**COMP3011: Design and Analysis of Algorithms** Due: 31 Oct 2024

# Assignment 2

*The Hong Kong Polytechnic University*

**Problem 1** (25 points)

Our TA suggests the following variant on mergesort: instead of splitting the list into two halves, we split it into three thirds. Then we recursively sort each third and merge them.

Mergesort3($A[0, \cdots, n-1]$):

1. If $n \leq 1$, then return $A[0, \cdots, n-1]$.
2. Let $k = \lceil n/3 \rceil$ and $m = \lfloor 2n/3 \rfloor$.
3. Return Merge3(Mergesort3($A[0, \cdots, k-1]$), Mergesort3($A[k, \cdots, m-1]$), Mergesort3($A[m, \cdots, n-1]$)).

Merge3($L_0, L_1, L_2$):

1. Return Merge($L_0$, Merge($L_1, L_2$)).

Assume that you have a subroutine Merge that merges two sorted lists of lengths $l, l'$ in time $O(l + l')$. You may assume that $n$ is a power of three, if you wish.

(a) What is the asymptotic running time for executing Merge3($L_0, L_1, L_2$), if $L_0, L_1, L_2$ are three sorted lists each of length $n/3$? Express your answer using $O()$ notation.

(b) Let $T(n)$ denote the running time of Mergesort3 on an array of size $n$. Write a recurrence relation for $T(n)$.

(c) Solve the recurrence relation in part (b). Express your answer using $O()$ notation.

**Problem 2** (25 points)

Recall the Longest Common Subsequence problem, where we are given two strings, $S_1$ and $S_2$, the task is to find the length of the Longest Common Subsequence, i.e. longest subsequence present in both of the strings. A longest common subsequence (LCS) is defined as the longest subsequence which is common in all given input sequences. Use the dynamic programming algorithm to compute the LCS between $S_1$ = "quantity" and $S_2$ = "inequality". You need to present the intermediate results.

**Problem 3** (25 points)

There are $n$ points on a road you are driving your taxi on. The $n$ points on the road are labeled from 1 to $n$ in the direction you are going, and you want to drive from point 1 to the direction of point $n$ to make money by picking up passengers. You cannot change the direction of the taxi. The passengers are represented by an array $rides$, where $rides[i] = [start_i, end_i, tip_i]$ denotes the $i$th passenger requesting a ride from point $start_i$ to point $end_i$ who is willing to give a $tip_i$ dollar tip. We have $end_i \geq start_i$ for all $i$. For each passenger $i$ you pick up, you earn $end_i - start_i + tip_i$ dollars. You may only drive at most one passenger at a time. All the numbers are positive integers.

Given $n$ and the array $rides$, design an $O(n \log n)$-time algorithm to return the maximum number of dollars you can earn by picking up the passengers optimally.

Note: You may drop off a passenger and pick up a different passenger at the same point.

**Problem 4** (25 points)

We are given $n$ rectangles, where each rectangle can be described by two integers $a > 0$ and $b > 0$ (i.e, the length and width respectively). Rectangle $X(a, b)$ can be put inside of rectangle $Y(c, d)$ if and only if (1) $a < c$ and $b < d$ *or* (2) $b < c$ and $a < d$ (i.e., the rectangle can be rotated 90°). For example, $(1, 5)$ can be put inside of $(6, 2)$ but not inside of $(3, 4)$. The task is to find the maximum number of rectangles that can be arranged in a line so that every rectangle (except the last one) can be put inside of the next one.

Design a polynomial-time algorithm for the above problem, and justify the correctness and the running time of the algorithm.