1. 1/0 Knapsack

#include <bits/stdc++.h>

using namespace std;

 int knapsack(int n, vector<int> profit, vector<int> weight, int capacity){

    vector<vector<int> > dp(n + 1, vector<int>(capacity + 1));

    for(int i=0;i<n+1;i++){

        for(int j=0;j<capacity+1;j++){

            if(j==0 || i==0) dp[i][j]=0;

            else{

                if(weight[i-1]<=j){

                    dp[i][j]=max(profit[i-1]+dp[i-1][j-weight[i-1]],dp[i-1][j]);

                }

                else{

                    dp[i][j]=dp[i-1][j];

                }

            }

        }

    }

    return dp[n][capacity];

 }

int main(){

    int n,capacity;

    vector<int> profit;

    vector<int> weight;

    cout<<"Enter the number of items: ";

    cin>>n;

    cout<<"Enter the capacity of the knapsack: ";

    cin>>capacity;

    cout<<"Enter the profits: ";

    for(int i = 0; i < n; i++){

        int p;

        cin>>p;

        profit.push\_back(p);

    }

    cout<<"Enter the weights: ";

    for(int i = 0; i < n; i++){

        int w;

        cin>>w;

        weight.push\_back(w);

    }

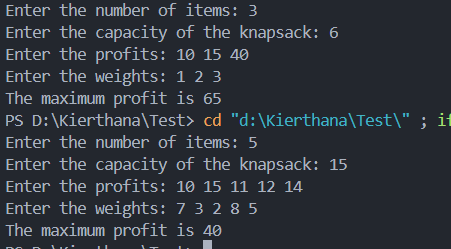
    cout<<"The maximum profit is "<<knapsack(n,profit,weight,capacity)<<endl;

    return 0;

}

TIME COMPLEXITY: O(N\*Capacity)

Output:



1. Floor in sorted array

#include <bits/stdc++.h>

using namespace std;

int floorsearch(int n,vector<int>arr,int x){

    int low=0,high=n-1;

    int mid;

    if(arr[0]>x)return -1;

    if(arr[n-1]<=x) return arr[n-1];

    while(low<=high){

        mid=low+(high-low)/2;

        if(arr[mid]==x) return arr[mid];

        if(arr[mid]<x){

            if(mid+1<n && x<=arr[mid+1]) return arr[mid];

            low=mid+1;

        }

        else{

            high=mid-1;

        }

    }

}

int main(){

    int n,x;

    vector<int> arr;

    cout<<"Enter the size of the array: ";

    cin>>n;

    cout<<"Enter the elements of the array: ";

    for(int i = 0; i < n; i++){

        int x;

        cin>>x;

        arr.push\_back(x);

    }

    cout<<"Enter the element to be searched: ";

    cin>>x;

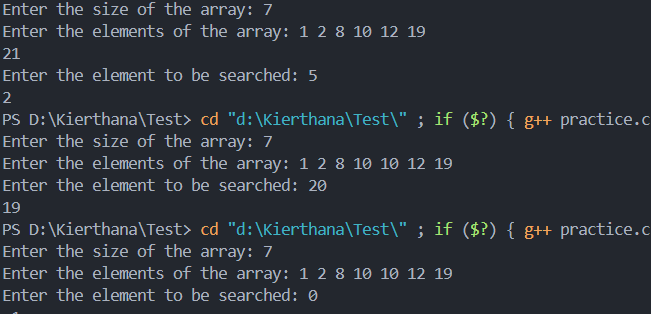
    cout<<floorsearch(n,arr,x);

    return 0;

}

TIME COMPLEXITY: O(Log N)

Output:



1. Check Equal Array

#include <bits/stdc++.h>

using namespace std;

bool equalarray(vector<int>arr1,vector<int>arr2){

    unordered\_map<int,int> h;

    for(auto i:arr1) h[i]++;

    for(auto j:arr2){

        h[j]--;

        if(h[j]<0) return false;

    }

    return true;

}

int main(){

    int n;

    vector<int> arr1,arr2;

    cout<<"Enter the size of the array: ";

    cin>>n;

    cout<<"Enter the elements of the array: ";

    for(int i = 0; i < n; i++){

        int x;

        cin>>x;

        arr1.push\_back(x);

    }

    cout<<"Enter the elements of the array: ";

    for(int i = 0; i < n; i++){

        int x;

        cin>>x;

        arr2.push\_back(x);

    }

    if(equalarray(arr1,arr2))cout<<"YES"<<endl;

