CS 373U: Combinatorial Algorithms, Spring 2004
Homework 1

Due Monday, February 9, 2004 at noon

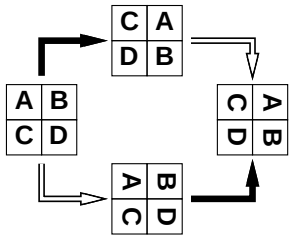
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:

- For this and all following homeworks, groups of up to three people can turn in a single solution. Please write *all* your names and NetIDs on *every* page you turn in.

#	1	2	3	4	5	6*	Total
Score							
Grader							

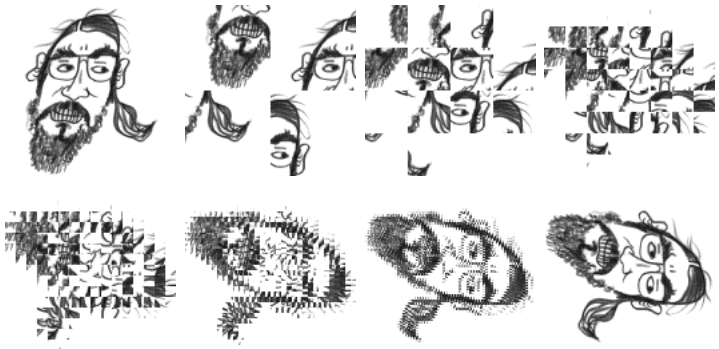
1. Some graphics hardware includes support for an operation called *blit*, or **bl**ock **t**ransfer, which quickly copies a rectangular chunk of a pixelmap (a two-dimensional array of pixel values) from one location to another. This is a two-dimensional version of the standard C library function `memcpy()`.

   Suppose we want to rotate an $n \times n$ pixelmap 90° clockwise. One way to do this is to split the pixelmap 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 can first recursively rotate the blocks and blit them into place afterwards.



Two algorithms for rotating a pixelmap.
Black arrows indicate blitting the blocks into place.
White arrows indicate recursively rotating the blocks.

   The following sequence of pictures shows the first algorithm (blit then recurse) in action.



   In the following questions, assume n is a power of two.

   (a) Prove that both versions of the algorithm are correct. *[Hint: If you exploit all the available symmetries, your proof will only be a half of a page long.]*

   (b) *Exactly* how many blits does the algorithm perform?

   (c) What is the algorithm's running time if each $k \times k$ blit takes $O(k^2)$ time?

   (d) What if each $k \times k$ blit takes only $O(k)$ time?

