**8.4 Array challenges**

**Problem 1:**

**Given an array a[] of size n. for every 'i' from 0 to n-1.**

**Approach:**

1. Keep a variable mx which stores the maximum till 'i' element.

2. Iteration over the array and update.

mx = max(mx, a[i])

**Output**: max(a[0],a[1],....a[i]).

#include<iostream>

using namespace std;

int main()

{

    int mx = -1999999;

    int n;

    cin>>n;

    int a[n];

    for(int i=0; i<n; i++){

        cin>>a[i];

    }

    for(int i=0; i<n; i++){

        mx=max(mx, a[i]);

        cout<<mx<<endl;

    }

    return 0;

}

**Sub-array v/s sub-sequence array.**

**Sub-Array:**

Continuous part of the array.

Array with nth element = ncr+n = n\*(n+1)/2.

**Sub-Sequence Array:**

A Sub-sequence is a sequence that can be derived an array by selection zero or more element, without changing the order of the remaining elements.

Array with nth element = 2n.

**Problem 2:**

Given an array a[] of size n. **Output**: sum of each sub-array of the given array.

#include<iostream>

using namespace std;

int main()

{

    int n;

    cin>>n;

    int a[n];

    for(int i=0; i<n; i++){

        cin>>a[i];

    }

    int curr=0;

    for(int i=0; i<n; i++)

    {

        curr=0;

        for(int j=i; j<n; j++){

            curr += a[j];

            cout<<a[j]<<" ";

        }

    }

    return 0;

}

**Important Question: longest Arithmetic sub-Array**

***(Google kick-start)***

**Problem:**

An arithmetic array is an array that contains at least two integers and the differences between consecutive integers are equal. For example, [9, 10], [3, 3, 3], and [9, 7, 5, 3] are arithmetic arrays, while [1, 3, 3, 7],

[2, 1, 2], and [1, 2, 4] are not arithmetic arrays.

Sarasvati has an array of **N** non-negative integers. The i-th integer of the array is **Ai**. She wants to choose a contiguous arithmetic sub array from her array that has the maximum length. Please help her to determine the length of the longest contiguous arithmetic sub array.

**Input:**

The first line of the input gives the number of test cases, **T. T** test cases follow. Each test case begins with a line containing the integer **N**. The second line contains N integers. The i-th integer is **Ai.**

**Output:**

For each test case, output one line containing **Case #x: y**, where **x** is the test case number (starting from 1) and **y** is the length of the longest contiguous arithmetic subarray.

**Constraints:**

Time limit: 20 seconds per test set.

Memory limit: 1GB.

1 ≤ T ≤ 100.

0 ≤ Ai ≤ 109

Test Set 1

2 ≤ N ≤ 2000.

Test Set 2

2 ≤ N ≤ 2 × 105 for at most 10 test cases.

For the remaining cases, 2 ≤ N ≤ 2000.

**Solution:**

**Constraints Analysis**

1 sec = 108 operations

20 sec = 2x109 operations

**Intuition:**

We have to loop over the array and find the answer.

**Steps:**

1. While iterating in the array we need to maintain the following variables,

    a. Previous common difference (pd) - To compare it with current

        common difference **(a[i] - a[i-1]).**

    b. Current arithmetic subarray length (curr) - It denotes the arithmetic

        subarray length including **a[i].**

    c. Maximum arithmetic subarray length (ans) - It denotes the max.

        Arithmetic subarray length till **a[i].**

2. While iterating, there will be two cases,

    a. **pd = a[i] - a[i-1]**

        i. Increase curr by 1

        ii. **ans = max(ans, curr)**

3. After loop ends, output the answer. (stored in ans variable).

#include<iostream>

using namespace std;

//longest Arithematic sub-Array

int main()

{

    int n;

    cin>>n;

    int a[n];

    for(int i=0; i<n; i++){

        cin>>a[i];

    }

    int ans = 2;

    int pd = a[1]-a[0];

    int j=2;

    int curr = 2;

    while(j<n)

    {

        if(pd == a[j] - a[j-1])

        {

            curr++;

        }

        else{

            pd = a[j] - a[j-1];

            curr = 2;

        }

        ans = max(ans, curr);

        j++;

    }

    cout<<ans<<endl;

    return 0;

}

**Arrays Challenge-Record Breaker**

***(Google kickstart)***

**Problem:**

Isyana is given the number of visitors at her local theme park on **N** consecutive days. The number of visitors on the i-th day is **Vi** . A day is record breaking if it satisfies both of the following conditions:

* The number of visitors on the day is strictly larger than the number of visitors on each of the previous days.
* Either it is the last day, or the number of visitors on the day is strictly larger than the number of visitors on the following day.

