CS 373: Combinatorial Algorithms, Spring 1999

http://www-courses.cs.uiuc.edu/~cs373

Homework 1 (due February 9, 1999 by noon)

Name:	
Net ID:	Alias:

Everyone must do the problems marked ▶. Problems marked ▷ are for 1-unit grad students and others who want extra credit. (There's no such thing as "partial extra credit"!) Unmarked problems are extra practice problems for your benefit, which will not be graded. Think of them as potential exam questions.

Hard problems are marked with a star; the bigger the star, the harder the problem.

Note: When a question asks you to "give/describe/present an algorithm", you need to do four things to receive full credit:

1. Design the most efficient algorithm possible. Significant partial credit will be given for less efficient algorithms, as long as they are still correct, well-presented, and correctly analyzed.
2. Describe your algorithm succinctly, using structured English/pseudocode. We don't want full-fledged compilable source code, but plain English exposition is usually not enough. Follow the examples given in the textbooks, lectures, homeworks, and handouts.
3. Justify the correctness of your algorithm, including termination if that is not obvious.
4. Analyze the time and space complexity of your algorithm.

Undergrad/.75U Grad/1U Grad Problems

▶1. Consider the following sorting algorithm:

```
StupidSort($A[0\,..\,n-1]$) :
    if $n = 2$ and $A[0] > A[1]$
        swap $A[0] \leftrightarrow A[1]$
    else if $n > 2$
        $m = \lceil 2n/3 \rceil$
        StupidSort($A[0\,..\,m-1]$)
        StupidSort($A[n-m\,..\,n-1]$)
        StupidSort($A[0\,..\,m-1]$)
```

(a) Prove that StupidSort actually sorts its input.

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

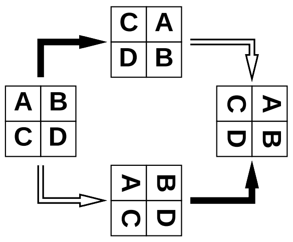
(c) State a recurrence (including the base case(s)) for the number of comparisons executed by StupidSort.

(d) Solve the recurrence, and prove that your solution is correct. [Hint: Ignore the ceiling.] Does the algorithm deserve its name?

*(e) Show that the number of *swaps* executed by StupidSort is at most $\binom{n}{2}$.

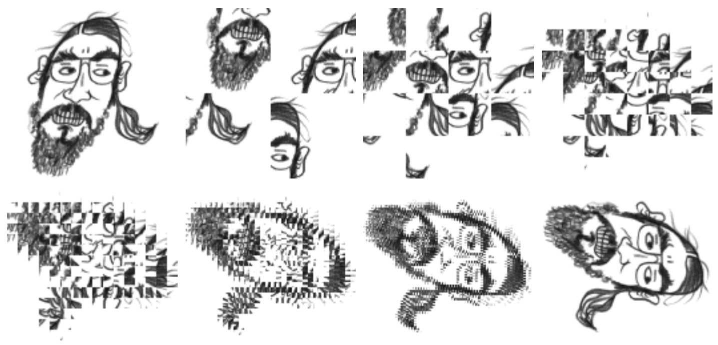
►2. Some graphics hardware includes support for an operation called *blit*, or **bl**ock **t**ransfer, which quickly copies a rectangular chunk of a pixelmap (a two-dimensional array of pixel values) from one location to another. This is a two-dimensional version of the standard C library function `memcpy()`.

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.

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

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

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

►3. Dynamic Programming: The Company Party
A company is planning a party for its employees. The organizers of the party want it to be a fun party, and so have assigned a 'fun' rating to every employee. The employees are organized into a strict hierarchy, i.e. a tree rooted at the president. There is one restriction, though, on the guest list to the party: both an employee and their immediate supervisor (parent in the tree) cannot both attend the party (because that would be no fun at all). Give an algorithm that makes a guest list for the party that maximizes the sum of the 'fun' ratings of the guests.

►4. Dynamic Programming: Longest Increasing Subsequence (LIS)
Give an $O(n^2)$ algorithm to find the longest increasing subsequence of a sequence of numbers. Note: the elements of the subsequence need not be adjacent in the sequence. For example, the sequence $(1, 5, 3, 2, 4)$ has an LIS $(1, 3, 4)$.

