CS2351 Data Structures

Homework 1 (Suggested Solution)

There are many different ways to answer the questions perfectly. Don't worry if your answer is different with ours. In general, what we want to have is *precise and concise*.

1. **Ans:** Our algorithm is similar to binary search. We test the middle entry of A, say $A[x]$, and compare with the element $A[x+1]$ on its right. If $A[x+1]$ is smaller, we discard all entries in $A[x+1..n]$, and will recursively search for the maximum entry in $A[1..x]$. Otherwise, we discard all entries in $A[1..x]$, and will recursively search for the maximum entry in $A[x+1..n]$.

   After each step of the above algorithm, the problem size is reduced by half, so that it runs in $O(\log n)$ time. The correctness follows from the fact that the entries we discarded in each step cannot hold the maximum entry.

2. **Ans:**

   (a) For simplicity, we first assume that there are swaps in each round.

   After the first round, the largest element will occupy the last entry $A[n]$, and will never be moved in the future rounds. In general, we can easily show by induction that after the first k rounds, the largest j elements will occupy the last j entries, sorted, and will never be moved in the future rounds. So after $n-1$ rounds, the largest $n-1$ items will occupy the last $n-1$ entries, sorted, which implies that the whole array becomes sorted.

   Now, consider in general that at some round j, no swaps occur. This immediately implies that all the first $n-j+1$ entries are already sorted. Together with the induction hypothesis that the last $j-1$ entries contain the largest $j-1$ items and are sorted, this implies the whole array is sorted, so that we can skip the processing in the remaining rounds.

   (b) Each round runs through the array entries at most once, so that each round runs in $O(n)$ time. As there are $O(n)$ rounds, the running time is $O(n^2)$.

   (c) When the input array is reversely sorted, so that $A[1] > A[2] > \cdots > A[n]$, round j will require exactly $n-j$ swaps. In total, the number of swaps is equal to $(n-1)+(n-2)+\cdots+2+1=\Theta(n^2)$. The worst case can only be worse, so that the worst case running time is $\Omega(n^2)$.

   (d) In the best case, we go through the array entries once, find out no swaps occur, and skip the remaining rounds. The running time will then be only $O(n)$, which is not $\Omega(n^2)$. Thus, we cannot say the running time for Bubble Sort is $\Theta(n^2)$, as it is not true that "with all input, Bubble Sort's running time will be $\Theta(n^2)$".

3. Our algorithm below first finds the starting location of a desired portion, and after that, finds the corresponding ending location of the desired portion. Briefly speaking, we test iteratively whether $B[1]$, $B[2]$, and so on can be the correct starting location, and remove them from consideration once we know that they are wrong.

   Description of algorithm: We iteratively to compute the sum of $B[1]+B[2]+\cdots+B[i]$ until it is at least Y. If the sum is exactly Y, we have obtained a desired portion of B, and stop. Else, we iteratively remove $B[1]$, $B[2]$, and so on from the sum, until the sum is at most Y. Note that whenever we remove an entry $B[k]$ from the sum, it is guaranteed that

$$B[k] + \cdots + B[i] > Y,$$

while

$$B[k] + \cdots + B[i-1] < B[1] + B[2] + \cdots + B[i-1] < Y,$$

so that the desired portion (if it exists) cannot start at $B[k]$.

Again, if the sum is now exactly Y, we have obtained a desired portion of B, and stop. Otherwise, we continue the search by now adding more entries at the back (those after $B[i]$) to the current sum until the sum is at least Y, and then (if needed) removing entries at the front from the current sum until the sum is at most Y. This adding and removing processes are repeated until we find a portion whose sum is exactly Y.

Analysis of algorithm: The running time is $O(n)$, as each entry is added to and removed from the sum at most once, so that there are only $2n$ different sums to check. The correctness follows because whenever we remove an entry, such an entry cannot be the starting location of a desired portion.