DSA DAY 2

1. 0/1 KNAPSACK PROBLEM

#include <bits/stdc++.h>

using namespace std;

int knapSackRec(int W, int wt[], int val[], int index, int\*\* dp)

{

    if (index < 0)

        return 0;

    if (dp[index][W] != -1)

        return dp[index][W];

    if (wt[index] > W) {

        dp[index][W] = knapSackRec(W, wt, val, index - 1, dp);

        return dp[index][W];

    }

    else {

        dp[index][W] = max(val[index]

                           + knapSackRec(W - wt[index], wt, val,

                                         index - 1, dp),

                       knapSackRec(W, wt, val, index - 1, dp));

        return dp[index][W];

    }

}

int knapSack(int W, int wt[], int val[], int n)

{

    int\*\* dp;

    dp = new int\*[n];

    for (int i = 0; i < n; i++)

        dp[i] = new int[W + 1];

    for (int i = 0; i < n; i++)

        for (int j = 0; j < W + 1; j++)

            dp[i][j] = -1;

    return knapSackRec(W, wt, val, n - 1, dp);

}

int main()

{

    int profit[] = { 60, 100, 120 };

    int weight[] = { 10, 20, 30 };

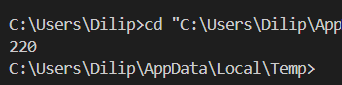
    int W = 50;

    int n = sizeof(profit) / sizeof(profit[0]);

    cout << knapSack(W, weight, profit, n);

    return 0;

}

OUTPUT:  
  
  
TC- O(N\*W)

2)FLOOR IN SORTED ARRAY

int findFloor(vector<int>& arr, int k) {

// Your code here

int low=0;

int n=arr.size();

int high=n-1;

int result=-1;

while(low<=high){

int mid=(low+high)/2;

if(arr[mid]<=k){

result=mid;

low=mid+1;

}

else{

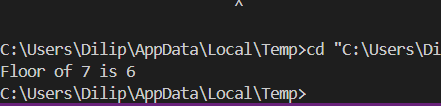
high=mid-1;

}

}

return result;

}

OUTPUT:  
  
  
TC-O(logN)

**3)CHECK EQUAL ARRAYS**

#include <bits/stdc++.h>

using namespace std;

bool areEqual(vector<int>& arr1, vector<int>& arr2)

{

    int N = arr1.size(), M = arr2.size();

    if (N != M)

        return false;

    unordered\_map<int, int> mp;

    for (int i = 0; i < N; i++)

        mp[arr1[i]]++;

    for (int i = 0; i < N; i++) {

        if (mp.find(arr2[i]) == mp.end())

            return false;

        if (mp[arr2[i]] == 0)

            return false;

        mp[arr2[i]]--;

    }

    return true;

}

int main()

{

    vector<int> arr1 = { 3, 5, 2, 5, 2 };

    vector<int> arr2 = { 2, 3, 5, 5, 2 };

    if (areEqual(arr1, arr2))

        cout << "Yes";

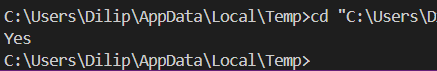
    else

        cout << "No";

    return 0;

}

OUTPUT:

  
  
TC-O(N)

**4) Palindrome linked list**

#include <iostream>

#include <stack>

using namespace std;

class Node {

public:

    int data;

    Node\* next;

    Node(int d) {

        data = d;

        next = nullptr;

    }

};

bool isPalindrome(Node\* head) {

    Node\* currNode = head;

    stack<int> s;

    while (currNode != nullptr) {

        s.push(currNode->data);

        currNode = currNode->next;

    }

    while (head != nullptr) {

        int c = s.top();

        s.pop();

        if (head->data != c) {

            return false;

        }

        head = head->next;

    }

    return true;

}

int main() {

    Node\* head = new Node(1);

    head->next = new Node(2);

    head->next->next = new Node(3);

    head->next->next->next = new Node(2);

    head->next->next->next->next = new Node(1);

    bool result = isPalindrome(head);

    if (result)

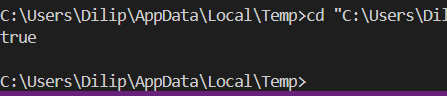
        cout << "true\n";

    else

        cout << "false\n";

    return 0;

}

OUTPUT:  
  
  
TC-O(N)

**5) Balanced tree check**

#include <bits/stdc++.h>

using namespace std;

struct Node {

    int key;

    struct Node\* left;

    struct Node\* right;

    Node(int k)

    {

        key = k;

        left = right = NULL;

    }

};

int isBalRec(Node\* root)

{

    if (root == NULL)

        return 0;

    int lh = isBalRec(root->left);

    if (lh == -1)

        return -1;

    int rh = isBalRec(root->right);

    if (rh == -1)

        return -1;

    if (abs(lh - rh) > 1)

        return -1;

    else

        return max(lh, rh) + 1;

}

bool isBalanced(Node \*root) {

  return (isBalRec(root) > 0);

}

int main()

{

    Node\* root = new Node(10);

    root->left = new Node(5);

    root->right = new Node(30);

    root->right->left = new Node(15);

    root->right->right = new Node(20);

    if (isBalanced(root))

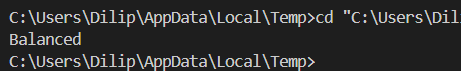
        cout << "Balanced";

    else

        cout << "Not Balanced";

    return 0;

}

OUTPUT:  
  
  
TC-O(N)  
  
**6) Triplet sum in array**  
  
#include <bits/stdc++.h>

using namespace std;

bool find3Numbers(vector<int>& arr, int sum)

{

    int n = arr.size();

    sort(arr.begin(), arr.end());

    for (int i = 0; i < n - 2; i++) {

        int l = i + 1;

        int r = n - 1;

        while (l < r) {

            int curr\_sum = arr[i] + arr[l] + arr[r];

            if (curr\_sum == sum) {

                cout << "Triplet is " << arr[i] << ", "

                    << arr[l] << ", " << arr[r];

                return true;

            } else if (curr\_sum < sum)

                l++;

            else

                r--;

        }

    }

    return false;

}

int main()

{

    vector<int> arr = { 1, 4, 45, 6, 10, 8 };

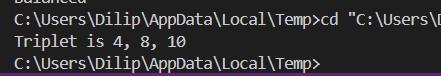
    int sum = 22;

    find3Numbers(arr, sum);

    return 0;

}

OUTPUT:

  
  
TC-O(N^2)