United International University
Department of Computer Science and Engineering
CSI 227: Algorithms, Mid Exam, Spring 2017
Total Marks: 90, Time: 1 hour 45 minutes

Answer any 6 out of 8 questions ($6 \times 15 = 90$).

1. (a) Perform worst case runtime analysis of the following C functions and express in Θ notation. [$2 \times 5 = 10$]

```
function1(A, B)
    s = 0
    n = A.length
    m = B.length
    for (i = 1; i <= n; i = i+1):
        s = s + A[i]
    for i = 1; i <= m; i = i+1):
        s = s + B[i]
    print s
```

```
function2(n):
    sum = 0
    for (i = 1; i <= n; i = i*2):
        sum = sum + i
    print sum
```

(b) Which of the sorting algorithms - *Merge-Sort* and *Heapsort* is better in terms of extra space usage besides the provided input array? Explain very briefly. [2]

(c) Write down the recursion for the length of a Longest Common Subsequence (LCS) of the two strings $A = \{a_1, a_2, \ldots, a_m\}$ and $B = \{b_1, b_2, \ldots, b_n\}$. Modify it to obtain a new recursion of length for the LCS of 3 strings $A = \{a_1, a_2, \ldots, a_m\}$, $B = \{b_1, b_2, \ldots, b_n\}$ and $C = \{c_1, c_2, \ldots, c_p\}$. [3]

2. (a) Using *recursion tree* method find out a good asymptotic upper bound on the following recurrence: $T(n) = 3T(n/2) + n$. [12]

(b) In a *ternary heap* H where the children of H[i] are H[3*i+1], H[3*i+2] and H[3*i+3] and the parent of H[i] is H[i/3], what would be the runtime complexity of HEAPIFY(H,i,heapsize)? [3]

3. (a) Given below is the pseudocode of Bubble-Sort algorithm to sort an input array in non-decreasing order. Perform runtime analysis to find the cost function in *best case* scenario and express the function in Big-Oh notation. [10]

```
Bubble-Sort(A):
    n = A.length
    while n>0:
        swapped = false
        for i = 2 to n:
            if A[i-1] > A[i]:
                swap(A[i-1], A[i])
                swapped = true
        n = n-1
        if swapped == false:
            break
```

(b) Suppose that for a problem X with size n, a divide-and-conquer algorithm solves it as follows: If the problem size is less than or equal to c, then it solves it at constant time. Otherwise, it divides the problem into b subproblems, each with a size of n/d. This division takes time $\theta(\log n)$. Then the algorithm solves the b number of subproblems recursively, and then combines their solution at time $\theta(n \log n)$. Design a recurrence relation for the running-time $T(n)$ of this algorithm. [3]

(c) Both the algorithmic paradigms: Divide-and-Conquer and Dynamic Programming solve a problem by breaking it into smaller problem instances, and by solving them. What is the fundamental difference between these two paradigms? [2]

4. (a) "In order for *radix sort* to work correctly, the digit sorts (sorting values by some digit i) must be stable" - justify this claim by providing appropriate examples. [5]

(b) Is it always possible to sort an array of positive integers belonging to any possible range using *counting sort*? Why does the $\Omega(nlgn)$ lower bound not apply to *counting sort*? [2+3]

(c) What does the *Find-Maximum-Subarray* algorithm return when all the array elements are negative? [2]

(d) AlgoCoins are used by the people of AlgoLand to make daily purchases. You are given an abundant supply of 23, 16, 9, and 1 AlgoCoins. You need to calculate the minimum number of AlgoCoins needed to make up a specific amount. Give an example where the greedy approach does not provide an optimal solution. [3]

5. (a) Show that $5n^2 + 2n + 1 = \Theta(n^2)$ [5]

(b) Describe an algorithm that returns the second maximum element from a *Max-Priority Queue* data structure in time $O(\log n)$ or better. [**Hint**: How about **temporarily** deleting the maximum element from the priority queue?] [7.5]

(c) Why is the worst case running-time of the *Max-Heapify* algorithm $O(\log n)$? [2.5]

6. (a) Demonstrate the simulation of the *Find-Minimum-Subarray* algorithm on the following sequence of numbers: $< -2, 3, -2, 4, 6, -5, 2, -4 >$. Clearly demonstrate the recursion-tree of the execution of the algorithm. Besides each node of the tree, mention the following four values: solutions from the left and right children, minimum crossing sum, and the minimum subarray sum that is to be returned to the parent node. **Note that**, you are required to execute the *Find-Minimum-Subarray* algorithm, which is very much similar to the *Find-Maximum-Subarray* algorithm. [12]

(b) While removing the maximum priority element from a *Max-Priority Queue* data structure, we replace the root (index 1) of the corresponding heap with the last element of the heap, decrement the heapsize by 1, and then execute the *Max-Heapify* algorithm on index 1. Why do we choose the last element instead of any other element? [3]

7. (a) Design a max-heap with height 2 and having the maximum possible number of elements. All the node values need to be positive. Then replace the value at the root with -1, and execute the *Max-Heapify* algorithm on this node. Clearly demonstrate each step. [2+3]

(b) Use a dynamic programming algorithm to find the length of the LCS of the following strings: X = $\{a, b, c, d, d, c, b\}$, Y = $\{a, c, b, d, c, b\}$. You must show all the steps of your simulation. [10]

8. (a) You are given a rod of length n which you must cut into pieces and sell to obtain the maximum profit, r_n. The price p_i of a rod of length i and the cost c_i of cutting a rod at length i is provided in the array p and c. When you calculate the profit of cutting and selling a rod at length i, you should consider the following things:

i. The profit obtainable from the two parts

ii. The cost of cutting the rod at length i

For example, if you have a rod of length 8 and you wish to cut it at length 1. Assume that the profit obtainable from the two parts are 4 Tk and 5 Tk respectively and the cost of cutting the rod at length 1 is 2 Tk. Then the total profit obtainable from cutting the rod at length 1 is Tk (4 + 5 – 2) = Tk 7. Modify **any one** of the pseudocodes below, using dynamic programming to calculate the highest profit obtainable from cutting the rod. What is the time complexity of this algorithm? [7+2]

```
r[0 ... n] array initialized to -INF
MEMOIZED-CUT-ROD-AUX(P, n, r):
    if r[n] >= 0: return r[n]
    if n == 0: q = 0
    else:
        q = -INF
        for i = 1 to n:
            q = max(q, P[i] +
                MEMOIZED-CUT-ROD-AUX(P, n-i, r))
    r[n] = q
    return r[n]
```

(i)

```
r[0 ... n] array initialized to -INF
MEMOIZED-CUT-ROD-AUX(P, n, r):
    if r[n] >= 0: return r[n]
    if n == 1:
        return P[n]
    else:
        q = P[n]
        for i = 1 to n-1:
            soln = MEMOIZED-CUT-ROD-AUX(P, i, r) +
                MEMOIZED-CUT-ROD-AUX(P, n-i, r)
        if q < soln:
            q = soln
    r[n] = q
    return r[n]
```

(ii)

(b) When the *Merge-Sort* algorithm is run on a sequence of n items ($n > 1$), if the running-time of the algorithm is $T(n)$, what are running-times of each of these three steps: divide, conquer and combine? [3]

(c) Suppose that for a *Heap* data structure, the heap size is n, and you need to determine if the i'th node is a leaf or not. How can you check this in constant time? [3]