Coding interview

//1 convert a string of roman numerals into an integer value.

public class reverseString { public static void main(String[] args) {

String input="";

System.out.println("Enter the input string");

try

{

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

input = br.readLine();

char[] try1= input.toCharArray();

for (int i=try1.length-1;i>=0;i--)

System.out.print(try1[i]);

}

catch (IOException e) {

e.printStackTrace();

}

}}

Method 2:

String input="AliveisAwesome";

StringBuilder input1 = new StringBuilder();

input1.append(input);

input1=input1.reverse();

for (int i=0;i<input1.length();i++)

System.out.print(input1.charAt(i));

String input = "Be in present";

char[] temparray= input.toCharArray();

int left,right=0;

right=temparray.length-1;

for (left=0; left < right ; left++ ,right--)

{

// Swap values of left and right

char temp = temparray[left];

temparray[left] = temparray[right];

temparray[right]=temp;

}

for (char c : temparray)

System.out.print(c);

System.out.println();

}}

Method 3:

String input = "Be in present";

char[] hello=input.toCharArray();

List<Character> trial1= new LinkedList<>();

for(char c: hello)

trial1.add(c);

Collections.reverse(trial1);

ListIterator li = trial1.listIterator();

while(li.hasNext())

{System.out.print(li.next());}

//Method take string parameter and check string is empty or not

public static String reverseString(String input)

{

if (input.isEmpty()){

return input;

}

//Calling Function Recursively

return reverseString(input.substring(1)) + input.charAt(0);

}

String input = "Be in present";

byte [] strAsByteArray = input.getBytes();

byte [] result = new byte [strAsByteArray.length];

for(int i = 0; i<strAsByteArray.length; i++){

result[i] = strAsByteArray[strAsByteArray.length-i-1];

}

System.out.println( new String(result));

# Find The First Non Repeated Character In A String

public class FirstNonRepeated {

public static void main(String[] args)

{

// TODO Auto-generated method stub

System.out.println(" Please enter the input string :" );

Scanner in = new Scanner (System.in);

String s=in.nextLine();

char c=firstNonRepeatedCharacter(s);

System.out.println("The first non repeated character is : " + c);

}

public static Character firstNonRepeatedCharacter(String str)

{

HashMap<Character,Integer> characterhashtable=

 new HashMap<Character ,Integer>();

int i,length ;

Character c ;

length= str.length(); // Scan string and build hash table

for (i=0;i < length;i++)

{

c=str.charAt(i);

if(characterhashtable.containsKey(c))

{ // increment count corresponding to c

characterhashtable.put( c , characterhashtable.get(c) +1 );

}

else

{

characterhashtable.put( c , 1 ) ; } }

// Search characterhashtable in in order of string str

for (i =0 ; i < length ; i++ )

{

c= str.charAt(i);

if( characterhashtable.get(c) == 1 )

return c;

}

return null ;

}

}

# Intersection Of Two Arrays In Java With Example

Before giving coding examples , We should understand first what is Intersection of two arrays is ?  
Intersection of two arrays means finding out common elements or matching elements . Most of the algorithm based technical java interview served this question as a start up recipe to the students .  
So one should prepare this question before appearing for the technical interview .  
For example , Suppose we have given two arrays  
**array1** contains **{ 0 ,  1 , 2 ,  3 ,  4  ,  5  }**  
**array2**  contains  **{  4 ,  7 , 8 , 9 , 10 , 1 }**  
Then **intersection of array1 and array2** will be :    **{ 1 , 4  }**  
**Pseudo algorithm :**  
1.   Print both given arrays  
2.  Loop through first array till the first  array length  
                     Inside first array loop,  loop the second array  till the length   of the second array  
                             2.1       Compare elements of first array to the elements of the second array  
                             2.2       If  first array element matches with second array element  
                                              create new third  resulting array  and store the matching element in it  
                                         else  
                                         continue  
3. Print the third resulting  array   showing intersection of the two given arrays

public class Intersection {

static int i,j,k,c=0,w;

public static void main(String[] args)

{

int[] x= {2,5,3,7};

System.out.println("The first array is " +" ");

for(int i=0;i;x.length;i++)

System.out.print(x[i]+" ");

System.out.println("");

int[] y={5,2,9,0,1};

System.out.println("The second array is " +" ");

for(int j=0;j;y.length;j++)

