CS 373: Combinatorial Algorithms, Summer IMCS 2000

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

Homework 1 (due June 6, 2000)

Submit solutions by 1700 GMT (12noon Central standard time) by **attaching** a **postscript** file to an email sent to `maharri@cs.uiuc.edu` with the subject `cs373hw submit`. You will then get an automatic email acknowledgment.

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 within the specifications given. 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.

---

1. (6 pts, 2 each) Consider the following sorting algorithm:

```
StupidSort(A[0 .. n − 1]) :
    if n = 2 and A[0] > A[1]
        swap A[0] ↔ A[1]
    else if n > 2
        m = ⌈2n/3⌉
        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 and solve a recurrence (including the base case(s)) for the number of comparisons executed by StupidSort. [Hint: Ignore the ceiling.] Does the algorithm deserve its name?

2. (5 pts) Arbitrary Selection (using Median)
   Suppose you have a subroutine that can find the median (the middlemost element, the element at ordered position $\lfloor n/2 \rfloor$) in $O(n)$ time. Give an algorithm to find the kth biggest element (for arbitrary k) in $O(n)$ time.

3. (6 pts, 2 each) Dynamic Programming: Knapsack
   You're walking along the beach and you stub your toe on something in the sand. You dig around it and find that it is a treasure chest full of gold bricks of different (integral) weight. Your knapsack can only carry up to weight n before it breaks apart. You want to put as much in it as possible without going over, but you *cannot* break the gold bricks up.

   (a) Suppose that the gold bricks have the weights $1, 2, 4, 8, \ldots, 2^k$, $k \geq 1$. Describe and prove correct a greedy algorithm that fills the knapsack as much as possible without going over.

   (b) Give a set of 3 weight values for which the greedy algorithm does not yield an optimal solution and show why.

   (c) Give a dynamic programming algorithm that yields an optimal solution for an arbitrary set of gold brick values.

4. Dynamic Programming: (6 pts) 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: an employee and their immediate supervisor (parent in the tree) cannot both attend the party (because that would be no fun at all!).

   (a) (4 pts) Give an algorithm that makes a guest list for the party that *maximizes* the sum of the 'fun' ratings of the guests.

   (b) (2 pts) What would you do to make sure that the president attends the party. Justify.

5. Amortized cost of binary heaps (5 pts)
   You are given a binary heap with n elements that supports INSERT and EXTRACT-MIN in $O(\log n)$ worst-case time. Give a potential function Φ such that the amortized cost of INSERT is $O(\log n)$ and that of EXTRACT-MIN is $O(1)$. Justify.