United International University
CSI 227: Algorithms, Fall 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) Derive the **exact-cost equation** for the running-time of the algorithm at **Fig. 1**, and express it in the big-Oh ($\mathcal{O}$) notation. [6]

   (b) Demonstrate the full working process of the **Divide-and-Conquer** algorithm for the **Find Minimum Subarray** problem, on the following array: $[-2, 1, 0, -5, 3, -4, 7, -5]$. [9]

```
Algorithm1(n, m):
    for(i = 1; i <= n; i = i + 1)
        print(i)

    for(j = 1; j <= m; j = j + 1)
        for(i = 1; i <= n; i = i = i + 2)
            print(i * j)
```

Figure 1: **Q. 1a**

```
Algorithm2(n):
    sum = 0

    for(i = 1; i <= n/2; i = i + 1)
        for(j = 1; j <= 100; j = j + 1)
            sum = sum + j * i

    return sum
```

Figure 2: **Q. 4c**

2. (a) Suppose you are developing a greedy algorithm for the **Rod Cutting Problem** where each cut is **greedily** made using the **maximum price**. Now design a suitable price array for rod length 6 on which the algorithm will **not** provide an optimal solution. What is the optimal solution for your example? [3]

   (b) You are going to build a new computer system that requires t watts of power to run. But there might not be power chips available at the market with exactly t watts. There are n types of power chips available, with the following capacities: $P_1, P_2, ..., P_n$. You want to assemble the t watts using the **minimum number** of power chips. Propose a **Dynamic Programming** algorithm to solve this problem. [12]

3. (a) Design an algorithm that provided two parameters n and m, runs in time $\mathcal{O}(n + m \times logm)$. [7]

   (b) Add **memoization** to the algorithm in **Fig. 3** to get its **Dynamic Programming** version. Then calculate *CATALAN(3)* using the memoized algorithm. You must demonstrate the recursion-tree generated. [3+5]

```
CATALAN(n):
    if n = 0
        return 1

    c = 0
    for i = 0 to n - 1
        c = c + [CATALAN(i) * CATALAN(n - i - 1)]

    return c
```

Figure 3: **Q. 3b**

$$T(n) = \begin{cases} \mathcal{O}(1), & \text{if } n = 1; \\ 3T(\frac{n}{6}) + \mathcal{O}(n), & \text{otherwise.} \end{cases}$$

Figure 4: **Q. 7a**

4. (a) *Trump coins* are used by the people of the *Trump land* for everyday transactions. Only the following coins are available at this system: $1, 7, 12, 25$. Provide an example of an amount where the **greedy** strategy for the **Coin-Change** problem does not provide an optimal solution. You must mention the optimal solution, and the greedy solution. [3]

   (b) Suppose that you run a server lending business, where you have got the the following server-use requests for the next month. Each request is at the format: *[start date, end date]*. A server can be used by **at most one** user at a time. Using a **greedy algorithm**, find out the **minimum** number of servers required to satisfy all the requests. $\{[6, 10], [3, 5], [3, 8], [4, 9], [1, 7], [1, 2], [3, 7], [9, 11]\}$ [6]

(c) Derive the **exact-cost equation** for the running-time of the algorithm at **Fig. 2**, and express it in the big-Oh ($\mathcal{O}$) notation. [6]

5. (a) Consider this modified version of the **Merge sort** algorithm as follows: divide the provided array of size n into **three** subarrays of sizes roughly $\frac{n}{3}$, sort each of these three subarrays recursively, and then combine the three sorted subarrays in time $\mathcal{O}(n)$. Answer the following:

   i. Design a **recurrence relation** for the running-time $T(n)$ of this algorithm. [3]

   ii. Using the **recursion-tree method**, solve the recurrence and derive its running-time in the big-Oh ($\mathcal{O}$) notation. [7]

   (b) Given the arrival and the departure times of 6 trains for a railway platform, find out the **maximum** number of trains that can use that platform without any collision, using a **greedy algorithm**. There **must exist at least** 10 **minutes of safety break** between the departure of one train and arrival of a next one. $\{[1000, 1030], [840, 1030], [850, 1040], [1700, 2000], [800, 835], 1300, 1800]\}$ [5]

6. (a) Take a look at the algorithm at **Fig. 5**. Now provide **both the best-case and worst-case examples** of the arrays A and B for $|A| = n = 4$ and $|B| = m = 5$, and $val = 10$. Also derive the **running-time** complexities for both the cases in the big-Oh ($\mathcal{O}$) notation. [3 + 3 + 5]

   (b) For the **Divide-and-Conquer** algorithm for the **Find Maximum Subarray** problem, if the cost for finding the *maximum crossing sum* is $\mathcal{O}(n^2)$, then what will be the **recursive equation** for the running-time of the algorithm? [4]

```
Algorithm3(A, B, val):

    n = A.length
    m = B.length

    for(i = 1; i <= n; i = i + 1)
        print(A[i])

    for(j = 1; j <= n; j = 2 * j)
        for(k = 1; k <= m; k = 2 * k)
            if(A[j] * B[k] == val)
                return j

    return -1
```

Figure 5: **Q. 6a**

```
n: total number of items
V: array of values for the items
W: array of weights for the items
M: DP memoization table,
    initially filled with -1

KNAPSACK(index, maxWeight):
    if i > n
        return 0

    if M[index][maxWeight] != -1
        return M[index][maxWeight]

    if W[index] > maxWeight
        val = KNAPSACK(index + 1, maxWeight)
    else
        val1 = KNAPSACK(index + 1, maxWeight)
        val2 = V[index] + KNAPSACK(index + 1, maxWeight - W[index])
        val = max(val1, val2)

    M[index][maxWeight] = val
    return val
```

Figure 6: **Q. 8a**

7. (a) Using the **recursion-tree method**, find out an asymptotic upper bound in the big-Oh ($\mathcal{O}$) for the recurrence in **Fig. 4**. [12]

   (b) What is the fundamental difference between **Greedy Strategies** and **Dynamic Programming**? [3]

8. (a) A **Dynamic Programming** algorithm for the classical **0/1 Knapsack problem** is provided in **Fig. 6**. Consider the following added restrictions to the problem: if you do not take the i^{th} item, you have to pay a penalty P_i, and if you take it, you have to pay a transportation fee T_i. Update the algorithm such that it can solve this modified problem. [7]

   (b) Provide a **Divide-and-Conquer** algorithm to find the count of negative (< 0) elements in an input array A. Also, mention the running-times of each of the **three steps** of the divide-and-conquer strategy in your algorithm. [5 + 3]