Note that the very first day could be a record breaking day! Please help Isyana find out the number of record breaking days.

**Input:**

The first line of the input gives the number of test cases, **T. T** test cases follow. Each test case begins with a line containing the integer **N**. The second line contains **N** integers. The i-th integer is **Vi** .

**Output :**

For each test case, output one line containing Case #x: y, where x is the test case number (starting from 1) and y is the number of record breaking days.

Constraints

Time limit: 20 seconds per test set.

Memory limit: 1GB.

1 ≤ T ≤ 100.

0 ≤ Vi ≤ 2 × 105 .

Test set 1

1 ≤ N ≤ 1000.

Test set 2

1 ≤ N ≤ 2 × 105 for at most 10 test cases.

For the remaining cases, 1 ≤ N ≤ 1000.

**Solution :**

**Constraints Analysis :**

1 sec = 108 operations

20 sec = 2x109 operations

**Brute Force Approach:**

Iterate over all the elements and check if it is record breaking day or not.

Note: To check if a[i] is a record breaking day, we have to iterate over a[0], a[1],...., a[i-1].

Time complexity for this operation: O(n)

Overall Time Complexity: O(n2 )

**Optimised Approach:**

Intuition: If we can optimise step (1), then we can optimise our overall solution.

For step (1): We need to check if a[i] > { a[i-1], a[i-2],..., a[0] }, which is same as

a[i] > max(a[i-1], a[i-2],..., a[0])

For this, we will keep a variable mx, which will store the maximum value till a[i]. Then we just need to check,

a[i] > mx

a[i] > a[i+1] , { if i+1 < n }

and update mx, mx = max(mx, a[i])

So step (1) time complexity reduces to O(1).

**Overall time complexity: O(n) Code**

#include<iostream>

using namespace std;

// record Breaker array

int main()

{

    int n;

    cin>>n;

    int a[n+1];

    a[n]= -1;

    for(int i=0; i<n; i++){

        cin>>a[i];

    }

    if(n == 1){

        cout<<"1"<<endl;

        return 0;

    }

    int ans = 0;

    int mx = -1;

    for(int i=0; i<n; i++){

        if(a[i]>mx && a[i]>a[i+1]){

            ans++;

        }

        mx = max(mx, a[i]);

    }

    cout<<ans<<endl;

    return 0;

}

**8.5 Array – Q’s asked by top MNC’s.**

**Arrays Challenge-First Repeating Element**

***(Amazon, Oracle)***

**Problem:**

Given an array **arr**[] of size **N**. The task is to find the first repeating element in an array of integers, i.e., an element that occurs more than once and whose index of first occurrence is smallest.

**Constraints**

1 <= N <= 106

0 <= Ai <= 106

**Example**

Input:

7

1 5 3 4 3 5 6

Output: 2

Explanation:

5 is appearing twice and its first appearance is at index 2 which is less than 3 whose first occurring index is 3.

**Solution:** Base idea:

To check if an element is repeating, we maintain an array idx[], which stores the first occurrence (index) of a particular element a[i].

Steps:

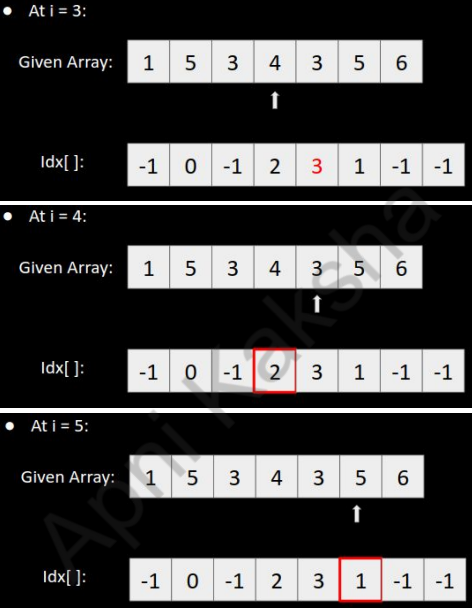
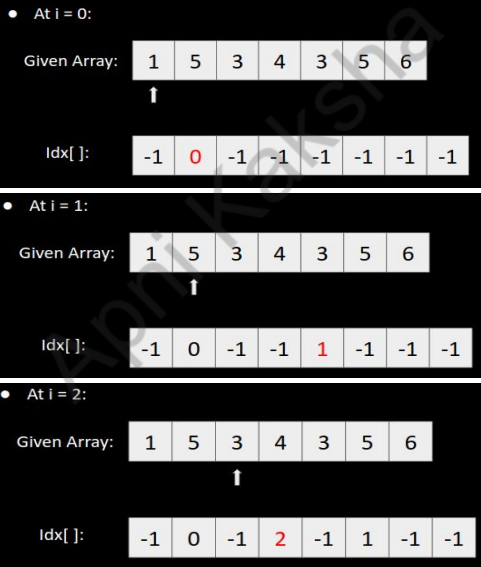
1. Initialise the idx[] with -1, and minidx with INT\_MAX.

