National University of Singapore

School of Computing

CS2040C - Data Structures and Algorithms

Final Assessment

(S2 AY2025/26)

Time Allowed: 2 hours

INSTRUCTIONS TO CANDIDATES:

1. Do **NOT** open this assessment paper until you are told to do so.
2. This assessment paper comprises SIXTEEN (16) printed pages, including this page.
3. This is an **Open Book Assessment**.
   You cannot use any electronic device except one calculator.
4. You can use either pen or pencil. Just make sure that you write **legibly**!
5. Important tips: Pace yourself! Do **not** spend too much time on one (hard) question.
   Read all the questions first! Some (sub-)questions might be easier than they appear.
   The first four questions have Generative AI answers that you can use/fix/improve (or rewrite).
6. If you choose to write your own answer and for the last two questions,
   you can use **pseudo-code** in your answer but beware of penalty marks for **ambiguous answer**.
   You can use **standard, non-modified** classic algorithm in your answer by just mentioning its name, e.g. run Inorder Traversal on BST T, DFS on graph G, Dijkstra's on graph G', etc.
7. The total marks is 60. All the best :)

A Application Questions ($6 \times 10 = 60$ marks)

A.1 Archery Practice (10 marks)

You want to practice your archery. The target is represented as the origin $(0, 0)$ on an infinite 2D plane. You shoot n arrows, each one landing at an integer coordinate (x, y). No two arrows land at the same coordinate.

The distance of an arrow at (x, y) from the origin is defined as $|x| + |y|$, i.e., the sum of the absolute values of x and y.

Starting from the k-th shot ($1 \leq k \leq n$), you record the distance of the k-th nearest arrow to the origin. You want to record your practice performance as follows: Calculate the total sum of these recorded distances from shot k through shot n.

Example: If $n = 4$, $k = 2$, and the arrows land at $(1, 1)$, $(3, 4)$, $(2, 1)$, and $(-1, 0)$ in that order:

- **Shot 1:** Lands at $(1, 1)$ with distance 2.
  This is not recorded as fewer than $k = 2$ arrows have been shot.
- **Shot 2:** There are now two arrows at distances 2 and 7.
  The 2nd nearest arrow has distance 7.
- **Shot 3:** There are now three arrows at distances $2, 3$, and 7.
  The 2nd nearest arrow has distance 3.
- **Shot 4:** There are now four arrows at distances $1, 2, 3$, and 7.
  The 2nd nearest arrow has distance 2.

For this example, we thus output $7 + 3 + 2 = 12$.

Solve this problem in C++ and analyze its time complexity in terms of n and/or k.

A.2 Integer Sibling Groups (10 marks)

You are given a set S containing n ($1 \leq n$) positive distinct integers, each less than 10^7. The integers are stored in an `std::set<int> S`. Two positive integers u and v are defined as **siblings** if they have the same sum of digits. A **sibling group** is a subset of S where all integers share the same digit sum. How many sibling groups are formed by the elements in S? See Figure 1 for an example.

Figure 1: An example $S = \{5, 6, 7, 23, 24, 33, 51, 123\}$ with $n = 8$ integers and the 3 sibling groups

Solve this problem in C++ and analyze its time complexity in terms of n.

A.3 VisuAlgo AVL Tree Visualization (10 marks)

Given the root of a Binary Search Tree (BST) T with n ($1 \leq n$) vertices (e.g., the vertex with value 37 in Figure 2), where each vertex value is a positive integer at most $(10^5 - 1)$ (suppose VisuAlgo can now show up to 5 digits without issue), return the root of an AVL Tree consisting of the same set of values (e.g., the vertex with value 46 in Figure 3). If there is more than one valid AVL Tree, you may return any of them.

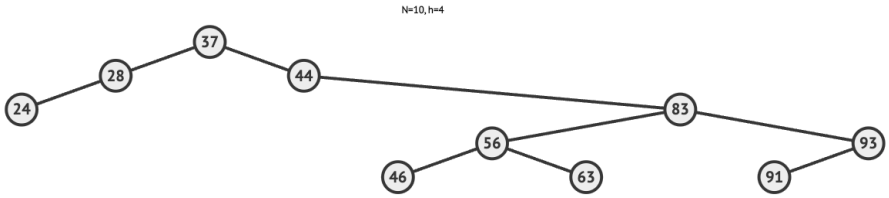


Figure 2: An example BST T with $n = 10$ that is not an AVL Tree (vertex 37 and 44 are unbalanced)

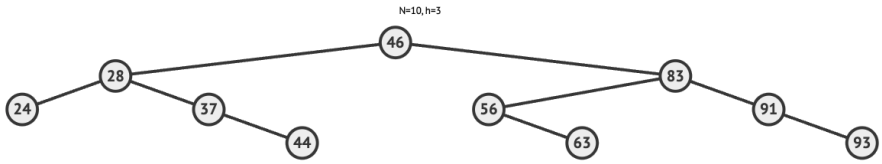


Figure 3: One possible AVL Tree equivalent to T

Recall that the AVL Tree property requires that the heights of the two subtrees of every vertex **differ by at most one**. If BST T is already an AVL Tree, you can choose to just return the root of T.