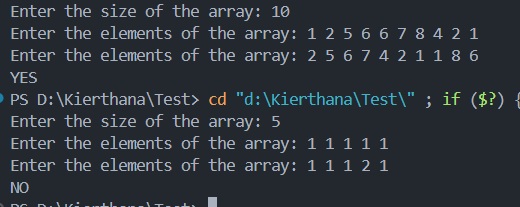
    else cout<<"NO"<<endl;

    return 0;

}

TIME COMPLEXITY: O(N)

Output:



1. Palindrome linked list

#include <bits/stdc++.h>

using namespace std;

class Node{

    public:

       int data;

       Node\* next;

       Node(int d){

        data=d;

        next=nullptr;

       }

};

class LinkedList{

    public:

        Node\* head;

       LinkedList(){

        head=nullptr;

       }

    void insert(int value){

        Node\* newnode=new Node(value);

        if(!head) head=newnode;

        else{

            Node\* curr=head;

            while(curr->next){

                curr=curr->next;

            }

            curr->next=newnode;

        }

    }

};

bool isPalindrome(Node\* head){

    stack<int> st;

    Node\* curr=head;

    while (curr){

        st.push(curr->data);

        curr=curr->next;

    }

    curr=head;

    while(curr){

        if(curr->data != st.top()) return false;

        st.pop();

        curr=curr->next;

    }

    return true;

}

int main(){

    int n;

    vector<int> arr1;

    cout<<"Enter the size of the array: ";

    cin>>n;

    cout<<"Enter the elements of the array: ";

    for(int i = 0; i < n; i++){

        int x;

        cin>>x;

        arr1.push\_back(x);

    }

    LinkedList arr;

    for(auto a:arr1){

        arr.insert(a);

    }

    if(isPalindrome(arr.head)) cout<<"YES"<<endl;

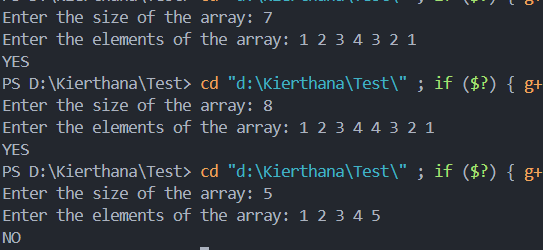
    else cout<<"NO"<<endl;

    return 0;

1. }

TIME COMPLEXITY: O(N)

Output:



1. Balanced Tree Check

#include <bits/stdc++.h>

using namespace std;

class Node{

    public:

       int data;

       Node\* left;

       Node\* right;

       Node(int d){

        data=d;

        left=nullptr;

        right=nullptr;

       }

};

int balancedtreecheck(Node\* root){

    if(!root) return 0;

    int lh=balancedtreecheck(root->left);

    if(lh==-1) return -1;

    int rh=balancedtreecheck(root->right);

    if(rh==-1) return -1;

    if(abs(lh-rh)>1) return -1;

    return max(lh,rh)+1;

}

int main(){

    Node\* root = new Node(1);

    root->left = new Node(2);

    root->right = new Node(3);

    root->right->left = new Node(4);

    root->right->right = new Node(5);

    root->left->left=new Node(6);

    root->left->left->right= new Node(7);

    if(balancedtreecheck(root)>0) cout<<"Balanced"<<endl;

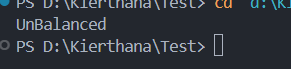
    else cout<<"UnBalanced"<<endl;

    return 0;

}

TIME COMPLEXITY: O(N)

Output:



1. Triplet Sum in Array

#include <bits/stdc++.h>

#include <algorithm>

using namespace std;

bool triplletsum(vector<int>v,int target){

    int n=v.size();

    for(int i=0;i<n;i++){

        unordered\_set<int> s;

        for(int j=i+1;j<n;j++){

            int req=target-v[i]-v[j];

            if(s.find(req) != s.end()){

                cout<<"The triplets are "<<v[i]<<" "<<v[j]<<" "<<req<<endl;

                return true;

            }

            s.insert(v[j]);

        }

    }

    return false;

}

int main(){

    int n;

    vector<int> v;

    cin >> n;

    for(int i = 0; i < n; i++){

        int x;

        cin >> x;

        v.push\_back(x);

    }

    int target;

    cin >> target;

    if(!triplletsum(v,target)){

        cout<<"No such triplets found"<<endl;

    }

    return 0;

}

TIME COMPLEXITY: O(N^2)

Output:

