DOMAIN WINTER CAMP

(Department of Computer Science and Engineering)

Name: Jay Ananya   UID: 22BCS10822   Section: 22BCS-KPIT 901

DAY 2

Ques 1. You are given an integer array jobs, where jobs[i] is the amount of time it takes to complete the ith job.There are k workers that you can assign jobs to. Each job should be assigned to exactly one worker. The working time of a worker is the sum of the time it takes to complete all jobs assigned to them. Your goal is to devise an optimal assignment such that the maximum working time of any worker is minimized.

Return the minimum possible maximum working time of any assignment.

Program code:

```cpp
#include <iostream>

#include <vector>

#include <algorithm>

#include <numeric> using

namespace std;

bool canAssign(const vector<int>& jobs, vector<int>& workers, int maxWorkingTime, int index) {

    if (index == jobs.size()) {

return true;

    }

for (int i = 0;
```

```cpp
i < workers.size(); ++i) {
 if (workers[i] + jobs[index] <= maxWorkingTime) {
        workers[i] += jobs[index];
if (canAssign(jobs, workers, maxWorkingTime, index + 1)) {
            return true;

        }

        workers[i] -= jobs[index];

     }

     if (workers[i] == 0) {

        break;

     }

}

    return false;

}
int minimumTimeRequired(vector<int>& jobs, int k) {
int left = *max_element(jobs.begin(), jobs.end());
int right = accumulate(jobs.begin(), jobs.end(), 0);

    sort(jobs.rbegin(), jobs.rend());
while (left < right) {
 int mid = left + (right - left) / 2;
vector<int> workers(k, 0);
if (canAssign(jobs, workers, mid, 0)) {
        right = mid;
} else
{
left = mid + 1;
     }
```
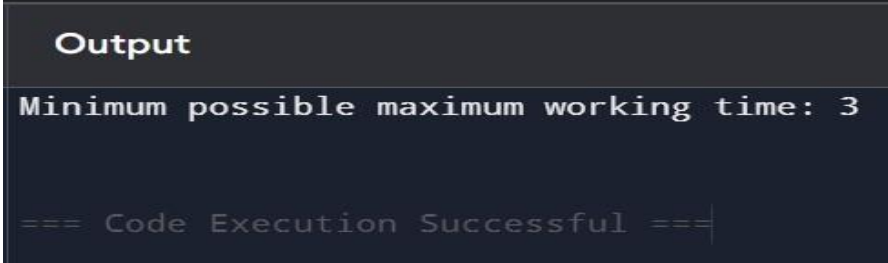
```
}
    return left;
} int main() {
vector<int> jobs = {3, 2, 3};
    int k = 3;
    cout << "Minimum possible maximum working time: " <<
minimumTimeRequired(jobs, k) << endl;
    return 0;
}
```

Output:


```
Output
Minimum possible maximum working time: 3


=== Code Execution Successful ===
```

Ques 2. On a social network consisting of m users and some friendships between users, two users can communicate with each other if they know a common language.

You are given an integer n, an array languages, and an array friendships where:

There are n languages numbered 1 through n, languages[i] is the set of

languages the ith user knows, and friendships[i] = [ui, vi] denotes a

friendship between the users ui and vi.

You can choose one language and teach it to some users so that all friends can communicate with each other. Return the minimum number of users you need to teach.

Note that friendships are not transitive, meaning if x is a friend of y and y is a friend of z, this doesn't guarantee that x is a friend of z.

