = 2$
     (f) $F(n) = \dfrac{1}{F(n-1)\,F(n-2)}$, where $F(0) = 1$ and $F(2) = 2$
     *(g) $G(n) = n\,G(\sqrt{n}) + n^2$

2. The *Fibonacci numbers* F_n are defined recursively 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:

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

3. Whenever groups of pigeons gather, they instinctively establish a *pecking order*. For any pair of pigeons, one pigeon always pecks the other, driving it away from food or potential mates. The same pair of pigeons always chooses the same pecking order, even after years of separation, no matter what other pigeons are around. Surprisingly, the overall pecking order can contain cycles—for example, pigeon A pecks pigeon B, which pecks pigeon C, which pecks pigeon A.

   Prove that any finite set of pigeons can be arranged in a row from left to right so that every pigeon pecks the pigeon immediately to its left. Pretty please.

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 an algorithm to count the number of inversions in an n-element array in $O(n \log n)$ time.

2. (a) Prove that the following algorithm actually sorts its input.

```
StoogeSort(A[0..n−1]):
  if n = 2 and A[0] > A[1]
      swap A[0] ⟷ A[1]
  else if n > 2
      m = ⌈2n/3⌉
      StoogeSort(A[0..m−1])
      StoogeSort(A[n−m..n−1])
      StoogeSort(A[0..m−1])
```

   (b) Would StoogeSort still sort correctly if we replaced $m = \lceil 2n/3 \rceil$ with $m = \lfloor 2n/3 \rfloor$? Justify your answer.

   (c) State a recurrence (including base case(s)) for the number of comparisons executed by StoogeSort.

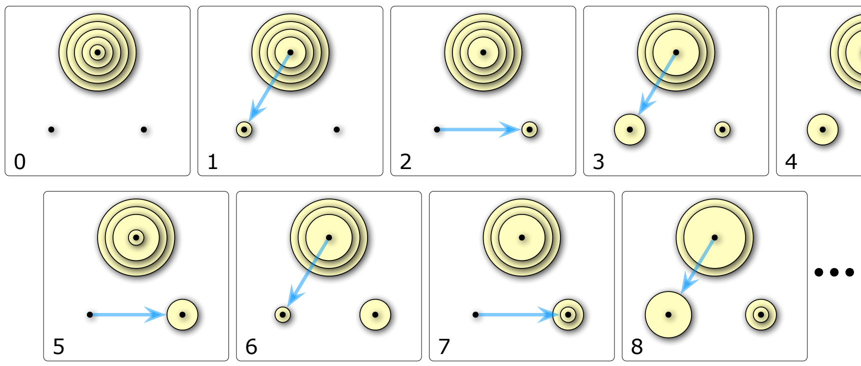
   (d) Solve this recurrence. *[Hint: Ignore the ceiling.]*

   (e) **To think about on your own:** Prove that the number of *swaps* executed by StoogeSort is at most $\binom{n}{2}$.

3. Consider the following restricted variants of the Tower of Hanoi puzzle. In each problem, the needles are numbered 0, 1, and 2, and your task is to move a stack of n disks from needle 1 to needle 2.

   (a) Suppose you are forbidden to move any disk directly between needle 1 and needle 2; *every* move must involve needle 0. Describe an algorithm to solve this version of the puzzle using as few moves as possible. *Exactly* how many moves does your algorithm make?

   (b) Suppose you are only allowed to move disks from needle 0 to needle 2, from needle 2 to needle 1, or from needle 1 to needle 0. Equivalently, Suppose the needles are arranged in a circle and numbered in clockwise order, and you are only allowed to move disks counterclockwise. Describe an algorithm to solve this version of the puzzle using as few moves as possible. *Exactly* how many moves does your algorithm make?



The first eight moves in a counterclockwise Towers of Hanoi solution

★(c) Finally, suppose you are forbidden to move any disk directly from needle 1 to 2, but any other move is allowed. Describe an algorithm to solve this version of the puzzle using as few moves as possible. *Exactly* how many moves does your algorithm make?

[Hint: This version is ***considerably*** *harder than the other two.]*

CS 473: Undergraduate Algorithms, Spring 2009
HBS 10

1. Consider the following problem, called *BOX-DEPTH*: Given a set of n axis-aligned rectangles in the plane, how big is the largest subset of these rectangles that contain a common point?

   (a) Describe a polynomial-time reduction from *BOX-DEPTH* to *MAX-CLIQUE*.