Solve this problem in C++ and analyze its time complexity in terms of n.

A.4 Strongest Trinity (10 marks)

You are given an **undirected simple** graph G with n vertices ($1 \leq n \leq 77$) and a list of m edges EL where each edge $EL[i] = \{u_i, v_i\}$ ($EL[i]$ is an `std::pair<int, int>`; $1 \leq u_i, v_i \leq n$; $u_i \neq v_i$) indicates that there is an undirected edge between u_i and v_i.

We introduce two new definitions:

- A **trinity** is a set of three vertices where an edge exists between every pair in the set.
- The **strength** of a trinity is the number of edges that have exactly one endpoint inside the trinity and the other endpoint outside the trinity.

Return the maximum strength among all trinities in G, or -1 if G contains no trinity.

Example 1: If $n = 7$ and $EL = [\{1,2\}, \{1,3\}, \{3,2\}, \{4,1\}, \{5,2\}, \{3,6\}, \{6,7\}]$, then the answer is `3`. The trinity is $\{1,2,3\}$. Its incident edges are $\{4,1\}, \{5,2\}$, and $\{3,6\}$.

Example 2: If $n = 7$ and $EL = [\{1,2\}, \{1,3\}, \{4,1\}, \{5,2\}, \{3,6\}, \{6,7\}]$, then the answer is `-1`. There is no trinity.

Solve this problem in C++ and analyze its time complexity in terms of n and/or m.

A.5 Yoshi's Tasks (10* marks)

Yoshi has n tasks to complete. These tasks are not independent; the execution of a task is only possible if all its precedent tasks have already been completed.

Yoshi provides two integers n ($1 \leq n$), the number of tasks (numbered from 1 to n) and m ($0 \leq m \leq n \cdot (n-1)$), the number of direct precedence relations between tasks. You are given m pairs of integers $\{i, j\}$ in an `std::vector<pair<int, int>> precedence`, where each pair denotes that task i must be completed before task j.

Help Yoshi by listing the n tasks in the lexicographic smallest order that allows them to be completed successfully by Yoshi. If it is not possible for Yoshi to complete all n tasks, output -1.

Example 1: If $n = 5$, $m = 4$, and `precedence` $= \{\{1,4\},\{4,2\},\{1,2\},\{1,5\}\}$, then Yoshi must do the $n = 5$ tasks in this order $1, 3, 4, 2, 5$.

Example 2: If $n = 5$, $m = 4$, and `precedence` $= \{\{1,4\},\{4,2\},\{2,1\},\{1,5\}\}$, then output -1.

This is a starred task ('you are out of messages until 3pm').
Design a correct $O((n+m)\log n)$ algorithm to solve this task for 10* marks.
(or leave it blank for 2 marks).
An incorrect or significantly inefficient solution will receive no marks.

A.6 The Last Question (10* marks)

You are given a connected, undirected, simple, non-negative weighted graph G with n vertices labeled 0 to $n-1$. There are m weighted edges provided in an edge list format (u, v, w), where each entry indicates a bidirectional edge between vertex u and vertex v with weight w.

Since G is connected, there is at least one (but possibly more) shortest path(s) from the source vertex 0 to the target vertex $n-1$ in G. Your task is to **count** how many unique edges of G are part of *at least one* such shortest path.

Example: In Figure 4, there are $n = 6$ vertices and $m = 7$ bidirectional edges. All edges except for bidirectional edge $(0, 3, 4)$ are part of at least one shortest path from 0 to 5. Therefore, the output for this case is 6.

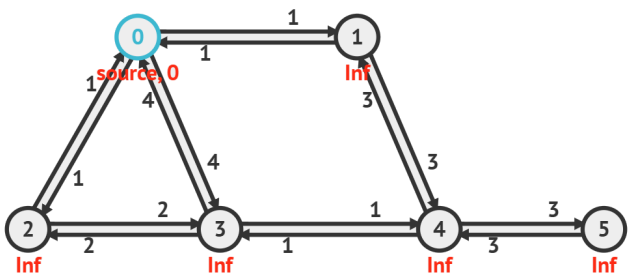
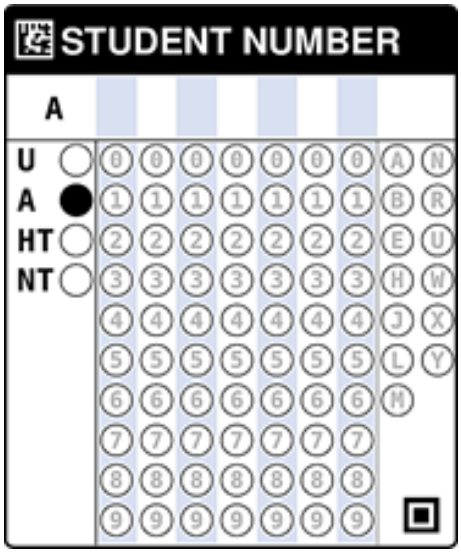