Program code:

```cpp
#include <iostream>
#include <vector>
#include <unordered_set>
#include <unordered_map>
#include <algorithm> using
namespace std;
int minNumberOfUsersToTeach(int n, vector<vector<int>>& languages,
vector<vector<int>>& friendships) {
unordered_map<int, unordered_set<int>> userLanguages;
    for (int i = 0; i < languages.size(); ++i) {
for (int lang : languages[i]) {
userLanguages[i + 1].insert(lang);
    }
  }
unordered_set<int> toTeachUsers;
vector<pair<int, int>> toTeachPairs;
for (const auto& friendship : friendships) {
int u = friendship[0], v = friendship[1];
bool canCommunicate = false;
for (int lang : userLanguages[u]) {
if (userLanguages[v].count(lang)) {
canCommunicate = true;
        break;
      }
    }
```

```cpp
        if (!canCommunicate) {
toTeachPairs.emplace_back(u, v);
toTeachUsers.insert(u);
 toTeachUsers.insert(v);
        }
    }

    if (toTeachPairs.empty()) {
        return 0;
    }

    int minTeach = toTeachUsers.size();
for (int lang = 1; lang <= n; ++lang) {
unordered_set<int> teachSet;
for (const auto& pair : toTeachPairs) {
int u = pair.first, v = pair.second;
if (!userLanguages[u].count(lang)) {
            teachSet.insert(u);
        }
        if (!userLanguages[v].count(lang)) {
            teachSet.insert(v);
        }
    }
    minTeach = min(minTeach, (int)teachSet.size());
}

    return minTeach;
}
```

```cpp
int main() {
int n = 2;
    vector<vector<int>> languages = {{1}, {2}, {1, 2}};
vector<vector<int>> friendships = {{1, 2}, {1, 3}, {2, 3}};


    cout << "Minimum number of users to teach: " <<
minNumberOfUsersToTeach(n, languages, friendships) << endl;


    return 0;
}
```
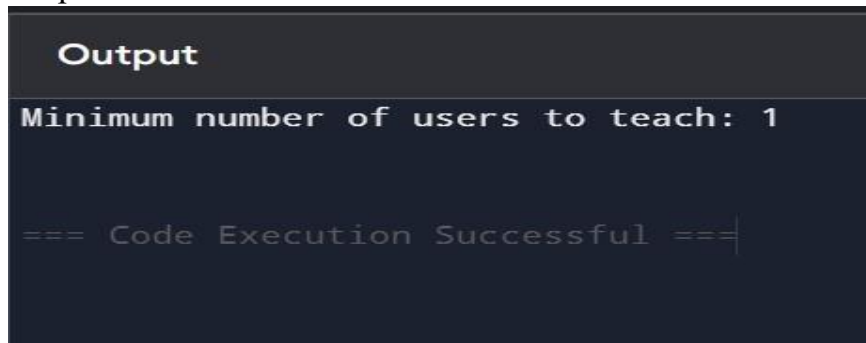
Output:

```
Output

Minimum number of users to teach: 1


=== Code Execution Successful ===
```

Ques 3. Determine if a 9 x 9 Sudoku board is valid. Only the filled cells need to be validated according to the following rules:

Each row must contain the digits 1-9 without repetition.

Each column must contain the digits 1-9 without repetition.

Each of the nine 3 x 3 sub-boxes of the grid must contain the digits 1-9 without repetition.

Note:

A Sudoku board (partially filled) could be valid but is not necessarily solvable.

Only the filled cells need to be validated according to the mentioned rules.

Program Code:

```cpp
#include <iostream>
#include <vector>
#include <unordered_set>

using namespace std;

bool isValidSudoku(vector<vector<char>>& board) {
vector<unordered_set<char>> rows(9);
vector<unordered_set<char>> cols(9);
vector<unordered_set<char>> boxes(9);

    for (int i = 0; i < 9; ++i) {
for (int j = 0; j < 9; ++j) {
char num = board[i][j];
        if (num != '.') {
 // Check row
if (rows[i].count(num)) {
return false;
        }
        rows[i].insert(num);
if (cols[j].count(num)) {
            return false;
        }
        cols[j].insert(num);
```

```cpp
            int boxIndex = (i / 3) * 3 + (j / 3);
            if (boxes[boxIndex].count(num)) {
                return false;
            }
            boxes[boxIndex].insert(num);
        }
    }

    return true;
}

int main() {
vector<vector<char>> board = {
    {'5', '3', '.', '.', '7', '.', '.', '.', '.'},
    {'6', '.', '.', '1', '9', '5', '.', '.', '.'},
    {'.', '9', '8', '.', '.', '.', '.', '6', '.'},
{'8', '.', '.', '.', '6', '.', '.', '.', '3'},
    {'4', '.', '.', '8', '.', '3', '.', '.', '1'},
    {'7', '.', '.', '.', '2', '.', '.', '.', '6'},
{'.', '6', '.', '.', '.', '.', '2', '8', '.'},
    {'.', '.', '.', '4', '1', '9', '.', '.', '5'},
    {'.', '.', '.', '.', '8', '.', '.', '7', '9'}
    };

    cout << (isValidSudoku(board) ? "Valid Sudoku" : "Invalid Sudoku") << endl;
    return 0;
}
```