   (b) Describe and analyze a polynomial-time algorithm for *BOX-DEPTH*. [Hint: $O(n^3)$ time should be easy, but $O(n \log n)$ time is possible.]

   (c) Why don't these two results imply that $P = NP$?

2. Suppose you are given a magic black box that can determine in polynomial time, given an arbitrary weighted graph G, the length of the shortest Hamiltonian cycle in G. Describe and analyze a polynomial-time algorithm that computes, given an arbitrary weighted graph G, the shortest Hamiltonian cycle in G, using this magic black box as a subroutine.

3. Prove that the following problems are NP-complete.

   (a) Given an undirected graph G, does G have a spanning tree in which every node has degree at most 17?

   (b) Given an undirected graph G, does G have a spanning tree with at most 42 leaves?

CS 473: Undergraduate Algorithms, Spring 2009
HBS 11

1. You step in a party with a camera in your hand. Each person attending the party has some friends there. You want to have exactly one picture of each person in your camera. You want to use the following protocol to collect photos. At each step, the person that has the camera in his hand takes a picture of one of his/her friends and pass the camera to him/her. Of course, you only like the solution if it finishes when the camera is in your hand. Given the friendship matrix of the people in the party, design a polynomial algorithm that decides whether this is possible, or prove that this decision problem is NP-hard.

2. A boolean formula is in disjunctive normal form (DNF) if it is a disjunctions (OR) of several clauses, each of which is the conjunction (AND) of several literals, each of which is either a variable or its negation. For example:
$$(a \wedge b \wedge c) \vee (\bar{a} \wedge b) \vee (\bar{c} \wedge x)$$
Given a DNF formula give a polynomial algorithm to check whether it is satisfiable or not. Why this does not imply $P = NP$.

3. Prove that the following problems are NP-complete.
   (a) Given an undirected graph G, does G have a spanning tree in which every node has degree at most 17?
   (b) Given an undirected graph G, does G have a spanning tree with at most 42 leaves?

CS 473: Undergraduate Algorithms, Spring 2009
HBS 2

1. Consider two horizontal lines l_1 and l_2 in the plane. There are n points on l_1 with x-coordinates $A = a_1, a_2, \dots, a_n$ and there are n points on l_2 with x-coordinates $B = b_1, b_2, \dots, b_n$. Design an algorithm to compute, given A and B, a largest set S of non-intersecting line segments subject to the following restrictions:

   (a) Any segment in S connects a_i to b_i for some $i (1 \leq i \leq n)$.

   (b) Any two segments in S do not intersect.

2. Consider a $2^n x 2^n$ chess board with one (arbitrarily chosen) square removed. Prove that any such chessboard can be tiled without gaps or overlaps by L-shaped pieces of 3 squares each. Can you give an algorithm to do the tiling?

3. Given a string of letters $Y = y_1 y_2 \dots y_n$, a segmentation of Y is a partition of its letters into contiguous blocks of letters (also called words). Each word has a quality that can be computed by a given oracle (e.g. you can call *quality("meet")* to get the quality of the word "meet"). The quality of a segmentation is equal to the sum over the qualities of its words. Each call to the oracle takes linear time in terms of the argument; that is quality(S) takes $O(|S|)$.

   Using the given oracle, give an algorithm that takes a string Y and computes a segmentation of maximum total quality.

CS 473: Undergraduate Algorithms, Spring 2009
HBS 3

1. Change your recursive solutions for the following problems to efficient algorithms (Hint: use dynamic programming!).

   (a) Consider two horizontal lines l_1 and l_2 in the plane. There are n points on l_1 with x-coordinates $A = a_1, a_2, \ldots, a_n$ and there are n points on l_2 with x-coordinates $B = b_1, b_2, \ldots, b_n$. Design an algorithm to compute, given A and B, a largest set S of non-intersecting line segments subject to the following restrictions:

      i. Any segment in S connects a_i to b_i for some $i (1 \leq i \leq n)$.

      ii. Any two segments in S do not intersect.

   (b) Given a string of letters $Y = y_1 y_2 \ldots y_n$, a segmentation of Y is a partition of its letters into contiguous blocks of letters (also called words). Each word has a quality that can be computed by a given oracle (e.g. you can call *quality("meet")* to get the quality of the word "meet"). The quality of a segmentation is equal to the sum over the qualities of its words. Each call to the oracle takes linear time in terms of the argument; that is quality(S) takes $O(|S|)$.

