Main Exam

● Graded

Student

Lihang Shen

Total Points

33.5 / 60 pts

Question 1

Problem 1 — **8** / 10 pts

1.1 **Problem 1a** — **3** / 5 pts

✔ **– 2 pts** Incorrect running time, but good explanation

1.2 **Problem 1b** — **5** / 5 pts

✔ **– 0 pts** Correct

Question 2

Problem 2 — **10** / 10 pts

Part a

✔ **– 0 pts** Correct counterexample

Part b

✔ **– 0 pts** Correct description of algorithm output with good explanation

Part c

✔ **– 0 pts** Correct optimal solution

Question 3

Problem 3

6.5 / 10 pts

Blanket Answers

✔ **– 0 pts** Correct in required time

Algorithm

✔ **– 1.5 pts** Explanation is unclear

Correctness Proof

✔ **– 2 pts** Unclear correctness argument

Complexity Analysis

✔ **– 0 pts** Correct running time and correct explanation

1 but how there could be two paths that passes only one vertex in S. How would you pick the shortest?

Question 4

Problem 4

4 / 10 pts

Blanket Answers

✔ **– 2 pts** Incorrect or very unclear in required time and space

Algorithm

✔ **– 1.5 pts** Explanation is unclear

Correctness Proof

✔ **– 2.5 pts** Incorrect argument

Incorrect algorithm: If you build a BST based on one end of the segment, you can't effectively search all valid segments in log(n) time during querying.

Question 5

Problem 5 **5** / 20 pts

5.1 **Problem 5 a-c** **4** / 16 pts

Blanket Answers

✔ **– 7 pts** Not a divide and conquer algorithm

Algorithm

✔ **– 1 pt** Minor error

Correctness Proof

✔ **– 2 pts** Unclear correctness argument

Complexity Analysis

✔ **– 2 pts** Incorrect time complexity analysis, without reasonable explanation

5.2 **Problem 5d** **1** / 2 pts

✔ **– 1 pt** Correct without reasonable argument

5.3 **Problem 5e** **0** / 2 pts

✔ **– 2 pts** Incorrect without reasonable argument

THE UNIVERSITY OF SYDNEY

Student number: 490051481

Room number: A14.02.N270

Seat number: 10

ANONYMOUSLY MARKED
(Please do not write your name on this exam paper)

CONFIDENTIAL EXAM PAPER

This paper is not to be removed from the exam venue.

Computer Science

EXAMINATION

Semester 2- Main, 2024

COMP9123 Data Structures and Algorithms

EXAM WRITING TIME: 120 minutes
READING TIME: 10 minutes

EXAM CONDITIONS:
This is a RESTRICTED OPEN book examination
- specified materials permitted.

MATERIALS PERMITTED IN THE EXAM VENUE:
One A4 sheet of handwritten and/or typed notes double-sided.
(No electronic devices of any kind are permitted.)

MATERIALS TO BE SUPPLIED TO STUDENTS:
This exam paper booklet.

INSTRUCTIONS TO STUDENTS:
Write your student ID at the top right of this front page.
Write your answers in the spaces provided in the exam paper booklet using blue or black ink. DO NOT ATTACH EXTRA PAGES TO THIS EXAM BOOKLET. If you run out of space in the space provided right after each problem statement, continue your answers on pages 10-16.

For examiner use only:

Problem	1	2	3	4	5	Total
Marks						
Out of	10	10	10	10	20	60

Problem 1. (10 points)

a) Let T be a binary search tree containing n nodes. Consider the following algorithm that takes as input a node $u \in T$.

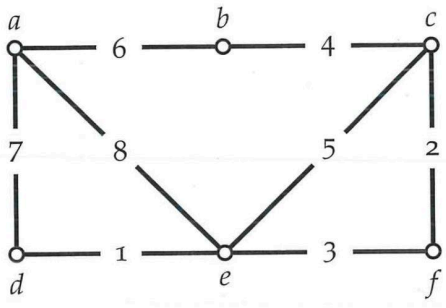
```
1: def FOO(u)
2:     BAR(u)
3:     if u.left ≠ null then
4:         FOO(u.left)
5:     else if u.right ≠ null then
6:         FOO(u.right)
```

