Name: Roshan Vijey K C

Dept: EEE

0-1 Knapsack problem

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

int knapsack(int i,int capacity,vector<int>&weight,vector<int>&val,vector<vector<int>>&dp){
 if(i>=weight.size()){
  return 0;
 }
 if(dp[i][capacity] != -1){
  return dp[i][capacity];
 }
 if(weight[i] <= capacity) {
  return dp[i][capacity] = max(val[i]+knapsack(i+1,capacity-weight[i],weight,val,dp) ,
knapsack(i+1,capacity,weight,val,dp));
 }
 else{
  return dp[i][capacity] = knapsack(i+1,capacity,weight,val,dp);
 }
}

int main()
{
 int capacity,weightSize,valSize;

 cout << "Enter Capacity: " << endl;
 cin >> capacity;

 cout << "Enter no of objects: " << endl;
 cin >> weightSize;
```

```
  valSize = weightSize;

  vector<int>weight(weightSize),val(valSize);

  cout << "Enter weights of the objects: " << endl;
  for(int i = 0;i<weightSize;i++){
    cin >> weight[i];
  }

  cout << "Enter values of the objects: " << endl << endl;
  for(int i = 0;i<valSize;i++){
    cin >> val[i];
  }

  vector<vector<int>>dp(weightSize,vector<int>(capacity+1,-1));
  cout << "Maximum Value Obtained: " << knapsack(0,capacity,weight,val,dp) << endl;
}
```

Output: 220

Complexity:

Time: O(n*w)

Space: O(n*w)

2.Floor in a Sorted Array

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

int Floor(vector<int>&arr,int arrSize,int target){
  int low = 0,high = arrSize-1;

  int ans = -1;

  while(low<=high){
```

```
    int mid = (low+high)/2;

    if(arr[mid] == target){
      ans = mid;
      return ans;
    }
    else if(arr[mid] < target){
      ans = mid;
      low = mid+1;
    }
    else{
      high = mid-1;
    }
  }
  return ans;
}

int main()
{
  int arrSize;
  cout << "Enter array size: " << endl;
  cin >> arrSize;

  vector<int>arr(arrSize);

  cout << "Enter sorted array elements: " << endl;
  for(int i = 0;i<arrSize;i++){
    cin >> arr[i];
  }

  int target;
```

```
 cout << "Enter target value: " << endl;
 cin >> target;

 int ans = Floor(arr,arrSize,target);
 cout << "Floor of the element " << target << " is at index " << ans << endl;

}
```
OUTPUT

2

Complexity

Time: O(nlog n)

Space: O(1)

3.Check Equal Arrays

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

bool isEqual(vector<int>&arr1,vector<int>&arr2,int arrSize1,int arrSize2){
 if(arrSize1 != arrSize2){
  return false;
 }

 unordered_map<int,int>mp;
 int count = 0;

 for(int i = 0;i<arrSize1;i++){
  mp[arr1[i]]++;
 }

 for(int i = 0;i<arrSize2;i++){
  mp[arr2[i]]--;
  if(mp[arr2[i]] == 0){
   count++;
```

```
    }
  }

  if(count == arrSize1){
    return true;
  }
  else{
    return false;
  }
}

int main()
{
  int arrSize1 , arrSize2;
  cout << "Enter array 1 size: " << endl;
  cin >> arrSize1;

  cout << "Enter array 2 size: " << endl;
  cin >> arrSize2;

  vector<int>arr1(arrSize1);
  vector<int>arr2(arrSize2);

  cout << "Enter array 1 elements: " << endl;
  for(int i = 0;i<arrSize1;i++){
    cin >> arr1[i];
  }

  cout << "Enter array 2 elements: " << endl;
  for(int i = 0;i<arrSize2;i++){
    cin >> arr2[i];
```

```
 }


 bool ans = isEqual(arr1,arr2,arrSize2,arrSize1);
 cout << ans << endl;