      Using the given oracle, give an algorithm that takes a string Y and computes a segmentation of maximum total quality.

2. Give a polynomial time algorithm which given two strings A and B returns the longest sequence S that is a subsequence of A and B.

3. Consider a rooted tree T. Assume the root has a message to send to all nodes. At the beginning only the root has the message. If a node has the message, it can forward it to one of its children at each time step. Design an algorithm to find the minimum number of time steps required for the message to be delivered to all nodes.

CS 473: Undergraduate Algorithms, Spring 2009
HBS 3.5

1. Say you are given n jobs to run on a machine. Each job has a start time and an end time. If a job is chosen to be run, then it must start at its start time and end at its end time. Your goal is to accept as many jobs as possible, regardless of the job lengths, subject to the constraint that the processor can run at most one job at any given point in time. Provide an algorithm to do this with a running time that is polynomial in n. You may assume for simplicity that no two jobs have the same start or end times, but the start time and end time of two jobs can overlap.

2. 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).

3. Design and analyze an algorithm that return a permutation of the integers $\{1, 2, \ldots, n\}$ chosen uniformly at random.

CS 473: Undergraduate Algorithms, Spring 2009
HBS 4

1. Let x and y be two elements of a set S whose ranks differ by exactly r. Prove that in a treap for S, the expected length of the unique path from x to y is $O(\log r)$

2. Consider the problem of making change for n cents using the least number of coins.

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

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

   (c) Give a set of 4 coin values for which the greedy algorithm does not yield an optimal solution, show why.

   (d) Give a dynamic programming algorithm that yields an optimal solution for an arbitrary set of coin values.

3. A heater is a sort of dual treap, in which the priorities of the nodes are given, but their search keys are generate independently and uniformly from the unit interval $[0,1]$. You can assume all priorities and keys are distinct. Describe algorithms to perform the operations INSERT and DELETEMIN in a heater. What are the expected worst-case running times of your algorithms? In particular, can you express the expected running time of INSERT in terms of the priority rank of the newly inserted item?

CS 473: Undergraduate Algorithms, Spring 2009
HBS 5

1. Recall that the staircase of a set of points consists of the points with no other point both above and to the right. Describe a method to maintain the staircase as new points are added to the set. Specifically, describe and analyze a data structure that stores the staircase of a set of points, and an algorithm $INSERT(x, y)$ that adds the point (x, y) to the set and returns $TRUE$ or $FALSE$ to indicate whether the staircase has changed. Your data structure should use $O(n)$ space, and your INSERT algorithm should run in $O(\log n)$ amortized time.

2. In some applications, we do not know in advance how much space we will require. So, we start the program by allocating a (dynamic) table of some fixed size. Later, as new objects are inserted, we may have to allocate a larger table and copy the previous table to it. So, we may need more than $O(1)$ time for copying. In addition, we want to keep the table size small enough, avoiding a very large table to keep only few items. One way to manage a dynamic table is by the following rules:

   (a) Double the size of the table if an item is inserted into a full table

   (b) Halve the table size if a deletion causes the table to become less than 1/4 full

   Show that, in such a dynamic table we only need $O(1)$ amortized time, per operation.

3. Consider a stack data structure with the following operations:

   - PUSH(x): adds the element x to the top of the stack
   - POP: removes and returns the element that is currently on top of the stack (if the stack is non-empty)
   - SEARCH(x): repeatedly removes the element on top of the stack until x is found or the stack becomes empty

   What is the amortized cost of an operation?

CS 473: Undergraduate Algorithms, Spring 2009
HBS 6

1. Let G be a connected graph and let v be a vertex in G. Show that T is both a DFS tree and a BFS tree rooted at v, then $G = T$.

2. An Euler tour of a graph G is a walk that starts from a vertex v, visits every edge of G exactly once and gets back to v. Prove that G has an Euler tour if and only if all the vertices of G has even degrees. Can you give an efficient algorithm to find an Euler tour of such a graph.

3. You are helping a group of ethnographers analyze some oral history data they have collected by interviewing members of a village to learn about the lives of people lived there over the last two hundred years. From the interviews, you have learned about a set of people, all now deceased, whom we will denote $P_1, P_2, \ldots, P_n$. The ethnographers have collected several facts about the lifespans of these people, of one of the following forms:

   (a) P_i died before P_j was born.

   (b) P_i and P_j were both alive at some moment.