From root, the algorithm will traverse one side (either left or right)

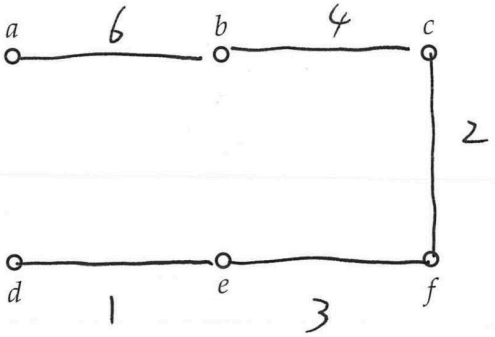
Your task is to find the time complexity of FOO($T.root$) when BAR(u) runs in $O(|T_u|)$ time, where $|T_u|$ is the size of the subtree of T rooted at u. Provide the tightest time complexity bound possible (in terms of n) and briefly justify your bound.

- The worst case of BST is an array (since it's unbalanced)
- Then for each iteration of recursion, BAR(u) will take $n-i$ Time where "i" is the iteration number
- Finally, we should get $O(n \cdot (n-1) \cdot \ldots \cdot (1))$ That's $O(n!)$ $\frac{n(n+1)}{2}$
- In addition, the algorithm will traverse one side of BST

b) Consider the weighted graph G on the left. Draw the minimum spanning tree (MST) of G on the right. You do **not** have to explain your answer.

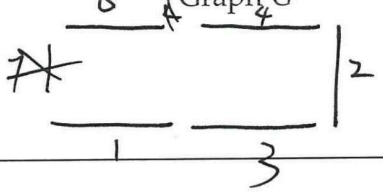


Graph G



MST(G)

MST



Problem 2. (10 points) We're given a non-empty set S of elements and a collection C of sets $C = \{C_1, C_2, C_3, ..., C_k\}$. Each $C_i \in C$ contains a subset of the elements of S and every element in S is contained in at least one C_i. We need to find that smallest number of sets C_i such that every element $e \in S$ is contained in at least one of the sets we picked.

For example, if $S = \{1,2,3,4\}$ and $C = \{\{1,4\},\{2,3\},\{2\},\{3\}\}$, returning $\{1,4\}$ and $\{2,3\}$ would be an optimal solution, as those two sets cover all elements in S. Returning $\{1,4\}$, $\{2\}$, and $\{3\}$ would not be optimal, as this isn't the smallest number of sets that we could return. Returning $\{1,4\}$ is incorrect, as we don't return any set covering elements 2 and 3 in S.

Construct a counterexample for the following algorithm for this problem: Sort the sets of C in non-decreasing order based on the number of elements a set contains. Process the sets one at a time in this order and add a set to the solution if it contains an element of S that isn't contained by any other set in the solution yet.

Remember to:

a) describe your instance,

b) show what solution the algorithm gives for the instance and briefly explain why, and

c) show what the optimal solution of the instance is.

(a) $\{\{1,2\}\ \{2,3\}\ \{3,4\}\ \{4,1\}\} = C$ $S = \{1,2,3,4\}$

C_1 C_2 C_3 C_4

(b) Since there is a tie, and algorithm said we should sort them in non-decreasing order. so we can sort them in following

C_1 (1,2)
C_2 (2,3)
C_3 (3,4)
C_4 (4,1)

⇒

① Then we pick (1,2) because result do not have 1 and 2

② Then we pick (2,3) because result do not have 3

③ Then we pick (3,4) because result do not have 4

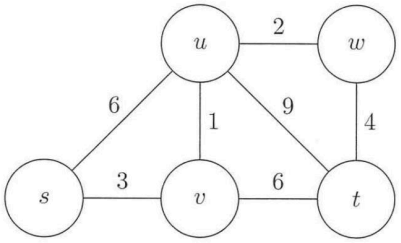
(solution to Problem 2 continues here)

(C) as you may notice, the best solution is (1,2) (3,4) which only need ~~to~~ 2 sets

But ~~my~~ my counter example takes 3. You can ~~also~~ also find other counter solution when you ~~arrange~~ sort C in(a) in a different way. Such as (2 3)
(3 4)
(4 1)
(1 2)

Problem 3. (10 points) Let $G = (V, E)$ be a simple weighted undirected graph consisting of n vertices and m edges, given to you in adjacency list representation. All edge weights are non-negative integers. You're also given a set of k vertices $S \subset V$ (where $1 \leq k \leq n-2$, i.e., k is **not** a constant). You're asked to design an algorithm that computes, for two given input vertices $s \in V$ and $t \in V$, the length of the shortest path from s and t that includes exactly one vertex of S, if such a path exists. If no such path exists, you can return ∞. For simplicity, you can assume that $s \notin S$ and that $t \notin S$, and the shortest path you return the length of doesn't have to be simple.

