CSE 373: Analysis of Algorithms

Assignment 1 (Sept. 27th 2016)

Due date and time: Oct. 5th 17:00p.m.
Submit in class (handwritten hardcopy).

You can work in pairs. If so, turn in one copy and write both participants.

Following are problems taken from the Book:

3-3. We have seen how dynamic arrays enable arrays to grow while still achieving constant-time amortized performance. This problem concerns extending dynamic arrays to let them both grow and shrink on demand.

1. Consider an underflow strategy that cuts the array size in half whenever the array falls below half full. Give an example sequence of insertions and deletions where this strategy gives a bad amortized cost.
2. Then, give a better underflow strategy than that suggested above, one that achieves constant amortized cost per deletion.

* Amortized analysis: Given n sequence of operations what is the estimated cost of each operations? There are several types of amortized analysis methods, but the simplest is taking the average cost over n sequence of operations. EX> n sequence of PUSH and POP operations in a STACK takes T(n) time then amortized analysis of both PUSH and POP will be T(n)/n.

3-6. Describe how to modify any balanced tree data structure such that search, insert, delete, minimum, and maximum still take $O(\log n)$ time each, but successor and predecessor now take $O(1)$ time each. Which operations have to be modified to support this?

3-9. A *concatenate* operation takes two sets $S1$ and $S2$, where every key in $S1$ is smaller than any key in $S2$, and merges them together. Give an algorithm to concatenate two binary search trees into one binary search tree. The worst-case running time should be $O(h)$, where h is the maximal height of the two trees.

3-11. Suppose that we are given a sequence of n values $x1,x2,...,xn$ and seek to quickly answer repeated queries of the form: given i and j, find the smallest value in $xi,\ldots,xj$.

1. Design a data structure that uses $O(n)$ space and answers queries in $O(\log n)$ time. For partial credit, your data structure can use $O(n\log n)$ space and have $O(\log n)$ query time.

3-13. Let $A[1..n]$ be an array of real numbers. Design an algorithm to perform any sequence of the following operations:

1. *Add(i,y)* -- Add the value y to the ith number.
2. *Partial-sum(i)* -- Return the sum of the first i numbers, i.e. $\sum_{j=1}^{i} A[j]$.

There are no insertions or deletions; the only change is to the values of the numbers. Each operation should take $O(\log n)$ steps. You may use one additional array of size n as a work space

4-6. Given two sets $S1$ and $S2$ (each of size n), and a number x, describe an $O(n\log n)$ algorithm for finding whether there exists a pair of elements, one from $S1$ and one from $S2$, that add up to x. (For partial credit, give a $\Theta(n^2)$ algorithm for this problem.

4-7. Outline a reasonable method of solving the following problem. Give the order of the worst-case complexity of your methods.

Problem: You are given a pile of thousands of telephone bills and thousands of checks sent in to pay the bills. Find out who did not pay.

4-8. Given a set of S containing n real numbers, and a real number x. We seek an algorithm to determine whether two elements of S exist whose sum is exactly x. Assume that S is unsorted. Give an $O(n\log n)$ algorithm for the problem.

4-14. Give an $O(n\log k)$-time algorithm that merges k sorted lists with a total of n elements into one sorted list. (Hint: use a heap to speed up the elementary $O(kn)$-time algorithm).

4-16. Use the partitioning idea of quicksort to give an algorithm that finds the *median* element of an array of n integers in expected $O(n)$ time. (Hint: must you look at both sides of the partition?)

4-17. Suppose quicksort were always to pivot on the $\lceil n/3 \rceil$th smallest value of the current sub-array. How many comparisons would be made then in the worst case?

4-20. Give an efficient algorithm to rearrange an array of n keys so that all the negative keys precede all the nonnegative keys. Your algorithm must be in-place, meaning you cannot allocate another array to temporarily hold the items. How fast is your algorithm?

4-31. Suppose you are given an array A of n sorted numbers that has been *circularly shifted* k positions to the right. For example, {35,42,5,15,27,29} is a sorted array that has been circularly shifted k=2positions, while {27,29,35,42,5,15} has been shifted k=4 positions.

1. Suppose you know what k is. Give an $O(1)$ algorithm to find the largest number in A.
2. Suppose you *do not* know what k is. Give an $O(\lg n)$ algorithm to find the largest number in A. For partial credit, you may give an $O(n)$ algorithm.

4-42. Describe an algorithm that takes an input array and returns only the unique elements in it.

2-35. Consider the following code fragment.

```
for i=1 to n do
  for j=i to 2*i do
    output foobar
```

Let $T(n)$ denote the number of times `foobar' is printed as a function of n.

1. Express $T(n)$ as a summation (actually two nested summations).
2. Simplify the summation. Show your work