CSci 4041: Algorithms and Data Structures (Spring'15)

Homework 2, Due 02/19/15 (in class)

Answer all of the following questions, and always explain your answer. You are expected to do the work on your own.

Any proofs you do have to be rigorous, covering all cases, to get credit. Just giving an example does not constitute a correct proof, and will not receive credit. For complexity analysis, just stating the complexity is not sufficient. Clearly explain how you get the complexity.

The homework is due in class on Feb 19. We want a paper copy submission.

Good Luck!

1. (20 points) Suppose we want to sample a random subset of the set $\{1, 2, 3, \ldots, n\}$, that is, an m-element subset S, where $0 \leq m \leq n$, such that each m-subset is equally likely to be created. Since there are $\binom{n}{m}$ such subsets, the probability of picking any particular subset should be $1/\binom{n}{m}$.

   (a) (10 points) Given a budget of n calls to the RANDOM function, describe an algorithm which will output a random m-element subset S for $1 \leq m \leq n$. Prove that the algorithm outputs any specific m-element subset with probability $1/\binom{n}{m}$.

   (b) (10 points) Given a budget of only m calls to the RANDOM function, prove that the following recursive procedure returns a random m-subset S of $\{1, 2, 3, \ldots, n\}$ in which each m-subset is equally likely, while making only m calls to RANDOM:

```
QUICK-SAMPLE($m, n$)
1  if $m == 0$
2      return $\emptyset$
3  else $S$ = QUICK-SAMPLE($m-1, n-1$)
4      $i$ = RANDOM($1, n$)
5      if $i \in S$
6          $S = S \cup \{n\}$
7      else $S = S \cup \{i\}$
8      return $S$
```

2. (30 points) A d-ary heap is like a binary heap, but (with one possible exception) non-leaf nodes have d children instead of 2 children.

   (a) (5 points) How would you represent a d-ary heap in an array?

   (b) (5 points) What is the height of a d-ary heap of n elements in terms of n and d?

(c) (10 points) Give an efficient implementation of EXTRACT-MAX in a d-ary max-heap. You have to give details for any sub-routines used. Analyze the running time of the algorithm in terms of d and n.

(d) (10 points) Give an efficient implementation of INCREASE-KEY(A, i, k), which flags an error if $k < A[i]$, but otherwise sets $A[i] = k$ and then updates the d-ary max-heap structure appropriately. Analyze its running time in terms of d and n.

3. (20 points) Consider the problem of sorting using a max-heap. Recall that the HEAPSORT algorithm extracts the maximum element in each iteration, and maintains the max-heap property by calling MAX-HEAPIFY. For the problem, we consider a similar strategy, but by extracting the minimum element in each iteration instead of the maximum.

   (a) (10 points) Describe an algorithm for MAX-HEAP-EXTRACT-MIN(A) which outputs the minimum element in the heap A, and maintains the heap property after extraction. The description should include pseudo-code and a proof of correctness of the algorithm. Analyze the running time of the algorithm for a heap of size n.

   (b) (10 points) Describe an algorithm MY-HEAP-SORT(A) which uses MAX-HEAP-EXTRACT-MIN as a subroutine to sort an array. The description should include pseudo-code and a proof of correctness of the algorithm. Analyze the running time of the algorithm for an array of size n.

4. (30 points) Consider the following partitioning algorithm for QUICKSORT, which is different from what was discussed in class:

```
OTHER-PARTITION($A, p, r$)
1  $x = A[p]$
2  $i = p - 1$
3  $j = r + 1$
4  while TRUE
5      repeat
6          $j = j - 1$
7      until $A[j] \le x$
8      repeat
9          $i = i + 1$
10     until $A[i] \ge x$
11     if $i < j$
12         exchange $A[i]$ with $A[j]$
13     else return $j$
```

   (a) (5 points) Demonstrate the operation of OTHER-PARTITION on the array $A = \langle 13, 19, 9, 5, 12, 8, 7, 4, 11, 2, 6, 21 \rangle$, showing the values of the array and auxiliary values after each iteration of the **while** loop in lines 4-13.

The next three questions ask you to give a careful argument that the procedure OTHER-PARTITION is correct. Assuming that the subarray $A[p \cdots r]$ contains at least two elements, prove the following:

(b) (5 points) The indices i and j are such that we never access an element of A outside the subarray $A[p \cdots r]$.

(c) (5 points) When OTHER-PARTITION terminates, it returns a value j such that $p \leq j < r$.

(d) (10 points) When OTHER-PARTITION terminates, every element of $A[p \cdots j]$ is less than or equal to every element of $A[j + 1 \cdots r]$.

The PARTITION procedure for QUICKSORT (Section 7.1 in the book) separates the pivot value (originally in $A[r]$) from the two partitions it forms. The OTHER-PARTITION procedure, on the other hand, always places the pivot value (originally in $A[p]$) into one of the two partitions $A[p \cdots j]$ and $A[j + 1 \cdots r]$. Since $p \leq j < r$, this split is always nontrivial.

(e) (5 points) Rewrite the QUICKSORT procedure to use OTHER-PARTITION.