Name: **GAURAV YADAV**

Roll No**: 21075034**

Dept: **CSE (B.TECH)**

E-mail: gaurav.yadav.cse21@itbhu.ac.in

LAB ASSIGNMENT - 7

Q1) You have been given n candies where each candy i has a calorific value represented by candies [i]. You found a way to decrease the calorific value by combining the candies in some specific pattern. Suppose you combine the candy i and candy j, then the result of this combination would be a single candy with the resultant calorific value of |candies[i]-candies[j]|. At the end, there will be just 1 candy left. Return the smallest value that is possible for the last remaining candy.

problem1.cpp > main()

```cpp
#include <bits/stdc++.h>
#define ll long long
using namespace std;

int main()
{
    ll n;
    cin >> n;

    priority_queue<int> pq;
    for(int i=0;i<n;i++){
        int x;
        cin>>x;
        pq.push(x);
    }
    while(pq.size()>1){
        int a=pq.top();
        pq.pop();
        int b=pq.top();
        pq.pop();
        pq.push(abs(a-b));
    }
    cout<<pq.top();

    return 0;
}
```

Gaurav Yadav
21075034

Output:

```
PS C:\Users\Gaurav\Programming\practice> g++ problem1.cpp -o code
PS C:\Users\Gaurav\Programming\practice> ./code
6
5 12 8 9 3 7
0
PS C:\Users\Gaurav\Programming\practice> ./code
6
1 12 8 5 3 9
2
PS C:\Users\Gaurav\Programming\practice>
```

Time Complexity: we are adding n elements one by one which will take O(nlogn) time.
So time complexity will depend on O(nlogn).

Q2) You are given an array prices where prices[i] is the price of a given stock on the ith day. Find the maximum profit you can achieve. You may complete at most two transactions. Note: You may not engage in multiple transactions simultaneously (i.e., you must sell the stock before you buy again)

problem2.cpp > ...

```cpp
#include <bits/stdc++.h>
using namespace std;
int maxProfit(vector<int>& prices) {
        int dp[prices.size() + 1][5];
        for(int day = (int) prices.size();day >= 0;day--){
            for(int transactionsLeft = 0;transactionsLeft <= 4;transactionsLeft++){
                int &ans = dp[day][transactionsLeft];
                if(day == prices.size()){
                    ans = 0;
                }else if(transactionsLeft == 0){
                    ans = 0;
                }else{
                    // choice 1
                    // no transaction today
                    int ans1 = dp[day + 1][transactionsLeft];
                    // choice 2
                    // doing the possible transaction today
                    int ans2 = 0;
                    bool buy = (transactionsLeft % 2 == 0);
                    if(buy == true){ // buy
                        ans2 = -prices[day] + dp[day + 1][transactionsLeft - 1];
                    }else{ // sell
                        ans2 = prices[day] + dp[day + 1][transactionsLeft - 1];
                    }
                    ans = max(ans1, ans2);

                }

            }
        }
        return dp[0][4];
    }
```

```cpp
int main(){
    int t=1;
   // cin >> t;
    while(t--){
        int n;
        cin >> n;
        vector<int>v(n);
        for(int i=0 ; i<n ; i++){
            cin >> v[i];
        }
        int ans = maxProfit(v);
        cout << ans << "\n";
    }
}

```

Output:

```
PS C:\Users\Gaurav\Programming\practice> g++ problem2.cpp -o code
PS C:\Users\Gaurav\Programming\practice> ./code
8
3 3 5 0 0 3 1 4
6
PS C:\Users\Gaurav\Programming\practice> ./code
5
1 2 3 4 5
4
PS C:\Users\Gaurav\Programming\practice>
```

Q3) In a fictional video game, the player controls an ambulance that has a patient inside and must transport the patient to a hospital as quickly as possible. A m x n 2-D grid represents the city streets, with the hospital located in the bottom-right corner and the ambulance starting in the top-left corner. Some locations on the grid represent areas with heavy traffic, causing the ambulance to move more slowly and delaying the patient's arrival at the hospital. These locations are represented by negative integers. Other locations represent areas with clear roads, allowing the ambulance to move easily and quickly. These locations are represented by 0. In some locations, marked by positive integers, the patient receives medicinal help, which increases their health. The patient has an initial health level that changes as the ambulance moves on the streets. If the patient's health drops to zero or below before reaching the hospital, the patient dies, and the game will end. The player's goal is to determine the minimum initial health level required by the patient to reach the hospital safely. The layout of traffic and clear roads in the city is random, so the player must adjust the initial health level of the patient according to the specific layout of the city to successfully transport the patient to the hospital.