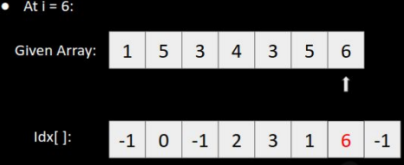


2. Keep updating idx[], while traversing the given array.



Iterations:





#include<bits/stdc++.h>

using namespace std;

int main() {

    int n;

    cin>>n;

    int a[n];

    for(int i=0; i<n; i++){

        cin>>a[i];

    }

    const int N = 1e6+2;

    int idx[N];

    for(int i=0; i<N;i++){

        idx[i] = -1;

    }

    int minidx = INT\_MAX;

    for(int i=0; i<n; i++)

    {

        if(idx[a[i]] != -1)

        {

            minidx = min(minidx, idx[a[i]]);

        }

        else

        {

           idx[a[i]] = i;

        }

    }

    if(minidx == INT\_MAX)

    {

        cout<<"-1"<<endl;

    }

    else

    {

        cout << minidx +1 << endl;

    }

    return 0;

}

**Arrays Challenge-Subarray with given Sum**

***(Google, Amazon, Facebook, Visa)***

**Problem:**

Given an unsorted array **A** of size **N** of non-negative integers, find a continuous subarray which adds to a given number **S**.

**Constraints**

1 <= N <= 105

0 <= Ai <= 1010

**Example**

Input:

N = 5, S = 12

A[] = {1,2,3,7,5}

Output**: 2 4**

Explanation: The sum of elements from 2nd position to 4th position is 12.

**Solution:**

**Brute Force Solution**

* Find sum of all possible sub-arrays. If any of the sum equates to S, output the starting and ending index of the sub-array.

Time Complexity: **O(n2)**

**Optimized Approach**

**Steps**:

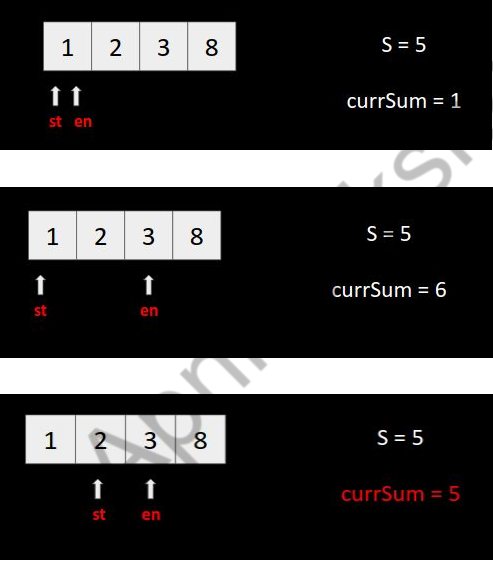
1. Keep the pointers st and en, and a variable currSum that stores the sum from st to en.

