CSCI4041 Homework1 Solution Key

Spring 2009

1 [20pt]

[10pt] Pseudocode

```
   SORT(A,n) // A: array, n: size of array
1     for i <- 1 to n-1
2           // find small lest
3           smallest <- i
4           for j <- i+1 to n
5                 if(A[j] < A[smallest]) then
6                       smallest <- j
7                 end
8           end // j loop
9           // swap i and smallest
10          Temp <- A[i]
11          A[i] <- A[smallest]
12          A[smallest] <- Temp
13    end // i loop
14 end // SORT
```

(-1) for each incorrect loop start/end index
(-1) for each mistakes of keeping smallest index in i-loop

[4pt] Loop invariant

For each iteration of I-loop (line 1-13), every element of index < i is sorted in increasing order (2pt), and every element of index ≥ i is greater than every element of index < i (2pt).

[2pt] According to the loop invariant, left n-1 elements are sorted, and n-th element is greater than its n-1 left elements in n-th iteration. So n-th iteration is not needed.

[2pt] Best case: $\Theta(n^2)$

[2pt] Worst case: $\Theta(n^2)$

2. [20pt]

[10] Pseudocode

```
FIND_MIN_MAX(A,s,t) // A: array, i:start index, t: last index,
                        returns array with 2 elements {min, max}
if    s=t then
      return {A[s], A[s]}
end
mid <- s + floor( (t-s+1)/2)
// divide array into 2 subarrays each of size n/2
{min1, max1} <- FIND_MIN_MAX(A,s,mid)
{min1, max2} <- FIND_MIN_MAX(A, mid+1,t)
// merge step
if min1<min2 then min = min1
else min = min2
if max1>max2 then max = max1
else max = max2
return {min, max}
```

(-1) for incorrect selecting mid
(-3) for no base case
(-3) for incorrect divide step
(-3) for incorrect merge step

[8pt] Recurrence Relation
[4pt] $T(n) = \Theta(1)$, if $n=\Theta(1)$
$2T(n/2) + \Theta(1)$, otherwise
[4pt] By master's theorem, $T(n)= \Theta(n)$ (case 1)

[2pt] It runs with as much time as linear scan

3. [20pt]
[10pt] Insertion sort is stable since it swaps A[i] and A[j] ($i<j$) only when $A[i]>A[j]$.
[10pt] Mergesort is stable since merge function takes the smallest element from the left subarray first.
(5pt for each answer and reason, -10pt if heapsort is included in the answer)

4. [20pt]
Sort the array with any sorting algorithm runs in $\Theta(n\lg n)$ – heapsort, mergesort
for each i, $1 \le i \le n$, find x-A[i] in sorted array A using binary search.
(5pt) for sort with $\Theta(n\lg n)$ sorting
(5pt) for find x-A[i] for each element of A
(10pt) for using binary search

5. [20pt]
Best case scenario is when the input is already a heap.
(5pt) Build-max-heap runs at $\Theta(n)$.
(5pt) For loop iterations runs $\Theta(n)$ times.
(10pt) For each step of the loop, root is replaced by a leaf and run max-heapify. Since this replace root should be moved from root to the leaf, max-heapify takes $\Theta(\lg n)$ each.
Therefore, best-case running time for heapsort is $\Omega(n\lg n)$
(-5pt for saying max-heapify takes constant time or max-heapify 'always' takes $\Theta(\lg n)$ regardless of the input)