Figure 4: There are two shortest paths from 0 to 5, i.e., $0 \to 1 \to 4 \to 5$ and $0 \to 2 \to 3 \to 4 \to 5$

This is a starred task ('you are out of messages until 3pm').
Design a correct $O((n+m)\log n)$ algorithm to solve this task for 10* marks.
(or leave it blank for 2 marks).
An incorrect or significantly inefficient solution will receive no marks.

The Answer Sheet for Semester 2 AY2025/26

Write your Student Number in the box below using **(2B) pencil**. **Do NOT write your name**.



This portion is for examiner's use only

Section	Maximum Marks	Your Marks	Grading Remarks
Total	60		

Box A.1. Archery Practice

```
class SolutionShallow {
public:
    int archery(vector<pair<int, int>>& shots, int k) {
        set<int> S;
        long long ans = 0;
        for (const auto& [x, y] : shots) {
            int d = abs(x) + abs(y);
            S.insert(d);
            if ((int)S.size() >= k) {
                auto it = S.begin();
                for (int i = 1; i < k; ++i) ++it;
                ans += *it;
            }
        }
        return ans;
    }
};
```

My analysis for SolutionShallow is:

It uses the correct data structure that we learn in class, i.e., std::set ◯ YES | ◯ NO, I will elaborate

Ignoring time complexity, it gives correct output for all valid inputs: ◯ YES | ◯ NO, I will elaborate

Ignoring correctness, it runs in: ◯ $O(n \log n)$ | ◯ $O(n \cdot k)$ | ◯ $O(n^2)$ | ◯ Other, I will elaborate

It can crash on at least one valid corner case input: ◯ YES, I will elaborate | ◯ NO

I can improve upon the Generated AI code: ◯ YES, I will elaborate | ◯ NO, it is the best possible

Elaboration of my analysis above:

Only if you need to improve upon the Generated AI code.

My faster+correct code runs in: ◯ $O(\log n)$ | ◯ $O(n \log n)$ | ◯ $O(n \cdot k)$ | ◯ Other, as written above.

Box A.2. Integer Sibling Groups

```
// https://github.com/stevenhalim/cpbook-code/blob/master/ch2/ourown/unionfind_ds.cpp
class SolutionGrasp {
public:
    int countSiblingGroups(std::set<int>& S) {
        std::vector<int> nums(S.begin(), S.end());
        int n = nums.size();
        if (n == 0) return 0;

        UnionFind uf(n); // assume UnionFind is defined from that github repo above
        std::vector<int> digitSums(n);

        for (int i = 0; i < n; ++i) {
            digitSums[i] = getDigitSum(nums[i]);
        }

        for (int i = 0; i < n; ++i) {
            for (int j = i + 1; j < n; ++j) {
                if (digitSums[i] == digitSums[j]) {
                    uf.unionSet(i, j); // assume that this is O(1) per call
                }
            }
        }

        return uf.numDisjointSets(); // assume that this is O(1) per call
    }

private:
    int getDigitSum(int n) {
        int sum = 0;
        while (n > 0) {
            sum += n % 10;
            n /= 10;
        }
        return sum;
    }
};
```

My analysis of SolutionGrasp is:

It uses the correct data structure that we learn in class, i.e., UFDS ◯ YES | ◯ NO, I will elaborate

Ignoring time complexity, it gives correct output for all valid inputs: ◯ YES | ◯ NO, I will elaborate

Ignoring correctness, it runs in: ◯ $O(n)$ | ◯ $O(n \log n)$ | ◯ $O(n^2)$ | ◯ Other, I will elaborate

It can crash on at least one valid corner case input: ◯ YES, I will elaborate | ◯ NO

I can improve upon the Generated AI code: ◯ YES, I will elaborate | ◯ NO, it is the best possible

Elaboration of my analysis above:

Only if you need to improve upon the Generated AI code.
My faster+correct code runs in: ◯ $O(\log n)$ | ◯ $O(n)$ | ◯ $O(n \log n)$ | ◯ Other, as written above.