   Naturally, the ethnographers are not sure that their facts are correct; memories are not so good, and all this information was passed down by word of mouth. So they'd like you to determine whether the data they have collected is at least internally consistent, in the sense that there could have existed a set of people for which all the facts they have learned simultaneously hold.

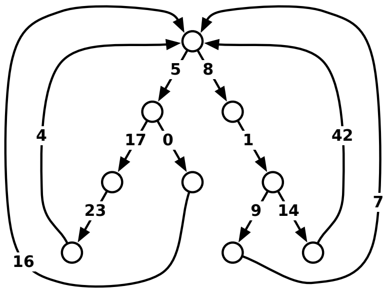
   Describe and analyze and algorithm to answer the ethnographers' problem. Your algorithm should either output possible dates of birth and death that are consistent with all the stated facts, or it should report correctly that no such dates exist.

CS 473: Undergraduate Algorithms, Spring 2009
HBS 6.5

1. (a) Describe and analyze and algorithm to find the *second smallest spanning tree* of a given graph G, that is, the spanning tree of G with smallest total weight except for the minimum spanning tree.

   *(b) Describe and analyze an efficient algorithm to compute, given a weighted undirected graph G and an integer k, the k smallest spanning trees of G.

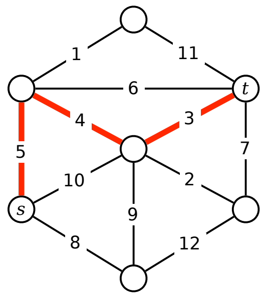
2. A *looped tree* is a weighted, directed graph built from a binary tree by adding an edge from every leaf back to the root. Every edge has a non-negative weight.



   (a) How much time would Dijkstra's algorithm require to compute the shortest path between two vertices u and v in a looped tree with n nodes?

   (b) Describe and analyze a faster algorithm.

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



The bottleneck distance between s and t is 5.

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

CS 473: Undergraduate Algorithms, Spring 2009
HBS 6.55

1. Suppose you are given a directed graph $G = (V, E)$ with non-negative edge lengths; $\ell(e)$ is the length of $e \in E$. You are interested in the shortest path distance between two given locations/nodes s and t. It has been noticed that the existing shortest path distance between s and t in G is not satisfactory and there is a proposal to add exactly one edge to the graph to improve the situation. The candidate edges from which one has to be chosen is given by $E' = \{e_1, e2, \ldots, e_k\}$ and you can assume that $E \cup E' = \emptyset$. The length of the e_i is $\alpha_i \geq 0$. Your goal is figure out which of these k edges will result in the most reduction in the shortest path distance from s to t. Describe an algorithm for this problem that runs in time $O((m+n)\log n + k)$ where $m = |E|$ and $n = |V|$. Note that one can easily solve this problem in $O(k(m+n)\log n)$ by running Dijkstra's algorithm k times, one for each G_i where G_i is the graph obtained by adding e_i to G.

2. Let G be an undirected graph with non-negative edge weights. Let s and t be two vertices such that the shortest path between s and t in G contains all the vertices in the graph. For each edge e, let $G\backslash e$ be the graph obtained from G by deleting the edge e. Design an $O(E \log V)$ algorithm that finds the shortest path distance between s and t in $G\backslash e$ for all e. [*Note that you need to output E distances, one for each graph $G\backslash e$*]

3. Given a Directed Acyclic Graph (DAG) and two vertices s and t you want to determine if there is an s to t path that includes at least k vertices.

CS 473: Undergraduate Algorithms, Spring 2009
HBS 7

1. Let $G = (V, E)$ be a directed graph with non-negative capacities. Give an efficient algorithm to check whether there is a unique max-flow on G?

2. Let $G = (V, E)$ be a graph and $s, t \in V$ be two specific vertices of G. We call $(S, T = V \backslash S)$ an (s, t)-cut if $s \in S$ and $t \in T$. Moreover, it is a minimum cut if the sum of the capacities of the edges that have one endpoint in S and one endpoint in T equals the maximum (s, t)-flow. Show that, both intersection and union of two min-cuts is a min-cut itself.

3. Let $G = (V, E)$ be a graph. For each edge e let $d(e)$ be a demand value attached to it. A flow is feasible if it sends more than $d(e)$ through e. Assume you have an oracle that is capable of solving the maximum flow problem. Give efficient algorithms for the following problems that call the oracle at most once.

   (a) Find a feasible flow.

   (b) Find a feasible flow of minimum possible value.

