School Name: Computer Science and Engineering

COURSE CODE – COURSE NAME:

COMP3821 Extended Algorithms and Programming Techniques/

COMP9801 Extended Design and Analysis of Algorithms

Assessment type/name: Midterm Examination 2019T2

- Reading Time: 10 minutes
- Time allowed: 2 hours including reading time.
- Answer all questions in all 4 parts
- You cannot keep the examination paper
- You MUST WRITE VERY LEGIBLY
- Calculators are NOT allowed
- This paper must be completed in ink
- This exam is closed books
- Only a double-sided A4 sheet of personal notes is allowed.
  The notes can be handwritten and/or typed.
- You can keep the nodes at the end of the examination.

Part 1 (3 questions)

An array A of length n is *single-peaked* if there exists an index $1 \leq p \leq n$ such that for $1 \leq i < j \leq p$ we have $A_i < A_j$ and for $p \leq i < j \leq n$ we have $A_i > A_j$. We call p the *peak* of A.

Question 1.1 Among the following arrays, which one are **single-peaked**? (Tick the correct answer(**s**).)

☐ $[1, 2, 5, 6, 2]$	☐ $[1, 2, 3, 4, 5]$	☐ $[9, 7, 3, 4, 5]$
☐ $[2, 3, 1, 4, 0]$	☐ $[5, 4, 3, 2, 1]$	☐ $[3, 4, 6, 6, 5]$

We would like to design an algorithm that solves the following task T: take as input a single-peaked array A of length n, as well as a value x and determine if x appears in A. Note that A is assumed to be single-peaked, but the peak is not given explicitely in the input.

Question 1.2 Give the pseudo-code for an $O(\log n)$ algorithm for task T. It may be useful to define subroutines and call them from your algorithm.

Question 1.3 Give an informal argument justifying that the worst-case complexity of your efficient algorithm is $O(\log n)$. (One or two lines in English)

Part 2 (5 questions)

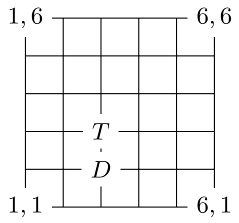
You are searching for a buried treasure on a beach. The beach is a square with side length n for some positive integer n. You the treasure is buried at unknown integer coordinates (t_x, t_y). In order to help you find the treasure you have a metal detector. The metal detector works as follows:

- First, place the metal detector at integer coordinates (d_x, d_y) of your choosing.
- Then activate the detector. It will beep twice if and only if $(t_x < d_x \wedge t_y < d_y) \vee (t_x > d_x \wedge t_y > d_y)$. Otherwise it will beep once.

Question 2.1 In the following scenarios where $n = 6$, the treasure is indicated with a T, the detector is indicated with a D, and some coordinates are highlighted. Indicate in each case whether the detector would bip once or twice.

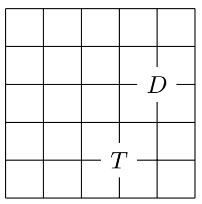
☐ Once

☐ Twice



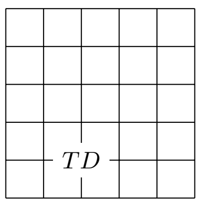
☐ Once

☐ Twice



☐ Once

☐ Twice



Question 2.2 Since the battery in your metal detector is running low, you wish to activate it as little as possible. Give the pseudo-code for an algorithm that will allow you to find the buried treasure with as few activations of the detector as you can (asymptotically). Your pseudo-code will use the subroutine `detect(`x`,`y`)` that takes two coordinate arguments and returns `true` if the metal detector would bip twice from these coordinates and returns `false` if the detector would bip just once.

Question 2.3 What is the worst-case complexity of your algorithm in terms of number of calls to `detect`?

Question 2.4 Prove that the worst-case complexity of your algorithm is indeed what you have answered in the previous question.

Question 2.5 Prove that your algorithm is correct.

Part 3 (4 questions)

Question 3.1 Compute, by any method you wish, the linear convolution of the sequence $R = \langle 4, 0, 3, 1 \rangle$ with itself. Only give the result of $R \star R$ as an answer, you do not need to write the intermediate steps.

Question 3.2 Explain in English how to *efficiently* compute the linear convolution of two (long) sequences.

Question 3.3 What is the time complexity of this efficient method? Only give the *Big-Oh* worst-case complexity with no justification.

Question 3.4 Consider the sequence $S = \langle 1, \underbrace{0, 0, ..., 0}_{n-2}, 1+i \rangle$ of length n. Compute $\hat{S}$, the DFT of S. Give the result by filling the following table without writing the intermediate steps of your computation.

		Real part	Imaginary part
Example:	$e^{\frac{\pi i}{3}}$	$\frac{1}{2}$	$\frac{\sqrt{3}}{2}$
	$e^{\theta i}$	$\cos\theta$	$\sin\theta$
	$\hat{S}_0$		
	$\hat{S}_1$		
DFT of S:	$\vdots$	$\vdots$	$\vdots$
	$\hat{S}_k$		
	$\vdots$	$\vdots$	$\vdots$
	$\hat{S}_{n-1}$		

Part 4 (3 questions)

You are given n white dots and n black dots, lying on a line, equally spaced. The dots appear in any order of black and white.

The goal is to connect each white dot with a (different) black dot while minimising the total length of wire used. Each dot should be connected to a single other dot. When solving this on a straight line, a greedy algorithm is sufficient. Simply connect the first (leftmost) white dot with the first black dot, the second white dot with the second black, and so on and so forth. You may assume this algorithm is $O(n)$. This algorithm is implemented in a subroutine `greedy-line(`A`)`. The input is an array A of boolean values such that $A_i =$ `true` if the dot at location i is black and $A_i =$ `false` if it is white. The input array has as many `true` values as `false` values. `greedy-line` returns the amount of wired used by the greedy solution as an integer.

We now consider the modified problem where the dots are placed equidistantly on a *circle*, and the length of wire required to connect two dots is the shortest arc length between them. The radius of the circle is chosen such that the arc length between two consecutive dots is 1. The input is a boolean array A of length $2n$ such that $A_j =$ `true` if and only if the dot at angle $\frac{j\pi}{n}$ is black. The input array has as many `true` values as `false` values. We would like to compute the way to connect each white dot with a black dot using a minimum length of wire.

Question 4.1 Prove that in an optimal solution, there will be at least one pair of consecutive dots which has no wires between them.

Question 4.2 Give the pseudo-code of a *polynomial time* algorithm to solve the *circle* problem and return the total length of wire required. You may use calls to `greedy-line` in your implementation.

Question 4.3 What is the worst-case complexity of your algorithm?