Output:

Ques 4. You are given an integer array height of length n. There are n vertical lines drawn such that the two endpoints of the ith line are (i, 0) and (i, height[i]).

Find two lines that together with the x-axis form a container, such that the container contains the most water.

Return the maximum amount of water a container can store.

Notice that you may not slant the container.

Program Code:

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std; int
maxArea(vector<int>& height) {

    int left = 0;    int right = height.size() - 1;
int max_area = 0;    while (left < right) {
int h = min(height[left], height[right]);
```
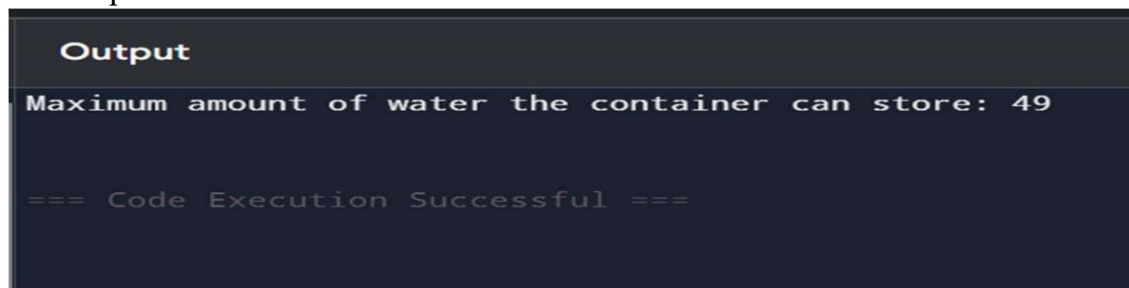
```cpp
    int w = right - left;

int current_area = h * w;

max_area = max(max_area, current_area);

        if (height[left] < height[right]) {

            ++left;

        } else {

            --right;

        }

    }

    return max_area;

} int main() {    vector<int> height = {1, 8, 6, 2,

5, 4, 8, 3, 7};

    cout << "Maximum amount of water the container can store: " <<
maxArea(height) << endl;

    return 0;

}
```

Output:

```
Output

Maximum  amount  of  water  the  container  can  store:  49


===  Code  Execution  Successful  ===
```

Ques 5. Given an integer numRows, return the first numRows of Pascal's triangle.

In Pascal's triangle, each number is the sum of the two numbers directly above it as shown:

Example 1:

Input: numRows = 5

Output: [[1],[1,1],[1,2,1],[1,3,3,1],[1,4,6,4,1]]

Program Code:

```cpp
#include <iostream>
#include <vector>
using namespace std;
vector<vector<int>> generate(int numRows) {
    vector<vector<int>> triangle;
    if (numRows <= 0) {
        return triangle;
    }
    triangle.push_back({1});
    for (int i = 1; i < numRows; ++i) {
        vector<int> row(i + 1, 1);
        for (int j = 1; j < i; ++j) {
            row[j] = triangle[i - 1][j - 1] + triangle[i - 1][j];
        }
        triangle.push_back(row);
    }
    return triangle;
}
int main() {
    int numRows = 5;
    vector<vector<int>> result = generate(numRows);
    for (const auto& row : result) {
        for (int val : row) {
            cout << val << " ";
        }
        cout << endl;
```

```
    }
    return 0;
}
```

Output:

```
Output

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1


=== Code Execution Successful ===
```