This homework assignment is also Sample Final.

Student Info

Your name and student id:

Name: , **ID No.**:

1 Answer Sheet

Please fill your answers in the answer sheet.

ANSWER SHEET FOR MULTIPLE CHOICE:

1	2	3	4	5	6
7	8	9	10	11	

ANSWER SHEET FOR WRITING QUESTIONS:

Solution: 3.1
(1)

(2)

(3)

Solution: 3.2

(i)

(ii)

(iii)

(iv)

(v)

(vi)

(vii)

Solution: 3.3
(1)

(2)

(3) 1.

2.

3.

4.

5.

Solution: 3.4
1.

2.

3.

4.

5.

6.

7.

8.

2 Multiple Choice

The following questions have only **ONE** correct answer. Each question is worth 2 points.

1. What is the time complexity of running the following function?

```
int function(int n) {
    if (n <= 1) {
        return 1;
    } else {
        int sum = 0;
        for (int i=0; i<n; i++) sum += i*i;
        return 3 * function(n / 3) + sum;
    }
}
```

   A) $\Theta(\log n)$

   B) $\Theta(n \log n)$

   C) $\Theta(n)$

   D) $\Theta(n^2)$

2. Let the length of the hash table be 11, the hash function be H(key)=key%11. Four elements have been added to the table: 15, 20, 38, 61. Now, the element with key word 49 is added to the table, and the conflict is resolved by **quadratic probing** method. Which position is the place to put in?

   A) 3

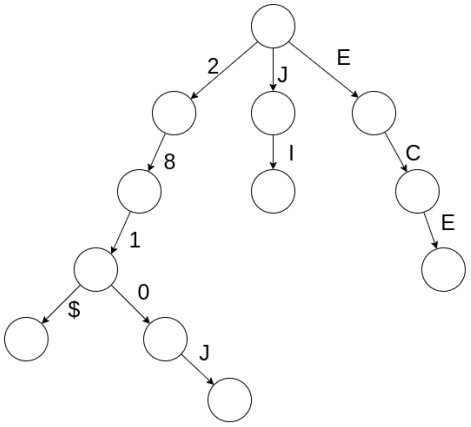
   B) 5

   C) 7

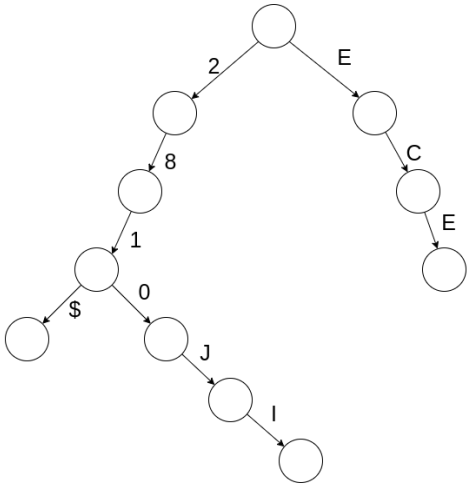
   D) 9

3. After inserting '281', '2810J', 'JI', and 'ECE' into an empty trie, what will the trie look like? For the simplicity, the choices ignore the leaf node.

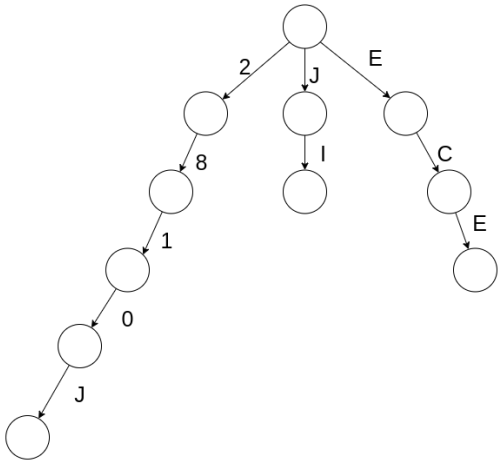
A)



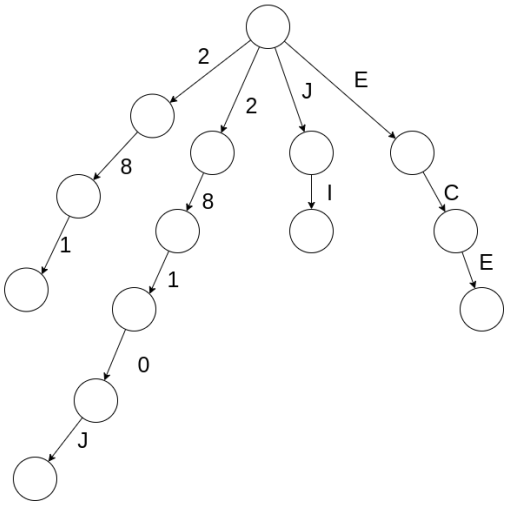
B)



C)



D)



4. Which of the following statements are true?

   (a) Represent binary tree using array. Assume the index of the root is 1. Then for any node with index i ($i \neq 1$), its parent has node index $\lfloor i/2 \rfloor$.

