ESO207A: Data Structures and Algorithms

Homework *2: Heaps, Stacks* HW Due Date: Sep 7, 2018

Instructions.

1. Start each problem on a new sheet. For each problem, write your name, Roll No., the problem number, the date and the names of any students with whom you collaborated. Remember that you must write the answer and the algorithm in your own words.
2. For questions in which algorithms are asked for, first summarize the problem you are solving and your results (including time/space complexity as appropriate). The body of the write-up should provide the following:
   (a) A clear description of the algorithm in English and/or pseudo-code, where, helpful.
   (b) At least one worked example or diagram to show more precisely how your algorithm works.
   (c) A proof/argument of the correctness of the algorithm.
   (d) An analysis of the running time of the algorithm.

   Remember, your goal is to communicate. *Full marks will be given only to correct solutions which are described clearly.* Convoluted and unclear descriptions will receive *low marks.*

Problem 1. *Heap.* The following is an alternative routine to build a max-heap by permuting the elements of an array. It uses the subroutine *Max-Heap-Insert*$(A, heapsize, v)$. This routine takes $A[1, \ldots, heapsize]$ and inserts v into the max-heap. It also increments *heapsize* by 1 (*heapsize* is a reference parameter).

```
procedure Build-Max-Heap1(A, n) // Build Max Heap for the elements in A[1, ..., n]
1.    heapsize = 1
2.    for i = 2 to n
3.          Max-Heap-Insert(A, heapsize, A[i])
```

1. Show that this procedure takes $O(n \log n)$ time. (5)
2. Show a worst case input and analyze it to show that on that input the procedure takes $\Omega(n \log n)$ time. Hence this algorithm takes $\Theta(n \log n)$ time worst-case. (20)

Problem 2. *CLRS 6-3 Young Tableau.* $(10 \times 4 = 40)$

An $m \times n$ Young tableau is an $m \times n$ matrix such that the entries of each row are in sorted order from left to right and the entries of each column are in sorted order from top to bottom. Some of the entries may be ∞ that are treated as non-existent items. Thus a Young tableau can be used to hold $r \leq mn$ finite numbers. Note that an $m \times n$ tableau Y is empty if $Y[1,1] = \infty$ and is full if $Y[m,n] < \infty$. An example Young tableau is $\begin{bmatrix} 2 & 4 & 8 & 12 \\ 3 & 5 & 9 & \infty \\ 12 & 14 & 16 & \infty \end{bmatrix}$

1. Give an algorithm for EXTRACT-MIN on a non-empty $m \times n$ Young tableau that runs in time $O(m+n)$. (*Hint*: Follow Min-Heapify.)

2. Show how to insert a new element into a non-full $m \times n$ Young tableau in $O(m+n)$ time.

3. Using no other sorting method as a subroutine, show how to use an $n \times n$ Young tableau to sort n^2 numbers in $O(n^3)$ time.

4. Given an $O(m+n)$-time algorithm to determine whether a given number is stored in a given $m \times n$ tableau.

Problem 3. *Stack: a span problem.* You are given an array $P[1, \ldots, n]$ giving the daily price quotes for a stock for n consecutive days. The span of the stock's price on a given day i is the maximum number of consecutive days that the stock's price is less than or equal to its price on day i. (This includes day i, so span is at least 1). For example, suppose P is the array $\{50, 45, 35, 40, 60, 50, 55\}$. Then, the span array for these 7 consecutive days is $\{1, 1, 1, 2, 5, 1, 2\}$. An $O(n^2)$ algorithm follows in a straightforward way from the definition. Design an $O(n)$ (linear-time) algorithm and provide an analysis. (30+5)

(*Hint*: For $i = 1, 2, \ldots, n$, let $h(i)$ be the highest index less than or equal to i such $P[h(i)] > P[i]$, if such an index exists, otherwise, it is 0. The span for index i is $i - h(i)$. Store $i, h(i), h(h(i)), \ldots$ downwards in the stack. Note that this constitutes the unique ascending sequence $P[i] < P[h(i)] < P[h(h(i))] < \ldots$. The economy of computation is obtained as follows. Given $P[i+1]$, if $P[i] > P[i+1]$, then, $h(i+1) = i$, but if $P[i] \leq P[i+1]$, then, we need only compare $P[i+1]$ with $P[h(i)]$, since, all elements j such that $h(i) < j \leq i$ have $P[j] \leq P[i]$. So values at indices $i-1$ down to $h(i)+1$ need not be stored or compared with, as they are clearly no larger than $P[i]$. So on, if $P[h(i)] \leq P[i+1]$, then, consider $P[h(h(i))]$, and so on.)