Example:



In this example, when $S = \{u, w\}$, the shortest path from s to t via exactly one of u and w is $s - v - u - v - t$. Hence, your algorithm should return the length of this path, i.e., 11 = 3 + 1 + 1 + 6. The path $s - v - u - t$ also goes via exactly one vertex in S, but its length is 13, making it not the shortest one. The path $s - v - t$ doesn't go via any vertex in S and is thus not considered. Similarly, the path $s - v - u - w - t$ isn't considered, as it goes via two vertices in S.

Your task is to give an algorithm for this problem that runs in $O(m + n \log n)$ time (i.e., its running time does **not** depend on k). Remember to:

a) describe your algorithm in plain English,

b) argue its correctness, and

c) analyze its time complexity.

(a) ① First, let's do a Dijkstra at node S. which takes $O(m+n\log n)$ if we use Fib heap for SPT. We call it SPT_s. we will stop a branch if we meet 2 nodes $\in S$

~~② Secondly do the same thing for node t. we call it SPT_t~~

~~③ store~~

② For each node in SPT_s (not include root), ~~if it inside~~ ~~then~~ we use Hash Map (Cuckoo Hashing) to store it. the key is node name, value is the distance from root to node. and a var "Has", I will talk about it later

(solution to Problem 3 continues here)

~~(3) For the node inside Hash Map,~~
we do

·In addition, whenever we meet ~~2~~ nodes $\in S$ in a branch, we should stop traverse down the SPTs, because we only allowed to have one node in S as mid point

(3) For node t, we do the same thing we did in ①,②

(4) We can use get() method for cuckoo hashing Maps to see whether there is a node in hash Map have 2 values in Hash Table 1 and Hash Table 2.

(4.1) If none, then return ∞

(4.2) If only one, return the sum of two value

(4.3) If multiple, compare them and get the shortest one.

~~But this time, we only record the path with exactly one node is S or no node in S~~

the node in Hash Table cell also have a variable indicate whether there is a node ~~in the path between s and~~ in the path belongs to S. We call it "Has" and its boolean type.

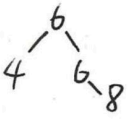
and only one of element in Hash Table has variable "Has == True"

HAS = True — HAS = False

see page 13

Problem 4. (10 points) You are given an array A containing n line segments in 1D (i.e., along the x-axis). The i-th line segment $A[i]$ is given as a tuple of its two endpoints $(left_i, right_i)$. You're asked to design a data structure that supports the following query: given a coordinate x, return the number of line segments that contain x. For simplicity, you can assume that x is distinct from all left and right endpoints of the line segments.

Example:
$A = [(1,4),(5,8),(1,6),(4,6)]$



If we query the data structure with $x = 2$, the data structure should return 2, since both $(1,4)$ and $(1,6)$ contain x. If we query with $x = 7$, we should return 1, since only $(5,8)$ contains x. If we query with $x = 10$, we should return 0, since no line segment contains x.

Your task is to design a data structure for the above problem that uses $O(n)$ space, can be constructed in $O(n \log n)$ time, and supports the query in $O(\log n)$ time. You do **not** need to support insertion and deletion of line segments. Remember to:

a) Describe your data structure implementation in plain English.

b) Prove the correctness of your data structure.

c) Analyze the time and space complexity of your data structure.