2. Initialize st = 0, en = 0

3. Increment en till currSum + a[en + 1] > S

4. When 3rd condition occurs, start increasing st until currSum <= S.

5. Whenever the condition (currSum = S) is satisfied, store st and en and BREAK from the loop.

Iterations:

**Code**:

#include<bits/stdc++.h>

using namespace std;

/\*  subarray with given sum \*/

int main()

{

    int n, s;

    cin >>n>>s;

    int a[n];

    for(int i=0; i<n; i++){

        cin>>a[i];

    }

    int i=0, j=0, st=-1, en=-1, sum=0;

    while(j<n && sum+a[j]<=s)

    {

        sum += a[j];

        j++;

    }

    if(sum==s){

        cout<<i+1<<""<<j<<endl;

        return 0;

    }

    while(j<n)

    {

        sum += a[j];

        while(sum>s){

            sum -= a[i];

            i++;

        }

        if(sum==s){

            st = i+1;

            en = j+1;

            break;

        }

        j++;

    }

    cout<<st<<" "<<en<<endl;

    return 0;

}

**Arrays Challenge - Smallest Positive Missing Number**

***(Amazon, Samsung, Snapdeal, Accolite)***

**Problem**

Find the smallest positive missing number in the given array. Example: [0, -10, 1, 3, -20], Ans = 2

**Steps to Solve:**

1. Build the Check[ ] array initialized with False at all indices.

2. By iterating over the array and marking non-negative a[i] as true i.e.

if(a[i] >= 0)

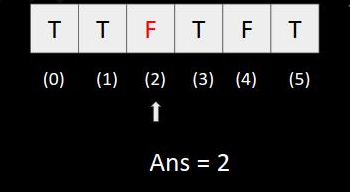
check[a[i]] = True

3. Iterate in the Check[ ] from i=1, BREAK the loop when you find check[i] =False and store that i in the ans variable.

4. Output the ans.

**Example:** Given Array: [0, -9, 1, 3, -4, 5]Iterations:





#include<bits/stdc++.h>

using namespace std;

/\* Smallest positive missing number.   \*/

int main()

{

    int n;

    cin>>n;

    int a[n];

    for(int i=0; i<n; i++){

        cin>>a[i];

    }

    const int N = 1e6 + 2;

    bool check[N];

    for(int i=0; i<n; i++){

        check[i]= false;

    }

    for(int i=0; i<n; i++){

        if(a[i] >= 0){

            check[a[i]] = 1;

        }

    }

    int ans = -1;

    for(int i=1; i<N; i++){

        if(check[i] == false){

            ans = i;

            break;

        }

    }

    cout<<ans<<endl;

    return 0;

}

**8.6 Sub-Array : (Important)**

“Subarray is a Continuous part of the array”. **Ex**:- {-2,0,3,7,11}

Array with nth element = nc2+n = n\*(n+1)/2.

**Q.Given sub array print . Input = {-1,4,7,2}**

#include<bits/stdc++.h>

using namespace std;

int main()

{

    int n;

    cin>>n;

    int arr[n];

    for(int i=0; i<n; i++)

    {

        cin>>arr[i];

    }

    for (int i=0; i<n; i++)

    {

        for(int j=i; j<n; j++)

        {

            for(int k=i; k<=j; k++)

            {

                cout<<arr[k]<<" ";

            }

            cout<<endl;

        }

    }

    return 0;

}

**Question:- Find the subarray is an array which has maximum sum.**

**1. Brute Force:**

Idea: For each subarray arr[i..j], calculate its sum. Time Complexity : **O(N3).** Space Complexity : **O(1)**

#include<bits/stdc++.h>

using namespace std;

int main()

{   int n, sum=0;

    int maxSum=INT\_MIN;

    cin>>n;

    int arr[n];

    for(int i=0; i<n; i++) {

        cin>>arr[i];

    }

    for (int i=0; i<n; i++) {

        for(int j=i; j<n; j++) {

            for(int k=i; k<=j; k++) {

                sum += arr[k];

            }

            maxSum=max(maxSum, sum);

        }

    }

    cout<<maxSum<<endl;

    return 0;

}

**2. Cumulative Sum approach:**

**Idea**: For each subarray arr[i..j], calculate its sum. Time Complexity: **O(N2)**Space Complexity**: O(N)**

#include<bits/stdc++.h>

using namespace std;

/\*  Maximum Sum Subarray with Cumulative Sum approach:  \*/

int main()

{

    int n;

    cin>>n;

    int arr[n];

    for(int i=0; i<n; i++){

        cin>>arr[i];

    }

    int currsum[n+1];

    currsum[0] = 0;

    for(int i=1; i<=n; i++){

        currsum[i] = currsum[i-1] + arr[i-1];

    }

    int maxsum = INT\_MIN;

    for(int i=1; i<=n; i++){

        int sum = 0;

        for(int j=0; j<i; j++){

            sum = currsum[i] - currsum[j];

            maxsum = max(sum, maxsum);

        }

    }

    cout<<maxsum<<endl;

    return 0;

}

**3. Kadane’s Algorithm:**

Idea: Start taking the sum of the array, as soon as it gets negative, discard the current subarray, and start a new sum.