   (b) Prim algorithm is a greedy algorithm.

   (c) If using the same graph representation method, BFS and DFS do not have the same time complexity.

   A) (a)(b)

   B) (a)(c)

   C) (b)(c)

   D) (a)(b)(c)

5. If you use a Fibonacci heap to optimize the Prim's algorithm, what's the **total** time complexity for finding and extracting the smallest $D(v)$?

   A) $\Theta(\log |V|)$

   B) $\Theta(|V| \log |V|)$

   C) $\Theta(|E| \log |V|)$

   D) $\Theta(|E|)$

6. The Tower of Hanoi is a puzzle consisting of three rods and a number of disks of various diameters, which can slide onto any rod. The puzzle begins with the disks stacked on one rod in order of decreasing size, the smallest at the top, thus approximating a conical shape. The objective of the puzzle is to move the entire stack to one of the other rods, obeying the following rules:

   (a) Only one disk may be moved at a time.

   (b) Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack or on an empty rod.

   (c) No larger disk may be placed on top of a smaller disk.

   Assume that we have 2023 disks. What is the minimum number of moves required to solve the puzzle?

   A) $2^{2023} - 1$

   B) 2^{2023}

   C) $2^{2023} + 1$

   D) 2^{2024}

7. Given a directed and weighted graph and a source node s, let $f(x)$ to be the shortest distance from s to x. If an edge from m to n has weight 2, which one of the following cannot be a possible value of $f(n) - f(m)$?

   A) -3

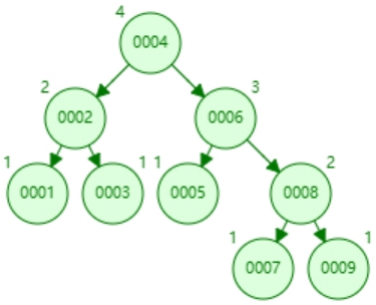
   B) 0

   C) 2

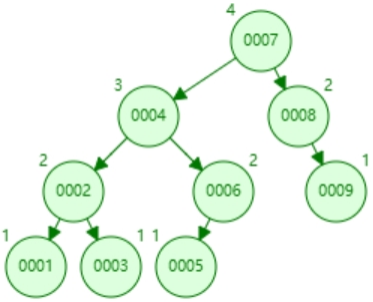
   D) 3

8. Given the sequence of keys $[1, 2, 3, 4, 5, 6, 7, 8, 9]$, draw the corresponding AVL tree after inserting the keys in the order given.

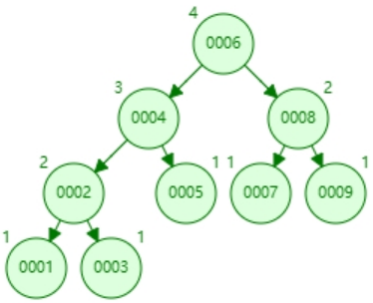
A)



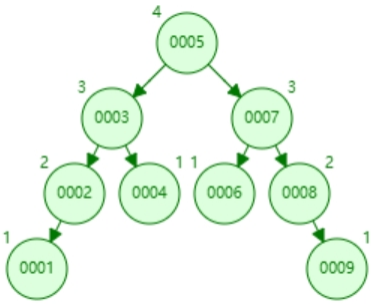
B)



C)

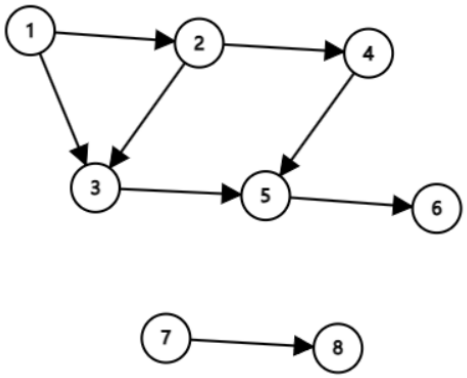


D)



The following questions have MULTIPLE correct answers. Each question is worth 4 points.

