National University of Computer and Emerging Sciences, Lahore Campus

	Course Name:	Design and Analysis of Algorithms	Course Code:	CS2009
	Degree Program:	BSCS	Semester:	Spring 2023
	Due Date:	Wednesday/Thursday, April 05/06, 2023	Total Marks:	10 + 10 + 10 + 15 = 45
	Section:	ALL	Page(s):	2
	Exam Type:	Assignment		

Student : Name:_________________________________ Roll No.__________________ Section:________

Instruction/Notes:

Q1: Dynamic Programming

a) Palindrome Partitioning:

Given a string, a partitioning of the string is a palindrome partitioning if every substring of the partition is a palindrome. For example, "aba | b | bbabb | a | b | aba" is a palindrome partitioning of "ababbbabbababa". Your task is to design an algorithm that determines the fewest cuts needed for palindrome partitioning of a given string. For example, minimum 3 cuts are needed for "ababbbabbababa". The three cuts are "a | babbbab | b | ababa". If a string is palindrome, then minimum 0 cuts are needed. If a string of length n containing all different characters, then minimum n-1 cuts are needed.

b) Billboards on a Highway:

Suppose you are managing the construction of billboards on a Highway, a heavily travelled stretch of road that runs west-east for M miles. The possible sites for billboards are given by numbers X= x1, x2, . . . , xn, each in the interval [0,M] (specifying their position along the highway, measured in miles from its western end). If you place a billboard at location xi, you receive a revenue of ri> 0. Regulations imposed by the county's Highway Department require that no two of the billboards be within less than or equal to 5 miles of each other. You'd like to place billboards at a subset of the sites to maximize your total revenue, subject to this restriction. Device an efficient algorithm that takes the input X and compute the sites for billboards that maximize the revenue. Give the pseudocode of your algorithm, also compute its time complexity.

Q2: Greedy Algorithms

a) **Minimize Waiting Time:**

A server has n customers waiting to be served. The service time required by each customer is known in advance: it is t_i minutes for customer i. So if, for example, the customers are served in order of increasing i, then the i th customer has to wait $\sum_1^i t_i$ minutes. We wish to minimize the total waiting time $T = \sum_{i=1}^{n}$ (Time spent waiting by cutomer i).

Give an efficient algorithm for computing the optimal order in which to process the customers. Also compute its time complexity.

b) **Activity Selection Problem:**

Part 1:

Suppose that instead of always selecting the first activity to finish, we select the last activity to start that is compatible with all previously selected activities. Describe how this approach is a greedy algorithm, and prove (see theorem 16.1) that it yields an optimal solution.

Write pseudocode (both recursive as well as iterative) to solve the activity selection problem using the above-mentioned greedy approach.

Let $S_k = \{a_i \in S : f_i \leq s_k\}$ be the set of activities that finish before activity a_k finishes. If we make the greedy choice of activity a_n (here we assume that the activities are **sorted by their start time**, hence activity a_n is the last activity to start), then S_n remains as the only subproblem to solve. Optimal substructure tells us that if a_n is in the optimal solution, then an optimal solution to the original problem consists of activity a_n and all the activities in an optimal solution to the subproblem S_n.

Theorem:

Consider any nonempty subproblem S_k, and let a_m be an activity in S_k with the latest start time. Then a_m is included in some maximum-size subset of mutually compatible activities of S_k.

Proof: Let A_k be a maximum-size subset of mutually compatible activities in S_k, and let a_j be the activity in A_k with the last start time. If $a_j = a_m$, we are done, since we have shown that a_m is in some maximum-size subset of mutually compatible activities of S_k. If $a_j \neq a_m$, let the set $\acute{A}_k = A_k - \{a_j\} \cup \{a_m\}$ be A_k but substituting a_m for a_j . The activities in $\acute{A}_k$ are disjoint, which follows because the activities in A_k are disjoint, a_j is the last activity in A_k to start, and $s_m \geq s_j$. Since

$|\acute{A}_k| = |A_k|$, we conclude that $\acute{A}_k$ is a maximum-size subset of mutually compatible activities of S_k, and it includes a_m.

Recursive Pseudocode: We assume that the n input activities are already ordered by monotonically increasing **start** time. If not, we can sort them into this order in $O(n \lg n)$ time, breaking ties arbitrarily. In order to start, we add the fictitious activity a_{n+1} with $s_{n+1} = \infty$, so that subproblem S_{n+1} is the entire set of activities S. The **initial call**, which solves the entire problem, is RECURSIVE-ACTIVITY-SELECTOR(s, f, n+1).

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k)
    m = k - 1
    while m ≥ 1 and f[m] > s[k]
        m = m - 1
    if m ≥ 1
        return {$a_m$} ∪ RECURSIVE-ACTIVITY-SELECTOR(s, f, m)
    else return ∅
```

Iterative Pseudocode: We assume that the n input activities are already ordered by monotonically increasing **start** time.

```
GREEDY-ACTIVITY-SELECTOR(s, f)
    n = s.length
    A = {$a_n$}
    k = n
    for m = n-1 downto 1
        if f[m] ≤ s[k]
            A = A ∪ {$a_m$}
            k = m
    return A
```

Part 2:

Not just any greedy approach to the activity-selection problem produces a maximum-size set of mutually compatible activities. Give an example to show that the approach of selecting the activity of least duration from among those that are compatible with previously selected activities does not work. Do the same for the approaches of always selecting the compatible activity that overlaps the fewest other remaining activities and always selecting the compatible remaining activity with the earliest start time.

Counterexample for selecting the activity of least duration:

$$(1,5), (4,6), (5,10)$$

Counterexample for selecting the compatible activity that overlaps the fewest other remaining activities:

$$(1,3), (2,4), (2,4), (2,4), (3,5), (4,6), (5,8), (7,9), (7,9), (7,9), (8,10)$$

Counterexample for selecting the compatible remaining activity with the earliest start time:

$$(1,10), (5,8), (8,11)$$

Part 3:

Consider a modification to the activity-selection problem in which each activity a_i has, in addition to a start and finish time, a value v_i . The objective is no longer to maximize the number of activities scheduled, but instead to maximize the total value of the activities scheduled. That is, we wish to choose a set A of compatible activities such that $\sum_{a_k \in A} v_k$ is maximized. Give a **polynomial-time** algorithm for this problem.

$$c[i,j] = \begin{cases} 0 & \text{if } \; S_{ij} = \emptyset \\ \max\limits_{a_k \in S_{ij}} \{c[i,k] + c[k,j] + v_k\} & \text{if } \; S_{ij} \neq \emptyset \end{cases}$$

We assume that the n input activities are already ordered by monotonically increasing **finish** time. We add the fictitious activities a_{n+1} with $s_{n+1} = \infty$ and a_0 with $f_0 = 0$, so that subproblem $S_{0\,n+1}$ is the entire set of activities S.

```
ActivitySelector(s,f,v)
   n=s.length
   Let c[0…n+1,0…n+1],d[0…n+1,0…n+1] be new arrays
   for i=0 to n+1
       c[i,i]=0
   diagonal=1
   for diagonal = 1 to n+1
       for i = 0 to (n+1)-diagonal
           j=i+diagonal
           max=0
           activity=NIL
           for k = i+1 to j-1
               if s[k] ≥ f[i] and f[k] ≤ s[j]
                    value = c[i,k]+c[k,j]+v[k]
                    if max < value
                         max=value
                         activity=k
           c[i,j]=max
           d[i,j]=activity
   return c and d
```