CS 473: Undergraduate Algorithms, Spring 2009
HBS 8

1. A box i can be specified by the values of its sides, say (i_1, i_2, i_3). We know all the side lengths are larger than 10 and smaller than 20 (i.e. $10 < i_1, i_2, i_3 < 20$). Geometrically, you know what it means for one box to nest in another: It is possible if you can rotate the smaller so that it fits inside the larger in each dimension. Of course, nesting is recursive, that is if i nests in j and j nests in k then i nests in k. After doing some nesting operations, we say a box is visible if it is not nested in any other one. Given a set of boxes (each specified by the lengthes of their sides) the goal is to find a set of nesting operations to minimize the number of visible boxes. Design and analyze an efficient algorithm to do this.

2. Let the number of papers submitted to a conference be n and the number of available reviewers be m. Each reviewer has a list of papers that he/she can review and each paper should be reviewed by three different persons. Also, each reviewer can review at most 5 papers. Design and analyze an algorithm to make the assignment or decide no feasible assignment exists.

3. Back in the euphoric early days of the Web, people liked to claim that much of the enormous potential in a company like Yahoo! was in the "eyeballs" - the simple fact that it gets millions of people looking at its pages every day. And further, by convincing people to register personal data with the site, it can show each user an extremely targeted advertisement whenever he or she visits the site, in away that TV networks or magazines could not hope to match. So if the user has told Yahoo! that he is a 20-year old computer science major from Cornell University, the site can throw up a banner ad for apartments in Ithaca, NY; on the other hand, if he is a 50-year-old investment banker from Greenwich, Connecticut, the site can display a banner ad pitching Lincoln Town Cars instead.

   But deciding on which ads to show to which people involves some serious computation behind the scenes. Suppose that the managers of a popular Web site have identified k distinct demographic groups $G_1, G_2, \ldots, G_k$. (These groups can overlap; for example G_1 can be equal to all residents of New York State, and G_2 can be equal to all people with a degree in computer science.) The site has contracts with m different advertisers, to show a certain number of copies of their ads to users of the site. Here is what the contract with the i^{th} advertiser looks like:

   (a) For a subset $X_i \subset \{G_1, \ldots, G_k\}$ of the demographic groups, advertiser i wants its ads shown only to users who belong to at least one of the demographic groups in the set X_i
   (b) For a number r_i , advertiser i wants its ads shown to at least r_i users each minute.

   Now, consider the problem of designing a good advertising policy - a way to show a single ad to each user of the site. Suppose at a given minute, there are n users visiting the site. Because we have registration information on each of these users, we know that user j(for $j = 1, 2, \ldots, n$) belongs to a subset $U_j \subset \{G_1, \ldots, G_k\}$ of the demographic groups. The problem is: is there a way to show a single ad to each user so that the site's contracts with each of the m advertisers is satisfied for this minute? (That is, for each $i = 1, 2, \ldots, m$, at least r_i of the n users, each belonging to at least one demographic group in X_i , are shown an ad provided by advertiser i.)

   Give an efficient algorithm to decide if this is possible, and if so, to actually choose an ad to show each user.

CS 473: Undergraduate Algorithms, Spring 2009
HBS 9

1. Prove that any algorithm to merge two sorted arrays, each of size n, requires at least $2n-1$ comparisons.

2. Suppose you want to determine the largest number in an n-element set $X = \{x_1, x_2, \ldots, x_n\}$, where each element x_i is an integer between 1 and $2^m - 1$. Describe an algorithm that solves this problem in $O(n+m)$ steps, where at each step, your algorithm compares one of the elements x_i with a *constant*. In particular, your algorithm must never actually compare two elements of X! *[Hint: Construct and maintain a nested set of 'pinning intervals' for the numbers that you have not yet removed from consideration, where each interval but the largest is either the upper half or lower half of the next larger block.]*

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. Prove that computing the staircase requires at least $\Omega(n \log n)$ comparisons in two ways,
   (a) Reduction from sorting.
   (b) Directly.

You have 90 minutes to answer four of the five questions.
Write your answers in the separate answer booklet.
You may take the question sheet with you when you leave.

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; each "I don't know" is worth $+1/4$ point. Your score will be rounded to the nearest *non-negative* integer.

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

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

   (c) How many digits are required to write 3^n in decimal?

   (d) What is the solution to the recurrence $D(n) = D(n/\pi) + \sqrt{2}$?