}
```

OUTPUT

The arrays are equal

Complexity

Time: O(n)

Space: O(n)

4. Palindrome Linked List

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

struct Node{
 int val;
 Node* next;
 Node(int data){
  val = data;
  next = NULL;
 }
};

Node* createLinkedList(vector<int>&arr){
 Node* head =  new Node(arr[0]);
 Node* temp = head;
 for(int i = 1;i<arr.size();i++){
  temp->next = new Node(arr[i]);
  temp = temp->next;
```

```
 }
 return head;
}

bool isPalindrome(Node* head){

 Node* slow = head, *fast = head, *pre = NULL;
 while(fast != NULL && fast->next != NULL){
  pre = slow;
  slow = slow->next;
  fast = fast->next->next;
 }

 if(fast == NULL){
  pre->next = NULL;
  while(slow != NULL){
   Node* tmp = slow->next;
   slow->next = pre;
   pre = slow;
   slow = tmp;
  }
 }
 else{
  pre = slow;
  slow = slow->next;
  pre->next = NULL;
  while(slow != NULL){
   Node* tmp = slow->next;
   slow->next = pre;
   pre = slow;
   slow = tmp;
```

```
    }
  }

  Node* tmp1 = head,*tmp2 = pre;
  while(tmp1 != NULL && tmp2 != NULL){
    if(tmp1->val != tmp2->val){
      return 0;
    }
    tmp1 = tmp1->next;
    tmp2 = tmp2->next;
  }
  return 1;
}

int main()
{
  int arrSize;
  cout << "Enter array size: " << endl;
  cin >> arrSize;

  vector<int>arr(arrSize);

  cout << "Enter array elements: " << endl;
  for(int i = 0;i<arrSize;i++){
    cin >> arr[i];
  }

  Node* head = createLinkedList(arr);

  bool ans = isPalindrome(head);
```

```
 if(ans){
  cout << "Palindrome" << endl;
 }
 else{
  cout << "Not Palindrome" << endl;
 }
}
```

OUTPUT:

The linked list is a palindrome

Complexity:

Time: O(2*n)

Space: O(n)

5.Balanced Tree Check

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

struct Node{
 int val;
 Node* left;
 Node* right;
 Node(int data){
  val = data;
  left = NULL;
  right = NULL;
 }
};

Node* createBinaryTree(vector<int>&arr){
 Node* root = new Node(arr[0]);
 queue<Node*>q;
 q.push(root);
```

```
 int i = 1;
 while(i<arr.size()){
  int levelSize = q.size();
  while(levelSize--){
   Node* front = q.front();
   q.pop();
   if(i<arr.size()){
    front->left = (arr[i] != -1) ? new Node(arr[i]) : NULL;
    if(front->left != NULL){
     q.push(front->left);
    }
   }
   i++;
   if(i<arr.size()){
    front->right = (arr[i] != -1) ? new Node(arr[i]) : NULL;
    if(front->right != NULL){
     q.push(front->right);
    }
   }
   i++;
  }
 }
 return root;
}

pair<int,bool> isBalanced(Node* root){
 if(root == NULL){
  return {0,true};
 }
 pair<int,bool>left = isBalanced(root->left);
```

```
  int leftHeight = left.first;
  bool isLeftBalanced = left.second;

  pair<int,bool>right = isBalanced(root->right);
  int rightHeight = right.first;
  bool isRightBalanced = right.second;

  if(!isLeftBalanced || !isRightBalanced){
    return {0,false};
  }

  if(abs(rightHeight - leftHeight) <= 1){
    return {1+max(rightHeight,leftHeight),true};
  }
  else{
    return {0,false};
  }
}

int main()
{
  int arrSize;
  cout << "Enter array size: " << endl;
  cin >> arrSize;

  vector<int>arr(arrSize);

  cout << "Enter array elements: " << endl;
  for(int i = 0;i<arrSize;i++){
    cin >> arr[i];
  }
```

```
 Node* root = createBinaryTree(arr);

 bool ans = isBalanced(root).second;

 if(ans){
  cout << "Balanced" << endl;
 }
 else{
  cout << "Not Balanced" << endl;
 }
}
```

OUTPUT

Is the tree balanced? True

Complexity

Time: O(n)

Space: O(n)

6.Triplet Sum Array

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

class Solution {
 public:
  bool find3Numbers(int arr[], int n, int x) {
      sort(arr,arr+n);
      int i = 0;
      while(i<n){
         int s = x-arr[i];
         int j = i+1,k = n-1;
         while(j<k){
            if(arr[j]+arr[k] == s){
               return true;
```

```
            }
            else if(arr[j]+arr[k] > s){
                k--;
            }
            else{
                j++;
            }
          }
          i++;
       }
       return false;

   }
};

int main() {
   int T;
   cout << "Enter no of testcases";
   cin >> T;
   while (T--) {
      cout << "Enter length of array";
      int n, X;
      cin >> n;
      cout << "Enter target sum";
      cin >> X;
      int i, A[n];
      for (i = 0; i < n; i++)
         cin >> A[i];
      Solution ob;
      cout << ob.find3Numbers(A, n, X) << endl;
   }
```

}

OUTPUT

Triplet: 12,3,9

Complexity

Time: O(nlogn)

Space: O(1)