(a) ~~We use greedy Algorithm~~ AVL Tree

Value

· ~~sort~~ ① We Initialize a ~~A~~ BST tree, the ~~value~~ for each node is $left_i$, ~~value~~ key is $right_i$, we sort the Balanced BST according to node's key (i.e. ~~left~~ $right_i$)

② When we query, we just use x compare to the node's key,

②.1 if x > node's key ~~then go to firstly check current node's value, if value ≤ x~~ go to right sub tree, and check again

②.2 if ~~x~~ x ≤ node's key, check current node's value, if value ≤ x, then CONNT ++

also ~~Go to~~ Do the same thing for current node's children

(solution to Problem 4 continues here)

(b) ~~It takes O(n) space~~

It's correct because of following possibly

- if $X \leq$ node.value then X could belong to current line segment, we need check node's value for more details to judge whether it contains X
- If X > node value, then X could never belong to current line segment since the line stops before X, so we need to go right subtree. Because we want to find weather there ave other tree has $right_i \geq X$

My algorithm also check all nodes whose Key (i.e. $right_i$) $\geq X$. These make sure I check all possible cases.

(C) Construct a Balanced BST takes O(n log n)

- Query step: we simply check from root to leaf node so its log(n) (since balanced BST)
- Space is O(n) because there are exactly n nodes, each one correspond to a line

Problem 5. (20 points) Let A be an array containing n distinct integers, representing locations of wireless transponders. Each transponder has an identical range d. Two wireless transponders i and j can communicate with each other if and only if $|A[i] - A[j]| \leq d$, i.e., if the absolute difference between their locations is at most the distance they can cover. Recall that $|x|$ equals x when $x \geq 0$ and $|x|$ equals $-x$ if $x < 0$. You need to determine how many pairs of transponders can communicate with each other. Your task is to compute the number of pairs of wireless transponders that can communicate with each other.

Example: 2 4 7 12

When $A = [4, 2, 12, 7]$ and $d = 3$, you should return 2, since $|4 - 2| = 2 \leq d$ and $|4 - 7| = 3 \leq d$ and no other pair is at distance at most d from each other.

Your task is to design a divide and conquer algorithm for the above problem. For full marks, your algorithm should run in $O(n \log n)$ time. Remember to:

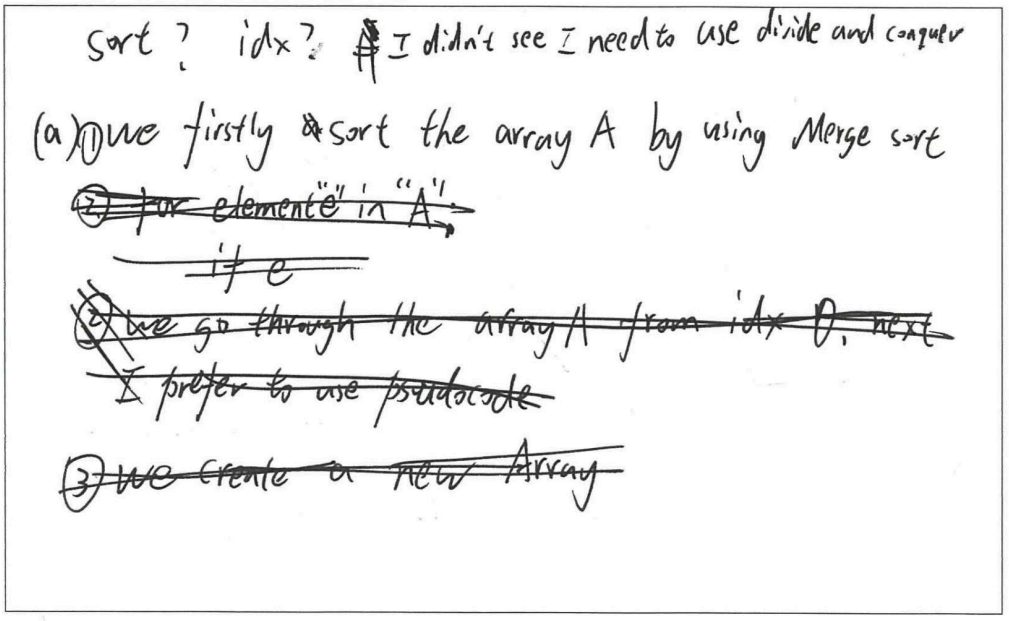
a) describe your algorithm in plain English,

b) argue its correctness, and

c) analyze its time complexity. (If you use the Master Theorem, don't forget to specify the parameters you get when you apply, so we can see you understand how this works.)

d) Also answer the following question: what would the running time of your algorithm become if the input is given as a doubly-linked list instead of an array. Briefly explain your answer.

e) Also answer the following question: can you solve this problem faster if the input is guaranteed to be sorted (and stored in an array)? Briefly explain your answer.