problem3.cpp > calculateMinimumHP(vector<vector<int>>&)

```cpp
#include<bits/stdc++.h>
using namespace std;
int calculateMinimumHP(vector<vector<int>>& street) {
        int n = street.size();
        int m = street[0].size();
        int dp[n][m];
        memset(dp,0,sizeof(dp));
        for(int i=n-1; i>=0; i--){
            for(int j=m-1;j>=0;j-- ){
                if(i==n-1 and j==m-1)
                    dp[i][j] = street[i][j];
                else if(i==n-1)
                    dp[i][j] = street[i][j] + dp[i][j+1];
                else if( j==m-1)
                    dp[i][j] = street[i][j] + dp[i+1][j];
                else
                    dp[i][j] = street[i][j] + max(dp[i+1][j], dp[i][j+1]);
                //to make positive values zero
                if(dp[i][j]>0)
                        dp[i][j] = 0;
            }
        }
        if(dp[0][0]>=0)
            return 1;
        else
            return (abs(dp[0][0])+1);
    }
int main(){
    int m,n;
    cin >> m >> n;
    vector<vector<int>>arr(m+1,vector<int>(n+1));
    for(int i=0 ; i<m ; i++){
        for(int j=0 ; j<n ; j++){
            cin >> arr[i][j];
        }
    }
    cout << calculateMinimumHP(arr) << "\n";
}
```

Output:

```
PS C:\Users\Gaurav\Programming\practice> g++ problem3.cpp -o code
PS C:\Users\Gaurav\Programming\practice> ./code
3 3
-3 9 1
-5 -9 0
12 -4 -2
4
PS C:\Users\Gaurav\Programming\practice>
```

Q4) Suppose you are given an array of integers arr and a positive integer k. Define a subarray of arr as contiguous elements of arr. Define the value of a subarray as the sum of its elements.

Your task is to find the maximum value of a subarray such that the subarray has exactly k distinct elements.

Write a dynamic programming algorithm to solve this problem in O(nk) time complexity.

Gaurav Yadav 21075034

problem4.cpp > main()

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){
    int t;
    cin >> t;
    while(t--){
        int n;
        cin >> n;
        int k;
        cin >> k;
        vector<int>v(n);
        for(int i=0 ; i<n ;i++){
            cin >> v[i];
        }
        vector<int>dp(n+1,INT_MIN);
        for(int i=k-1 ; i<n ; i++){
            int sum=0;
            set<int>st;
            for(int j=k-1 ; j>=0 ; j--){
                sum+=v[i-j];
                st.insert(v[i+j]);
            }
            //cout << sum << "\n";
            if(st.size()==k)
            dp[i]=max(dp[max(i-1,0)],sum);
        }
        cout << dp[n-1] << "\n";
    }
}

```

Output:

```
PS C:\Users\Gaurav\Programming\practice> g++ problem4.cpp -o code
PS C:\Users\Gaurav\Programming\practice> ./code
1
6
3
5 3 5 2 3 2
10
PS C:\Users\Gaurav\Programming\practice>
```

Gaurav Yadav 21075034

Q5) A frog is crossing a river. The river is divided into some number of units, and at each unit, there may or may not exist a stone. The frog can jump on a stone, but it must not jump into the water. Given a list of stones' positions (in units) in sorted ascending order, determine if the frog can cross the river by landing on the last stone. Initially, the frog is on the first stone and assumes the first jump must be 1 unit. if the frog's last jump was k units, its next jump must be either k - 1, k, or k + 1 units. The frog can only jump in the forward direction.

problem5.cpp > ...