9. Consider the following DAG, which of the following sequences can be the possible topological order of the graph?



A) 1,2,3,4,5,6,7,8

B) 7,8,1,2,3,4,5,6

C) 7,8,1,2,4,3,6,5

D) 1,7,3,2,4,5,6,8

E) 1,2,3,7,4,5,8,6

10. Which of the following operation might change the height of an AVL Tree?

    A) Insertion of an element that doesn't trigger any rotation.

    B) Insertion of an element that triggers one rotation (LL or RR).

    C) Insertion of an element that triggers two rotations (LR or RL).

    D) Deletion of an element that triggers one rotation.

    E) Deletion of an element that triggers multiple rotations.

11. Which of the following statements about the Binary Search Tree (BST) are correct?

    A) Pre-order traversal of a BST produces a sorted sequence of elements.

    B) The time complexity of searching for an element in a balanced BST is $\Theta(\log n)$, where n is the number of nodes.

    C) A BST can be constructed from a given array of elements in linear time.

    D) The height of a BST with n nodes should be $\Theta(\log n)$.

    E) Deleting a node with two children in a BST requires promoting the successor or predecessor node.

3 Writing Questions

3.1 Warm Up (15 points)

i) What is the average time complexity of Merge Sort and Insertion Sort? (3 points) Whether are they stable? (2 points)

ii) In principle, the time complexity of Selection Algorithms should be ____ ($\leq$ or $\geq$) the time complexity of Sorting Algorithms. (2 points) Why? (3 points)

iii) Draw the Binary Search Tree based on the insertion sequence "8, 3, 1, 6, 10, 4, 7". (3 points) If a proper binary tree has n 2-degree nodes, what is the number of its leaf nodes? (2 points)

3.2 Fix K-D Tree (14 points)

You must be very familiar with k-d trees as you have passed the trial of Project 3. The pseudocode below is directly copied from Project 3, illustrating the deletion operation of a k-d tree.

Function `Delete`(*node, key, depth*)
  if *node is null* **then**
    return *null*;
  end
  dimension $\leftarrow$ *depth* mod k;
  if *key* = *node.key* **then**
    if *node is a leaf* **then**
      delete *node* directly;
      return *null*;
    else if *node has right subtree* **then**
      minNode $\leftarrow$ the **minimum** node on *dimension* in the **right** subtree *node.right*;
      node.key $\leftarrow$ *minNode.key*;
      node.value $\leftarrow$ *minNode.value*;
      node.right $\leftarrow$ `Delete`(*node.right, key, depth* + 1);
    else if *node has left subtree* **then**
      maxNode $\leftarrow$ the **maximum** node on *dimension* in the **left** subtree *node.left*;
      node.key $\leftarrow$ *maxNode.key*;
      node.value $\leftarrow$ *maxNode.value*;
      node.left $\leftarrow$ `Delete`(*node.left, key, depth* + 1);
    end
  else
    if *key* < *node.key on dimension* **then**
      node.left $\leftarrow$ `Delete`(*node.left, key, depth* + 1);
    else
      node.right $\leftarrow$ `Delete`(*node.right, key, depth* + 1);
    end
  end
  return *node*;
end

Algorithm 1: Deletion of node.

The following code, copied from slides, illustrates the insertion operation of a k-d tree.

```
void insert(node *&root, Item item, int dim) {
    if(root == NULL) {
        root = new node(item);
        return;
    }
    if(item.key == root->item.key) // equal in all
        return;                    // dimensions
    if(item.key[dim] < root->item.key[dim])
        insert(root->left, item, (dim+1)%numDim);
    else
        insert(root->right, item, (dim+1)%numDim);
}
```

For the simplicity, we omit the values of nodes, and only consider the keys.

i) Suppose you already have a 2-d tree shown in Fig. 1. Now please insert (0,1) and (0,5) into the tree, and draw the tree after the insertion operation. Notice that you should strictly follow the code provided. (2 points)

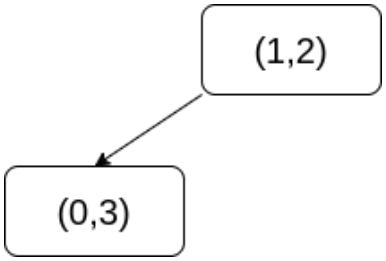


Figure 1: Initial K-D Tree

ii) Based on the results from part i), delete the root node (1,2) and draw the tree after the deletion operation. Notice that you should strictly follow the pseudocode provided.(2 points).