2. The traditional Devonian/Cornish drinking song "The Barley Mow" has the following pseudolyrics[1], where *container*$[i]$ is the name of a container that holds 2^i ounces of beer.[2]

```
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 ← 1 to n
      "We'll drink it out of the container[i], boys,"
      "Here's a health to the barley-mow!"
      for j ← 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.)

(b) If you want to sing this song for $n > 20$, you'll have to make up your own container names, and to avoid repetition, these names will get progressively longer as n increases[3]. 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 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] Pseudolyrics are to lyrics as pseudocode is to code.

[2] 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.

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

3. In each of the problems below, you are given a 'magic box' that can solve one problem quickly, and you are asked to construct an algorithm that uses the magic box to solve a different problem.

   (a) **3-Coloring**: A graph is *3-colorable* if it is possible to color each vertex red, green, or blue, so that for every edge, its two vertices have two different colors. Suppose you have a magic box that can tell you whether a given graph is 3-colorable in constant time. Describe an algorithm that constructs a 3-coloring of a given graph (if one exists) as quickly as possible.

   (b) **3SUM:** The 3SUM problem asks, given a set of integers, whether any three elements sum to zero. Suppose you have a magic box that can solve the 3SUM problem in constant time. Describe an algorithm that actually finds, given a set of integers, three elements that sum to zero (if they exist) as quickly as possible.

   (c) **Traveling Salesman:** A *Hamiltonian cycle* in a graph is a cycle that visits every vertex exactly once. Given a complete graph where every edge has a weight, the *traveling salesman* cycle is the Hamiltonian cycle with minimum total weight; that is, the sum of the weight of the edges is smaller than for any other Hamiltonian cycle. Suppose you have a magic box that can tell you the weight of the traveling salesman cycle of a weighted graph in constant time. Describe an algorithm that actually constructs the traveling salesman cycle of a given weighted graph as quickly as possible.

4. (a) Describe and analyze an algorithm to sort an array $A[1..n]$ by calling a subroutine SqrtSort(k), which sorts the subarray $A[k+1..k+\lceil\sqrt{n}\rceil]$ in place, given an arbitrary integer k between 0 and $n - \lceil\sqrt{n}\rceil$ as input. Your algorithm is *only* allowed to inspect or modify the input array by calling SqrtSort; in particular, your algorithm must not directly compare, move, or copy array elements. How many times does your algorithm call SqrtSort in the worst case?

   (b) Prove that your algorithm from part (a) is optimal up to constant factors. In other words, if $f(n)$ is the number of times your algorithm calls SqrtSort, prove that no algorithm can sort using $o(f(n))$ calls to SqrtSort.

   (c) Now suppose SqrtSort is implemented recursively, by calling your sorting algorithm from part (a). For example, at the second level of recursion, the algorithm is sorting arrays roughly of size $n^{1/4}$. What is the worst-case running time of the resulting sorting algorithm? (To simplify the analysis, assume that the array size n has the form 2^{2^k}, so that repeated square roots are always integers.)

5. In a previous incarnation, you worked as a cashier in the lost Antarctican colony of Nadira, spending the better part of your day giving change to your customers. Because paper is a very rare and valuable resource on Antarctica, cashiers were required by law to use the fewest bills possible whenever they gave change. Thanks to the numerological predilections of one of its founders, the currency of Nadira, called Dream Dollars, was available in the following denominations: \$1, \$4, \$7, \$13, \$28, \$52, \$91, \$365.[4]

   (a) The greedy change algorithm repeatedly takes the largest bill that does not exceed the target amount. For example, to make \$122 using the greedy algorithm, we first take a \$91 bill, then a \$28 bill, and finally three \$1 bills. Give an example where this greedy algorithm uses more Dream Dollar bills than the minimum possible.

   (b) Describe and analyze a recursive algorithm that computes, given an integer k, the minimum number of bills needed to make k Dream Dollars. (Don't worry about making your algorithm fast; just make sure it's correct.)

   (c) Describe a dynamic programming algorithm that computes, given an integer k, the minimum number of bills needed to make k Dream Dollars. (This one needs to be fast.)

⋆6. **[Extra Credit]** A popular puzzle called "Lights Out!", made by Tiger Electronics, has the following description. The game consists of a 5×5 array of lighted buttons. By pushing any button, you toggle (on to off, off to on) that light and its four (or fewer) immediate neighbors. The goal of the game is to have every light off at the same time.

   We generalize this puzzle to a graph problem. We are given an arbitrary graph with a lighted button at every vertex. Pushing the button at a vertex toggles its light and the lights at all of its neighbors in the graph. A *light configuration* is just a description of which lights are on and which are off. We say that a light configuration is *solvable* if it is possible to get from that configuration to the everything-off configuration by pushing buttons. Some (but clearly not all) light configurations are unsolvable.

   (a) Suppose the graph is just a cycle of length n. Give a simple and complete characterization of the solvable light configurations in this case. (What we're really looking for here is a *fast* algorithm to decide whether a given configuration is solvable or not.) *[Hint: For which cycle lengths is* **every** *configuration solvable?]*

   ⋆(b) Characterize the set of solvable light configurations when the graph is an arbitrary tree.

   ★(c) A *grid graph* is a graph whose vertices are a regular $h \times w$ grid of integer points, with edges between immediate vertical or horizontal neighbors. Characterize the set of solvable light configurations for an arbitrary grid graph. (For example, the original Lights Out puzzle can be modeled as a 5×5 grid graph.)

[4] For more details on the history and culture of Nadira, including images of the various denominations of Dream Dollars, see http://www.dream-dollars.com.