►5. Nut/Bolt Median
You are given a set of n nuts and n bolts of different sizes. Each nut matches exactly one bolt (and vice versa, of course). The sizes of the nuts and bolts are so similar that you cannot compare two nuts or two bolts to see which is larger. You can, however, check whether a nut is too small, too large, or just right for a bolt (and vice versa, of course).

In this problem, your goal is to find the median bolt (*i.e.*, the $\lfloor n/2 \rfloor$th largest) as quickly as possible.

(a) Describe an efficient deterministic algorithm that finds the median bolt. How many nut-bolt comparisons does your algorithm perform in the worst case?

(b) Describe an efficient *randomized* algorithm that finds the median bolt.

i. State a recurrence for the expected number of nut/bolt comparisons your algorithm performs.

ii. What is the probability that your algorithm compares the ith largest bolt with the jth largest nut?

iii. What is the expected number of nut-bolt comparisons made by your algorithm? [Hint: Use your answer to either of the previous two questions.]

Only 1U Grad Problems

▷1. You are at a political convention with n delegates. Each delegate is a member of exactly one political party. It is impossible to tell which political party a delegate belongs to. However, you can check whether any two delegates are in the *same* party or not by introducing them to each other. (Members of the same party always greet each other with smiles and friendly handshakes; members of different parties always greet each other with angry stares and insults.)

(a) Suppose a majority (more than half) of the delegates are from the same political party. Give an efficient algorithm that identifies a member of the majority party.

*(b) Suppose exactly k political parties are represented at the convention and one party has a *plurality*: more delegates belong to that party than to any other. Give an efficient algorithm that identifies a member of the plurality party.

*(c) Suppose you don't know how many parties there are, but you do know that one party has a plurality, and at least p people in the plurality party are present. Present a practical procedure to pick a person from the plurality party as parsimoniously as possible. (Please.)

$^{\bigstar}$(d) Finally, suppose you don't know how many parties are represented at the convention, and you don't know how big the plurality is. Give an efficient algorithm to identify a member of the plurality party. How is the running time of your algorithm affected by the number of parties (k)? By the size of the plurality (p)?

Practice Problems

1. Second Smallest
   Give an algorithm that finds the *second* smallest of n elements in at most $n + \lceil \lg n \rceil - 2$ comparisons. Hint: divide and conquer to find the smallest; where is the second smallest?

2. Linear in-place 0-1 sorting
   Suppose that you have an array of records whose keys to be sorted consist only of 0's and 1's. Give a simple, linear-time $O(n)$ algorithm to sort the array in place (using storage of no more than constant size in addition to that of the array).

3. Dynamic Programming: Coin Changing
   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.

   (e) Prove that, with only two coins a, b whose gcd is 1, the smallest value n for which change *can* be given for all values greater than or equal to n is $(a-1)(b-1)$.

   $^{\bigstar}$(f) For only three coins a, b, c whose gcd is 1, give an algorithm to determine the smallest value n for which change *can* be given for all values greater than n. (note: this problem is currently unsolved for $n > 4$.

4. Dynamic Programming: Paragraph Justification
   Consider the problem of printing a paragraph neatly on a printer (with fixed width font). The input text is a sequence of n words of lengths $l_1, l_2, \ldots, l_n$. The line length is M (the maximum # of characters per line). We expect that the paragraph is left justified, that all first words on a line start at the leftmost position and that there is exactly one space between any two words on the same line. We want the uneven right ends of all the lines to be together as 'neat' as possible. Our criterion of neatness is that we wish to minimize the sum, over all lines except the last, of the cubes of the numbers of extra space characters at the ends of the lines. Note: if a printed line contains words i through j, then the number of spaces at the end of the line is $M - j + i - \sum_{k=i}^{j} l_k$.

   (a) Give a dynamic programming algorithm to do this.

   (b) Prove that if the neatness function is linear, a linear time greedy algorithm will give an optimum 'neatness'.

5. Comparison of Amortized Analysis Methods
   A sequence of n operations is performed on a data structure. The ith operation costs i if i is an exact power of 2, and 1 otherwise. That is operation i costs $f(i)$, where:

$$f(i) = \begin{cases} i, & i = 2^k, \\ 1, & \text{otherwise} \end{cases}$$

   Determine the amortized cost per operation using the following methods of analysis:

   (a) Aggregate method

   (b) Accounting method

   *(c) Potential method