United International University
CSI 227: Algorithms, Summer 2017
Mid Term Exam
Total Marks: 90, Time: 1 hour 45 minutes

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

1. (a) Perform **worst case running-time analysis** of the following algorithms and express in $\mathcal{O}$ notation. [6 + 4]

```
function1(m, n):
for(i=1; i <= m; ++i):
    for(j=1; j <= n; j*=3):
        if (i%2 == 0):
            print i
for(i=1; i<=n; i*=4):
    for(j=1; j <= m; ++j):
        if(j%2 == 0):
            print j
```

```
function2(n):
for(i=1; i<=n; ++i):
    for(j=1; j<=i; ++j):
        print i+j
```

   (b) What do you understand by **constant-time costs**? Provide examples of a code fragment with constant-time cost, and another not having a constant-time cost. [2+3]

2. Using the **recursion tree** method, determine a good asymptotic upper bound on the following recurrence: $T(n) = 2T(\frac{n}{3}) + \mathcal{O}(n)$. Show details of your calculation. [15]

3. (a) A Dynamic Programming algorithm for the classical **Coin Change problem** is provided at **figure 1**, where given an amount n and m types of coins $C_1, C_2, ..., C_m$, you have to minimize the number of coins to build up the amount n. Now, suppose that, each coin type C_i has a weight W_i. Modify the algorithm such that now you have to **minimize the total weight of coins** to build the amount n. [9]

   (b) Suppose that you have got a set of the following activities, with the given start and end times for the **Activity Selection** problem:

$$\{[2,4],[1,5],[9,12],[6,10],[1,16],[11,15],[7,8]\}$$

   What will be the solution sets returned by two **greedy algorithms**: one using the end times as the choosing criteria, another using activity lengths? Which one is optimal here? [2.5 + 2.5 + 1]

4. (a) Provide a **Divide-and-Conquer** algorithm to find the count of non-zero elements in an input array. What would be the costs of each of the three steps of **divide-and-conquer**? [6+3]

   (b) What is the basic difference between the algorithmic paradigms - **greedy algorithms** and **dynamic programming**? [3]

   (c) For the classical **Rod cutting** problem, design an example case where a greedy algorithm fails to find an optimal solution. What are the optimal and the greedy solutions for your example? [3]

5. (a) Suppose that you are the owner of an Internet service provider company. The maximum bandwidth capacity that you can supply is *max_capacity*. There are n customers who are willing to acquire your Internet service. Each customer has two attributes: the bandwidth B_i that he/she wants, and the payment P_i that he/she will pay for that. Provide a **Dynamic Programming** algorithm to find out the maximum total profit that you can make by providing bandwidths within your *max_capacity*. [12]

(b) Design a min-heap with height 3 and the minimum possible number of elements. All the elements must be negative here. [3]

6. (a) A dynamic programming (DP) algorithm to the classical **Rod-Cutting problem** is provided at **figure 2**. Simulate the steps of tabulation of the **DP memoization table**, along with the **recursion tree** generated by the DP algorithm, for initial rod length 6 and the profit table $P = \{1, 2, 3, 3, 4, 5\}$. [9]

(b) Propose an algorithm to convert a provided **min-heap** to a **max-heap**. [6]

```
let M be the memoization table, initially filled with NULL

COIN-CHANGE(n)

    if n = 0
        return 0

    if M[n] ≠ NULL
        return M[n]

    bestSoln = infinity

    for i = 1 to m
        if n ≤ C[i]
            soln = 1 + COIN-CHANGE(n - C[i])
            if bestSoln > soln
                bestSoln = soln

    M[n] = bestSoln
    return M[n]
```

Figure 1: **COIN-CHANGE**

```
let M be the memoization table, initially filled with NULL

CUT-ROD(length)

    if length = 0
        return 0

    if M[length] ≠ NULL
        return M[length]

    bestSoln = P[length]

    for i = 1 to length - 1
        soln = P[i] + CUT-ROD(length - i)
        if bestSoln < soln
            bestSoln = soln

    M[length] = bestSoln
    return M[length]
```

Figure 2: **CUT-ROD**

7. (a) Provide a recursive equation for the running-time $T(n)$ for the **Divide-and-Conquer** algorithm provided at **figure 3**, in terms of the input array (A) size n, where $n = r - l + 1$. You must mention the costs of each of the step of this divide-and-conquer algorithm. [6]

(b) Design examples with $|A| = n = 6$ items of both a **worst-case** and a **best-case** for the algorithm provided at **figure 4**. Also mention the runtime complexities for both cases in $\mathcal{O}$ notation. [3 + 3 + 3]

8. (a) What is the depth of the **Merge-Sort** algorithm's recursion tree, when sorting an n-element array in *non-increasing* order? What is the space complexity of **Merge-Sort**? Why? [2.5 + 2.5]

(b) At a **Min-Priority-Queue** of size n and unique elements, what are the possible indices of the maximum element? [3]

(c) Suppose that you have a **Max-Heap** with n elements. Is it guaranteed to remain a max-heap if you perform the following changes on it? Explain briefly. [2 + 2 + 3]

(i) increase the value of the root; **(ii)** decrease the value of a leaf; **(iii)** decrease the value at index 2, and $n \geq 5$.

<u>*DUMMY*(A, l, r)</u>

if $l = r$ **then**
    return $A[l]$
$mid = \left\lfloor \frac{low + high}{2} \right\rfloor$
$r_1 = DUMMY(low, mid)$
$r_2 = DUMMY(mid + 1, r)$
$sum = 0$
for $i = l$ **to** r **do**
    $sum = sum + i$
return $sum - (r_1 + r_2)$

Figure 3: **DUMMY**

```
Search(A, l, r, key):
if l<=r:
    m = (l+r)/2
    if A[m] == key:
        return m
    else if A[m] > key:
        return Search(A, l, m-1, key)
    else:
        return Search(A, m+1, r, key)
return -1
```

Figure 4: **SEARCH**