   (e) What is the solution to the recurrence $E(n) = E(n - \sqrt{2}) + \pi$?

   (f) What is the solution to the recurrence $F(n) = 4F(n/2) + 3n$?

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

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

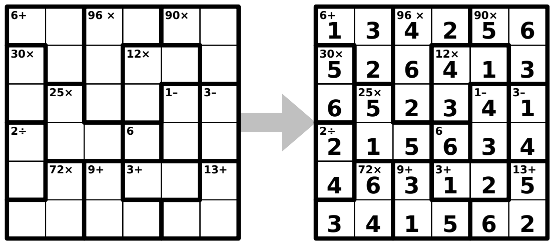
   (i) Let $H[1\,..\,n, 1\,..\,n]$ be a fixed array of numbers. Consider the following recursive function:

   $$Glub(i,j) = \begin{cases} 0 & \text{if } i = 0 \\ \infty & \text{if } i > n \text{ or } j = 0 \\ \max\left\{Glub(i-1,j),\ H[i,j] + Glub(i+1,j-1)\right\} & \text{otherwise} \end{cases}$$

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

   (j) What is the running time of the fastest possible algorithm to solve KenKen puzzles?

   A KenKen puzzle is a 6×6 grid, divided into regions called *cages*. Each cage is labeled with a numerical *value* and an arithmetic *operation*: $+$, $-$, $\times$, or $\div$. (The operation can be omitted if the cage consists of a single cell.) The goal is to place an integer between 1 and 6 in each grid cell, so that no number appears twice in any row or column, and the numbers inside each cage can be combined using *only* that cage's operation to obtain that cage's value. The solution is guaranteed to be unique.

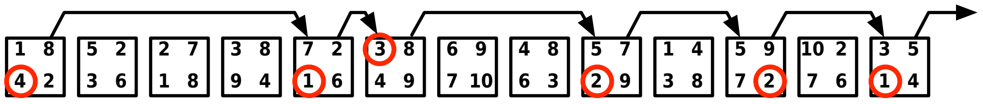


A Kenken puzzle and its solution

2. (a) Suppose $A[1..n]$ is an array of n distinct integers, sorted so that $A[1] < A[2] < \cdots < A[n]$. Each integer $A[i]$ could be positive, negative, or zero. Describe an efficient algorithm that either computes an index i such that $A[i] = i$ or correctly reports that no such index exists. An algorithm that runs in $\Theta(n)$ time is worth at most 3 points.

   (b) Now suppose $A[1..n]$ is a sorted array of n distinct ***positive*** integers. Describe an even faster algorithm that either computes an index i such that $A[i] = i$ or correctly reports that no such index exists. *[Hint: This is **really** easy!]*

3. *Moby Selene* is a solitaire game played on a row of n squares. Each square contains four positive integers. The player begins by placing a token on the leftmost square. On each move, the player chooses one of the numbers on the token's current square, and then moves the token that number of squares to the right. The game ends when the token moves past the rightmost square. The object of the game is to make as many moves as possible before the game ends.



A Moby Selene puzzle that allows six moves. (This is ***not*** the longest legal sequence of moves.)

   (a) ***Prove*** that the obvious greedy strategy (always choose the smallest number) does not give the largest possible number of moves for every Moby Selene puzzle.

   (b) Describe and analyze an efficient algorithm to find the largest possible number of legal moves for a given Moby Selene puzzle.

4. Consider the following algorithm for finding the largest element in an unsorted array:

```
RandomMax($A[1..n]$):
    $max \leftarrow \infty$
    for $i \leftarrow 1$ to $n$ in random order
        if $A[i] > max$
            $max \leftarrow A[i]$    $(\star)$
    return $max$
```

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

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

   (c) What is the ***exact*** expected number of executions of line $(\star)$? (A correct $\Theta(\,)$ bound is worth half credit.)

5. *This question is taken directly from HBS 0.* Whenever groups of pigeons gather, they instinctively establish a *pecking order*. For any pair of pigeons, one pigeon always pecks the other, driving it away from food or potential mates. The same pair of pigeons always chooses the same pecking order, even after years of separation, no matter what other pigeons are around. Surprisingly, the overall pecking order can contain cycles—for example, pigeon A pecks pigeon B, which pecks pigeon C, which pecks pigeon A.

   Prove that any finite set of pigeons can be arranged in a row from left to right so that every pigeon pecks the pigeon immediately to its left. Pretty please.