Hint: In the deletion operation, we need to find the **maximum** or **minimum** node on *dimension*. In cases where two nodes share identical keys within that *dimension*, a secondary comparison is conducted along the other dimension. To illustrate, consider the scenario where (1,4) represents the **maximum** node on the **first** dimension when compared to other nodes such as (1,2) and (1,0). Similarly, (1,0) stands as the **minimum** node on the **second** dimension when compared among nodes like (5,0), (1,0), and (9,0).

iii) Based on the results from part ii), insert (0,1) again and draw the tree after the insertion operation. Notice that you should strictly follow the code provided. (2 points).

iv) The tree seems a little bit strange now. Observe the tree, you will find one invariant of the k-d tree is violated. What is the violated invariant specifically? You should think about the relationship between one node and the nodes in its **subtree**. (2 points)

v) ____ (insertion or deletion) operation ruins the invariant. (2 points).

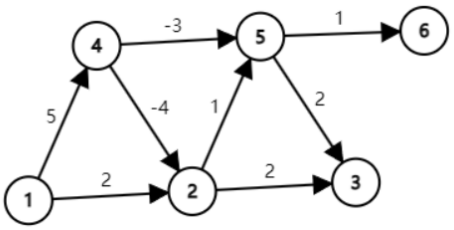
vi) Explain how does the operation you chose in part v) violate the invariant. (1 points).

vii) How will you modify the operation to maintain the invariant? Notice that you will not get full points if your method has a bad time complexity. (3 points)

3.3 Graph! Graph! Graph!(15 points)

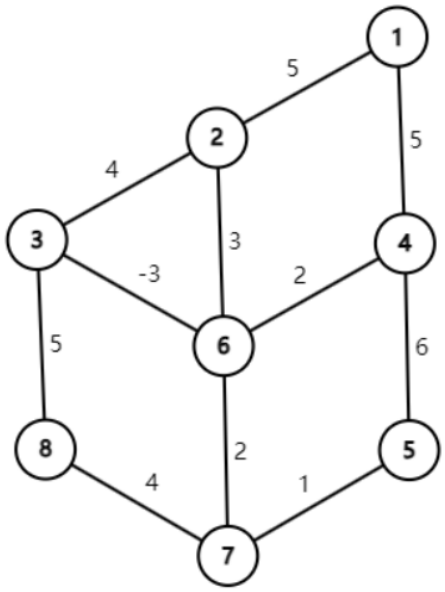
3.3.1 Dijkstra(5 points)

You are given a directed graph G shown below, 1 is the source node and you want to calculate the shortest distance from 1 to all other nodes x, which is denoted as $d(x)$. Although you know it's wrong to use Dijkstra's algorithm here, but you still use it. Write the actual $d(x)$ and the $d(x)$ you get by Dijkstra's algorithm for each node x.



3.3.2 Prim(5 points)

You are given an un-directed graph G shown below, apply Prim's algorithm to find the minimum spanning tree of the graph G. Illustrate step by step to get partial points.



3.3.3 Save Blue Tiger!(5 points)

During an expedition on an island, Mr. Blue Tiger found him lost. He desired to know the shortest distance to another island, so he asked you for help. You are given a matrix indicating a 2-D map, where 1 represents lands and 0 represents water. An island is a 4-directionally connected group of 1's not connected to any other 1's. There are exactly two islands in the grid.
Please complete the following code which uses BFS to find the shortest distance. Two examples are provided in the next page. Notice that the distance of a path is the number of 0's on this path.

```
int shortestBridge(vector<vector<int>>& grid) {
    int Dirs[4][2] = { { 0 , -1 } , { 0 , 1 } , { 1 , 0 } , { -1 , 0 } };
    int m = grid.size() , n = grid[0].size();
    queue<pair<int,int>> Q;
    vector<pair<int,int>> island1;
    for(int i = 0;i < m;++i)
        for(int j = 0;j < n;++j) if( grid[i][j] == 1 ){
            grid[i][j] = -1;
            Q.emplace(i,j); // same to Q.push(make_pair(i,j))
```

```
            while(!Q.empty()) {
                auto [x,y] = Q.front();
                Q.pop();
                ___1___ // TODO
                for(int k = 0;k < 4;++k) {
                    int nx = x + Dirs[k][0];
                    int ny = y + Dirs[k][1];
                    if( nx >= 0 && ny >= 0 && nx < m && ny < n
                        && grid[nx][ny] == 1 ) {
                        Q.emplace(nx,ny);
                        grid[nx][ny] = -1;
                    }
                }
            }
            // Find all lands for this island, then BFS to find another island]
            for(auto &&[x,y]:island1)
                ___2___ //TODO
            int distance = 0;
            while(!Q.empty()) {
                int size = Q.size();
                for(int i = 0; i < size; ++i) {
                    auto [x,y] = Q.front();
                    Q.pop();
                    for(int k = 0;k < 4;++k) {
                        int nx = x + Dirs[k][0];
                        int ny = y + Dirs[k][1];
                        if( nx >= 0 && ny >= 0 && nx < m && ny < n ){
                            if( grid[nx][ny] == 0 ) {
                                Q.emplace(nx,ny);
                                ___3___ // TODO
                            } else if( grid[nx][ny] == 1 )
                                ___4___ // TODO
                        }
                    }
                }
                ___5___ //TODO
            }
        }
    return 0;
}
// sample input: 1000111
//               1100011
//               1111001
//               0110001
// sample output: 2
//
// sample input: 1000001
//               1100001
//               1110001
//               1100001
//               1000111
// sample output: 3
```