Time Complexity: **O(N)** Space Complexity: **O(1)**

#include<bits/stdc++.h>

using namespace std;

int main()

{   int n;

    cin >> n;

    int arr[n];

    for (int i = 0; i < n; i++){

        cin >> arr[i];

    }

    int currsum = 0;

    int maxsum = INT\_MIN;

    for (int i = 0; i < n; i++)

    {

        currsum += arr[i];

        if (currsum<0){

            currsum = 0;

        }

        maxsum = max(maxsum, currsum);

    }

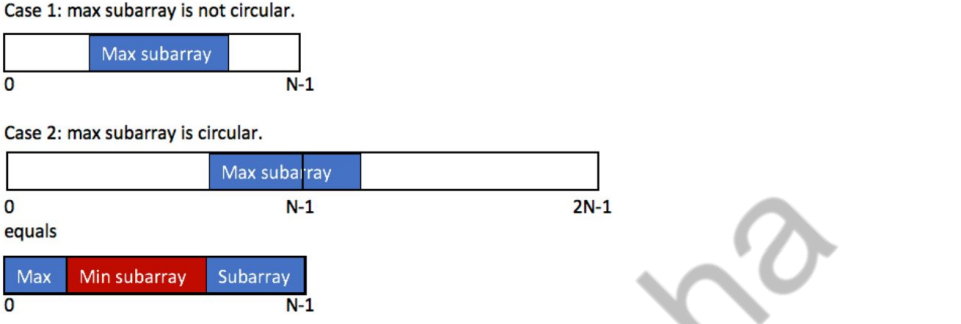
    cout << maxsum << endl;

    return 0;

}

**Question : - Maximum Sum Circular Subarray:**

**Idea**: There will 2 cases,



To get the Min subarray we multiply the array by -1 and get the maximum sum subarray.

**Time Complexity: O(N)**

#include<bits/stdc++.h>

using namespace std;

/\*  maximum cicular sub-array sum   \*/

int kadane(int arr[], int n){

    int currentsum=0;

    int maxsum=INT\_MIN;

    for(int i=0; i<n; i++){

        currentsum+=arr[i];

        if(currentsum<0){

            currentsum=0;

        }

        maxsum=max(maxsum, currentsum);

    }

    return maxsum;

}

int main()

{

    int n;

    cin>>n;

    int arr[n];

    for(int i=0; i<n; i++){

        cin>>arr[i];

    }

    int wrapsum;

    int nonwrapsum;

    nonwrapsum= kadane(arr, n);

    int totalsum=0;

    for(int i=0; i<n; i++){

        totalsum+=arr[i];

        arr[i]=-arr[i];

    }

    wrapsum = totalsum + kadane(arr, n);

    cout<<max(wrapsum, nonwrapsum)<<endl;

    return 0;

}

**Question: - Pair sum Problem:**

Check if exists two element is an array such that there is equal to given **k**.

1. Time complexity: **O(n2)**

#include<bits/stdc++.h>

using namespace std;

/\*  Pair Sum problem Sub-array  \*/

bool pairsum(int arr[], int n, int k){

    for (int i=0; i<n-1; i++){

        for(int j=i+1; j<n; j++){

            if(arr[i]+arr[j]==k){

                cout<<i<<" "<<j<<endl;

                return true;

            }

        }

    }

    return false;

}

int main()

{

    int arr[]= {2,4,7,11,14,16,20,21};

    int k=31;

    cout<<pairsum(arr, 8, k)<<endl;

    return 0;

}

1. Time complexity: **O(n)**

#include<bits/stdc++.h>

using namespace std;

/\*  Pair Sum problem Sub-array  in time complexity : O(n)\*/

bool pairsum(int arr[], int n, int k){

    int low=0;

    int high=n-1;

    while(low<high){

        if(arr[low]+arr[high]==k){

            cout<<low<<" "<<high<<endl;

            return true;

        }

        else if(arr[low]+arr[high]>k){

            high--;

        }

        else{

            low++;

        }

    }

    return false;

}

int main()

{

    int arr[]= {2,4,7,11,14,16,20,21};

    int k=31;

    cout<<pairsum(arr, 8, k)<<endl;

    return 0;

}