```cpp
#include <bits/stdc++.h>
using namespace std;
long long dp[2001][2005];
bool f(vector<int>& nums,int i ,int jump)
{
    if(i==nums.size())return false;
    if(i==nums.size()-1)return true;
    if(dp[i][jump]!=-1)return dp[i][jump];
    bool x=false,y=false,z=false;
    for(int j=i+1;j<nums.size();j++)
    {
        if(nums[j]-nums[i]==jump)
        {
            x|=f(nums,j,jump);
        }
        if(nums[j]-nums[i]==jump+1)
        {
            y|=f(nums,j,jump+1);
        }
        if(nums[j]-nums[i]==jump-1)
        {
            z|=f(nums,j,jump-1);
        }
    }
    return  dp[i][jump] =  x||y||z;

}
bool canCross(vector<int>& nums) {
    memset(dp,-1,sizeof(dp));
    return f(nums,0,0);
}
```

```cpp
int main(){
    int t;
    cin >> t;
    while(t--){
        int n;
        cin >> n;
        vector<int>v(n);
        for(int i=0 ; i<n ; i++){
            cin >> v[i];
        }
        if(canCross(v)){
            cout << "true\n";
        }else{
            cout<< "false\n";
        }
    }
}
```

Output:

```
PS C:\Users\Gaurav\Programming\practice> g++ problem5.cpp -o code
PS C:\Users\Gaurav\Programming\practice> ./code
1
8
0 1 3 5 6 8 12 17
true
PS C:\Users\Gaurav\Programming\practice>
```

Gaurav Yadav 21075034

Q6) In the game of Marble Solitaire, a row of marbles of different colors - red 'R', yellow 'Y', blue 'B', green 'G', and white 'W' - are placed on a table. Further, you have some marbles in your sack. The objective of the game is to remove all the marbles from the table. To achieve this, the player can select a marble from their sack and place it between two marbles on the table or at either end of the row. Whenever three or more marbles of the same color are arranged consecutively, the player can remove them from the table. If this removal creates more groups of three or more same-colored marbles, then the player can continue removing them until no more groups remain. The game ends when there are no marbles left on the table or when the player runs out of marbles in their sack.

problem6.cpp > updatetable(string)

```cpp
#include <bits/stdc++.h>
using namespace std;
int dfs(string table, vector<int>& freq, unordered_map<string, int>& cache);
string updatetable(string table);
string serialize(vector<int>& freq);
int findMinStep(string table, string sack) {
    vector<int> freq(26, 0);
    for(char c: sack)
        freq[c - 'A']++;
    unordered_map<string, int> cache;
return dfs(table, freq, cache);
}
int dfs(string table, vector<int>& freq, unordered_map<string, int>& cache) {
    string key = table + "#" + serialize(freq);
    if(cache.count(key)) {
        return cache[key];
    }
    int r = INT_MAX;
    if(table.length() == 0) {
        r = 0;
    } else {
        for(int i = 0; i < table.length(); i++) {
            for(int j = 0; j < freq.size(); j++) {
                bool worthTrying = false;
                if(table[i] - 'A' == j)
                    worthTrying = true;
                else if(0 < i && table[i] == table[i - 1] && table[i] - 'A' != j)
                    worthTrying = true;
                if(freq[j] > 0 && worthTrying) {
                    table.insert(table.begin() + i, j + 'A');
                    freq[j]--;
                    string newStr = updatetable(table);
                    int steps = dfs(newStr, freq, cache);
                    if(steps != -1) {
                        r = min(r, steps + 1);
                    }
                    freq[j]++;
                    table.erase(table.begin() + i);
                }
            }
        }
```

```cpp
    }
    if(r == INT_MAX) r = -1;
        cache[key] = r;
    return r;
}
string updatetable(string table) {
    int start = 0;
    int end = 0;
    int tableLen = table.length();
    while(start < tableLen) {
        while(end < tableLen && table[start] == table[end]) {
            end++;
        }
        if(end - start >= 3) {
            string newtable = table.substr(0, start) + table.substr(end);
            return updatetable(newtable);
        } else {
            start = end;
        }
    }
    return table;
}
string serialize(vector<int>& freq) {
    string key = "";
    for(int i = 0; i < freq.size(); i++) {
        if(freq[i] > 0)
        key += to_string((char) i + 'A') + to_string(freq[i]);
    }
    return key;
}
int main(){
    string table,sack;
    cin >> table >> sack;
    cout << findMinStep(table,sack);
}

```

Output:

```
PS C:\Users\Gaurav\Programming\practice> g++ problem6.cpp -o code
PS C:\Users\Gaurav\Programming\practice> ./code
WWRRBBWW WRBRW
2
PS C:\Users\Gaurav\Programming\practice>
```