Box A.3. VisuAlgo AVL Tree Visualization

```
class SolutionTwin {
public:
    TreeNode* balanceBST(TreeNode* root) {
        std::vector<int> nodes;
        inorder(root, nodes);
        return build(nodes, 0, nodes.size() - 1);
    }
private:
    void inorder(TreeNode* root, std::vector<int>& nodes) {
        if (!root) return;
        inorder(root->left, nodes);
        nodes.push_back(root->val);
        inorder(root->right, nodes);
    }
    TreeNode* build(const std::vector<int>& nodes, int start, int end) {
        if (start > end) return nullptr;
        int mid = start + (end - start) / 2;
        TreeNode* node = new TreeNode(nodes[mid]);
        node->left = build(nodes, start, mid - 1);
        node->right = build(nodes, mid + 1, end);
        return node;
    }
};

class SolutionTalk {
public:
    vector<int> a;
    void inorder(TreeNode* root) {
        if (!root) return;
        inorder(root->left);
        a.push_back(root->val);
        inorder(root->right);
    }
    TreeNode* build(int l, int r) {
        if (l > r) return NULL;
        int m = (l + r) >> 1;
        TreeNode* root = new TreeNode(a[m]);
        root->left = build(l, m - 1);
        root->right = build(m + 1, r);
        return root;
    }
    TreeNode* balanceBST(TreeNode* root) {
        inorder(root);
        return build(0, (int)a.size() - 1);
    }
};
```

My analysis for SolutionTwin is:

Ignoring time complexity, it gives correct output for all valid inputs: ◯ YES | ◯ NO, I will elaborate

Ignoring correctness, it runs in: ◯ $O(n)$ | ◯ $O(n \log n)$ | ◯ $O(n^2)$ | ◯ Other, I will elaborate

It can crash on at least one valid corner case input: ◯ YES, I will elaborate | ◯ NO

My analysis for SolutionTalk is:

Ignoring time complexity, it gives correct output for all valid inputs: ◯ YES | ◯ NO, I will elaborate

Ignoring correctness, it runs in: ◯ $O(n)$ | ◯ $O(n \log n)$ | ◯ $O(n^2)$ | ◯ Other, I will elaborate

It can crash on at least one valid corner case input: ◯ YES, I will elaborate | ◯ NO

Thus, I will use: ◯ Either solution | ◯ SolutionTwin | ◯ SolutionTalk | ◯ My own solution below

Elaboration of my analysis above:

Only if you need to improve upon the Generated AI code.

My faster+correct code runs in: ◯ $O(1)$ | ◯ $O(\log n)$ | ◯ $O(n)$ | ◯ Other, as written above.

Box A.4. Strongest Trinity

```
class SolutionShannon {
    int maxTrinityStrength(int n, vector<pair<int, int>>& EL) {
        vector<vector<int>> AM(n+1, vector<int>(n+1, 0));

        for (auto &e : EL) {
            int u = e.first, v = e.second;
            AM[u][v] = AM[v][u] = 1;
        }

        int ans = -1;

        for (int i = 1; i <= n; i++) {
            for (int j = i+1; j <= n; j++) {
                for (int k = j+1; k <= n; k++) {
                    if (AM[i][j] && AM[j][k] && AM[i][k]) {
                        int strength = 0;

                        for (int x = 1; x <= n; x++) {
                            if (x == i || x == j || x == k) continue;

                            if (AM[i][x]) strength++;
                            if (AM[j][x]) strength++;
                            if (AM[k][x]) strength++;
                        }

                        ans = max(ans, strength);
                    }
                }
            }
        }

        return ans;
    }
}
```

My analysis for SolutionShannon is:

It uses the correct graph data structure (check space complexity): ◯ YES | ◯ NO, I will elaborate

Ignoring time complexity, it gives correct output for all valid inputs: ◯ YES | ◯ NO, I will elaborate

Ignoring correctness, it runs in: ◯ $O(n^2)$ | ◯ $O(n^3)$ | ◯ $O(n^4)$ | ◯ Other, I will elaborate

It can crash on at least one valid corner case input: ◯ YES, I will elaborate | ◯ NO

I can improve upon the Generated AI code: ◯ YES, I will elaborate | ◯ NO, it is the best possible

Elaboration of my analysis above:

Only if you need to improve upon the Generated AI code.
My faster+correct code runs in: ◯ $O(n^2)$ | ◯ $O(n^3)$ | ◯ $O(n^4)$ | ◯ Other, as written above.

Box A.5. Yoshi's Tasks (This is a STARRED task without Generative AI assistance)

◯ I give up, just give me the free 2 marks

◯ I know how to solve this, I will elaborate below (pseudocode is allowed and encouraged):

My solution runs in $O($_____$)$.

Box A.6. The Last Question (This is a STARRED task without Generative AI assistance)

◯ I give up, just give me the free 2 marks

◯ I know how to solve this, I will elaborate below (pseudocode is allowed and encouraged):

My solution runs in $O($_____$)$.

You can use this page as an extra answer sheet - if needed

– END OF PAPER; All the Best –