You have 90 minutes to answer four of the five questions.
Write your answers in the separate answer booklet.
You may take the question sheet with you when you leave.

1. Recall that a ***tree*** is a connected graph with no cycles. A graph is ***bipartite*** if we can color its vertices black and white, so that every edge connects a white vertex to a black vertex.

   (a) ***Prove*** that every tree is bipartite.

   (b) Describe and analyze a fast algorithm to determine whether a given graph is bipartite.

2. Describe and analyze an algorithm SHUFFLE($A[1..n]$) that randomly permutes the input array A, so that each of the $n!$ possible permutations is equally likely. You can assume the existence of a subroutine RANDOM(k) that returns a random integer chosen uniformly between 1 and k in $O(1)$ time. For full credit, your SHUFFLE algorithm should run in $O(n)$ time. *[Hint: This problem appeared in HBS 3½.]*

3. Let G be an undirected graph with weighted edges.

   (a) Describe and analyze an algorithm to compute the *maximum* weight spanning tree of G.

   (b) A ***feedback edge set*** of G is a subset F of the edges such that every cycle in G contains at least one edge in F. In other words, removing every edge in F makes G acyclic. Describe and analyze a fast algorithm to compute the minimum weight feedback edge set of G.

   [Hint: Don't reinvent the wheel!]

4. Let $G = (V, E)$ be a connected directed graph with non-negative edge weights, let s and t be vertices of G, and let H be a subgraph of G obtained by deleting some edges. Suppose we want to reinsert exactly one edge from G back into H, so that the shortest path from s to t in the resulting graph is as short as possible. Describe and analyze an algorithm to choose the best edge to reinsert. For full credit, your algorithm should run in $O(E \log V)$ time. *[Hint: This problem appeared in HBS 6¾.]*

5. Describe and analyze an efficient data structure to support the following operations on an array $X[1..n]$ as quickly as possible. Initially, $X[i] = 0$ for all i.

   - Given an index i such that $X[i] = 0$, set $X[i]$ to 1.
   - Given an index i, return $X[i]$.
   - Given an index i, return the smallest index $j \geq i$ such that $X[j] = 0$, or report that no such index exists.

   For full credit, the first two operations should run in *worst-case constant* time, and the amortized cost of the third operation should be as small as possible.

You have 180 minutes to answer six of the seven questions.
Write your answers in the separate answer booklet.
You may take the question sheet with you when you leave.

1. SUBSETSUM and PARTITION are two closely related NP-hard problems, defined as follows.

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

   PARTITION: Given a set Y of positive integers, can Y be partitioned into two subsets whose sums are equal?

   (a) **[2 pts]** ***Prove*** that PARTITION and SUBSETSUM are both in NP.

   (b) **[1 pt]** Suppose you already know that SUBSETSUM is NP-hard. Which of the following arguments could you use to prove that PARTITION is NP-hard? ***You do not need to justify your answer*** — just answer ① or ②.

   ① Given a set X and an integer k, construct a set Y in polynomial time, such that PARTITION(Y) is true if and only if SUBSETSUM(X, k) is true.

   ② Given a set Y, construct a set X and an integer k in polynomial time, such that PARTITION(Y) is true if and only if SUBSETSUM(X, k) is true.

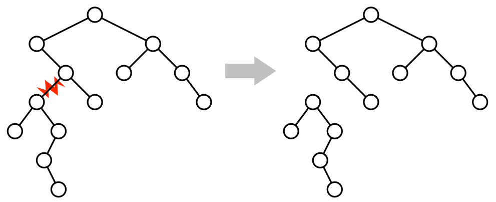
   (c) **[3 pts]** Describe and analyze a polynomial-time reduction from PARTITION to SUBSETSUM. ***You do not need to prove that your reduction is correct.***

   (d) **[4 pts]** Describe and analyze a polynomial-time reduction from SUBSETSUM to PARTITION. ***You do not need to prove that your reduction is correct.***

2. (a) **[4 pts]** For any node v in a binary tree, let $size(v)$ denote the number of nodes in the subtree rooted at v. Let k be an arbitrary positive number. ***Prove*** that every binary tree with at least k nodes contains a node v such that $k \le size(v) \le 2k$.

   (b) **[2 pts]** Removing any edge from an n-node binary tree T separates it into two smaller binary trees. An edge is called a ***balanced separator*** if both of these subtrees have at least $n/3$ nodes (and therefore at most $2n/3$ nodes). ***Prove*** that every binary tree with more than one node has a balanced separator. *[Hint: Use part (a).]*

   (c) **[4 pts]** Describe and analyze an algorithm to find a balanced separator in a given binary tree. *[Hint: Use part (a).]*


Removing a balanced separator from a binary tree.