[0, 1, 3, 7] d=2
2 3

(solution to Problem 5 continues here)

~~for idx < length(A):~~
~~if A[idx]+d~~

② for idx = length (A):
if A[idx]+d ≥ A[idx+d]:
PAIR += d;
idx += d;
else ~~A[idx]+d < A~~
~~# A[idx]+d < A[idx+d]~~ ~~do it similar~~
~~we do another for-loop, we check it by~~
~~to traverse down a balanced BST.~~
~~That's for cut the idx+d half, check~~

- we do binary search to find a element closest to A[idx]+d and less than it
- we can do it by ~~cut~~ halt d, that's firstly A[idx] + $\frac{d}{2}$, then check, A[idx] + $\lfloor \frac{d}{4} \rfloor + 1$ or A[idx] + $\lfloor \frac{3}{4} d \rfloor$

check

- If we find one, then PAIR += its idx
- ~~If will at~~ If no we just go to next element in A
- we can do it by doing idx++

(B)

(12) (15)

~~"~~ see additional page ~~for~~ "11" for rest

THIS AND FOLLOWING PAGES LEFT INTENTIONALLY BLANK.

Use this and following pages for additional writing space if you run out of space in an answer area. If you want any writing on this page or following pages to be marked, you MUST write "see additional pages" in the relevant answer area and clearly indicate the question number you are answering on the relevant page. Any writing outside of this booklet will not be examined.

(3) Finally we just return PAIR

(b). It's correct because I check every possible pair for each element in A (A is sorted)

- I do it by ~~adding~~ comparing A[Idx]+d and A[Idx+d]
- I also use binary search to find the farest possible a method similar to neighbour for A[Idx] within range d.
- Since every element is unique, so I am allowed to do above things
- I never check "left side" of an element, so there is not repeat pair
- After the for-loop, all pair can be get successfully.

(c) ① takes O(nlogn) because Merge sort

② takes O(n logn) because there is a for loop, and inside for loop, I just do a method similar to binary search. That's O(n)[O(logn)] = O(nlogn)

Hence we have O(nlogn) in total

X. see next page

(use the rest of the space of rough work)

(d), the time complexity for ① does not change

. $T(n) = 2T(\frac{n}{2}) + O(n) \Rightarrow O(n \log n)$ ### $n^{\log_2 2} = n$ case 2

. the time complexity for ② does not change

refer to the binary search method in Lecture by using linked list

* So the overall complexity doesnot change

you can Imagine each time output the first node of each sub linked list

↓

no difference from Array

(e) I don't think I can,

· it just simplify my step ①

· I still need to check for every element in A.

(use the rest of the space of rough work)

Problem 3

(b). We just do 2 Dijkstra for s and t respectively.

- We can get the shortest path from s to every other node
- When we do the Dijkstra, we will stop a branch when we meet 2 nodes belongs to S.
- That will help us to make sure the shortest path between S and other node can have at most one node $\in S$
- We also use Hash Map(cuckoo) to store it. Key is name of node, value is distance between s and other node. there is also a variable in "value" indicate whether this path has one node $\in S$. The name of variable is "Has".
- We do the same thing for t.
- Finally, we just check the hashmap to find whether there is a node has two values in Both Hash Tables and only one hash table has Varible "Has == True"
- If there are multiple, we pick the smallest one.

(See Page 14

(use the rest of the space of rough work)

Problem 3

(c). Two Dijkstra takes $O(m+n\log n)$ by using Fib Heap

- Store element in HashMap takes $O(1)$, but we store $2n$ times. So it's $O(2n)=O(n)$
- These operations are paralled to each other, so takes $O(m+n\log n)$ in total

(use the rest of the space of rough work)

Problem 5

(a) Divide and conquer:

$2T(\frac{n}{2}) + O(n)$

↓

last 10 minutes find I need to use Divide and conquerer.

(use the rest of the space of rough work)