3.4 Coin Change (15 points)

You are given an integer array `coins` representing coins of different denominations and an integer `amount` representing a total amount of money.

Return *the fewest number of coins that you need to make up that amount.* If that amount of money cannot be made up by any combination of the coins, return `-1`.

You may assume that you have an infinite number of each kind of coin.

Example 1:

```
Input: coins = [1,2,5], amount = 11
Output: 3
Explanation: 11 = 5 + 5 + 1
```

Example 2:

```
Input: coins = [2], amount = 3
Output: -1
```

Example 3:

```
Input: coins = [1], amount = 0
Output: 0
```

A simple solution is to consider every possible subset of coins and calculate the total number of coins needed for each subset. Finally, return the minimum of all the values.

```cpp
int coinChange(vector<int>& coins, int amount) {
    // base case
    if (amount == 0) return 0;
    // initialize the result
    int res = INT_MAX;
    // consider every possible subset of coins
    for (int i = 0; i < coins.size(); i++) {
        // if the coin value is less than or equal to the amount
        if (coins[i] <= amount) {
            // calculate the result for the subset
            int sub_res = coinChange(coins, amount - coins[i]);
            // if the result is not INT_MAX and the result + 1
            // is less than the current result
            if (sub_res != INT_MAX && sub_res + 1 < res) {
                // update the result
                res = sub_res + 1;
            }
        }
    }
    // return the result
    return res;
}
```

The time complexity of this solution is $O(S^n)$.

This solution is not efficient enough. We can optimize this solution using ___**1**___.
Assume $F(S)$ is the minimum number of coins needed to make change for amount S. And C is the last coin in the optimal solution. Then we have the following relation:

$$F(S) = \text{___}\mathbf{2}\text{___}$$

However, we don't know what C is. So we need to try every possible coin and choose the one that leads to the minimum number of coins. In above recursive solution, many subproblems are solved again and again. We can avoid this by storing the results of subproblems.

Please complete the following code to implement the solution.

```
class Solution {
    vector<int>count;
    int dp(vector<int>& coins, int rem) {
        if (rem < 0) return -1;
        if (rem == 0) return 0;
        if (count[rem - 1] != 0) return ___3___; // TODO
        int Min = INT_MAX;
        for (int coin:coins) {
            int res = ___4___; // TODO
            if (res >= 0 && res < Min) {
                Min = res + 1;
            }
        }
        count[rem - 1] = ( ( ___5___ ) ? -1 : Min ); // TODO
        return count[rem - 1];
    }
public:
    int coinChange(vector<int>& coins, int amount) {
        if (amount < 1) return 0;
        count.resize(amount);
        return dp(coins, amount);
    }

};
```

The time complexity of this solution is ___**6**___, where S is the amount, n is count size.
There is also another solution.

```
class Solution {
public:
    int coinChange(vector<int>& coins, int amount) {
        int Max = amount + 1;
        vector<int> dp(amount + 1, Max);
        dp[0] = 0;
        for (int i = 1; i <= amount; ++i) {
            for (int j = 0; j < (int)coins.size(); ++j) {
                if (coins[j] <= i) {
                    dp[i] = min(dp[i], ___7___); // TODO
                }
            }
        }
        return dp[amount] > amount ? -1 : dp[amount];
    }
};
```

This solution is a bottom-up approach. The time complexity of this solution is ___**8**___, where S is the amount, n is count size.