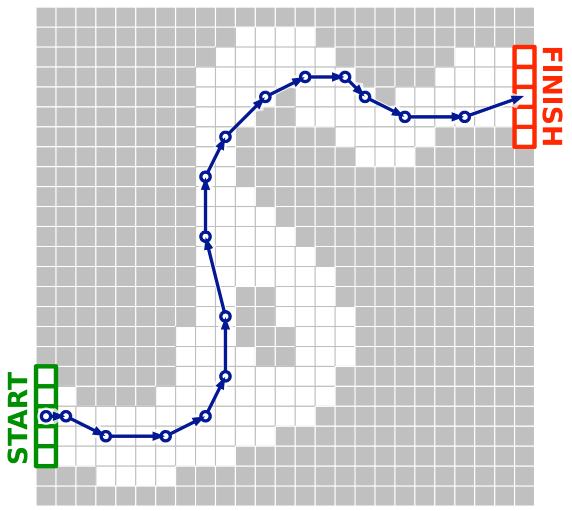
3. ***Racetrack*** (also known as *Graph Racers* and *Vector Rally*) is a two-player paper-and-pencil racing game that Jeff played on the bus in 5th grade.[1] The game is played with a track drawn on a sheet of graph paper. The players alternately choose a sequence of grid points that represent the motion 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. The initial position is a 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 be inside the track; otherwise, the car crashes and that player loses the race. The race ends when the first car reaches a position ***on*** the finish line.

   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 of a given racetrack. *[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.

4. A palindrome is any string that is exactly the same as its reversal, like `I`, or `DEED`, or `RACECAR`, or `AMANAPLANACATACANALPANAMA`. Describe and analyze an algorithm to find the length of the longest *subsequence* of a given string that is also a palindrome.

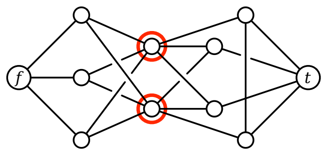
   For example, the longest palindrome subsequence of `MAHDYNAMICPROGRAMZLETMESHOWYOUTHEM` is `MHYMRORMYHM`, so given that string as input, your algorithm should output the number 11.

[1]The actual game is a bit more complicated than the version described here.

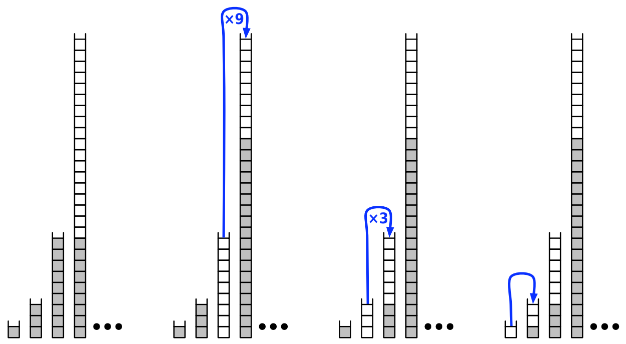
5. The Island of Sodor is home to a large number of towns and villages, connected by an extensive rail network. Recently, several cases of a deadly contagious disease (either swine flu or zombies; reports are unclear) have been reported in the village of Ffarquhar. The controller of the Sodor railway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

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

   For example, given the following input graph, your algorithm should return the number 2.



6. 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.

7. Recall the problem 3Color: Given a graph, can we color each vertex with one of 3 colors, so that every edge touches two different colors? We proved in class that 3Color is NP-hard.

   Now consider the related problem 12Color: Given a graph, can we color each vertex with one of twelve colors, so that every edge touches two different colors? ***Prove*** that 12Color 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?

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

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

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

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

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

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

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

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

DoubleHamiltonianCycle: Given a graph G, can is there a closed walk in G that visits every vertex twice?

DoubleHamiltonianPath: Given a graph G, can is there an open walk in G that visits every vertex twice?

MinDegreeSpanningTree: Given an undirected graph G, what is the minimum degree of any spanning tree of G?

MinLeavesSpanningTree: Given an undirected graph G, what is the minimum number of leaves in any spanning tree of G?

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

LongestPath: Given a graph G with weighted edges and two vertices s and t, what is the length of the longest *simple* path from s to t 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?