System.out.print(y[j]+" ");

System.out.println("");

intersection(x,y);

}

static void intersection(int x[],int y[])

{

int []z=new int[x.length+y.length];

for(i=0;i <(x.length);i++)

{

for(j=0;j < y.length;j++)

{

if(x[i]==y[j])

{

z[c]=x[i];

c++;

}

else

continue;

} }

System.out.println("Intersection of two array is " +" ");

for(k=0;k < c;k++)

{

System.out.print(" "+z[k]+" ");

}

System.out.println(" ");

} }

# 5 Ways To Determine If String Has All Unique Characters

**PseudoCode for Method 1 :**  
  
1. Create a HashSet object.  
2. Scan the whole string, and add each character one by one to the HashSet object  
3. If the add object  returns true then continue  
    else return false   
***Method 1***  
public class uniquechar {

public static void main (String args[])

{

boolean result=false;

String inputstring="Alve i@wsom";

System.out.println(inputstring);

HashSet < Character> uniquecharset= new HashSet();

for(int i=0;i < inputstring.length();i++)

{

result=uniquecharset.add(inputstring.charAt(i));

if (result == false)

break; } System.out.println(result); }}

**PseudoCode for Method 2 :**  
1.Scan the input string , take each character one by one and set count flag to 0.  
2. For each character in the inputstring ,Rescan the inputstring and compare the character with each character appear in the inputstring  
3. If equal then increase the count by 1  
                   else continue the loop  
4.  If count flag value is greater than 1 then return false  
                 else return true  
**Method 2**

public class UniqueChar2 {

public static void main (String args[])

{

boolean result=false;

String inputstring="Alive is awesome";

System.out.println("String method 2 answer "+ method2(inputstring));

}

public static boolean method2(String input)

{

for(int i=0; i < input.length();i++)

{

char charcterofinputstring=input.charAt(i);

int count=0;

for(int j=i; j < input.length();j++)

{

if (charcterofinputstring==input.charAt(j))

count++;

}

if(count > 1)

return false;

}

return true;

}

}  
PseudoCode for Method 3:  
1. indexOf() returns the index of first occurence of the character or else return -1. So , here we are creating an arraylist object.  
2. Scan the inputstring and add the index of each character to the arraylist object.  
3. Sort the arraylist object.

[](https://4.bp.blogspot.com/-IE5AD_UH_H0/VHtvDgBkl5I/AAAAAAAAAjw/sn1wWNihS2o/s1600/unique+characters+in+string+java+example.jpg)

  4. Compare the values of  each adjacent positions of arraylist object  
 if equal then return false  
                  else continue scanning the arraylist  
5.  return true  
**Method 3**  
public class UniqueChar3 {

public static void main (String args[])

{

boolean result=false;

String inputstring="Alive is awesome";

System.out.println("String method 3 answer "+ method3(inputstring));

}

public static boolean method3(String input)

{

ArrayList ar= new ArrayList();

for (int i=0; i < input.length() ; i++ )

{

int j = input.indexOf(input.charAt(i));

ar.add(j);

}

Collections.sort(ar);

for (int i=0;i < (ar.size()-1);i++)

{

if (ar.get(i) == ar.get(i+1))

return false;

}

return true;

}

}

 PseudoCode for Method 4 :  
1.  For this method we need to know about  two inbuilt functions in java , indexOf() which returns the index of first occurence of the character in the string , while second function lastIndexOf() returns the index of last occurence of the character in the given string.  
2. First , we convert the given inputstring into characterarray by using toCharArray() function.  
3. Calculate the indexOf() and lastIndexOf() for each character in the given inputstring  
4. If both are equal then continue and make result= true  
        else set flag result = false  
5. Return result  
Method 4  
public class UniqueChar4 {

public static void main (String args[])

{

boolean result=false;

String inputstring="Alive is awesome";

System.out.println("String method 4 answer "+ method4(inputstring));

}

public static boolean method4(String input)

{

boolean result=false;

for (char ch: input.toCharArray())

{

if(input.indexOf(ch)== input.lastIndexOf(ch))

result= true;

else

{

result=false;

break;

} }

return result;

}}

**Method 5**

public static boolean checkForUnique(String str){

boolean containsUnique = false;

for(char c : str.toCharArray()){

if(str.indexOf(c) == str.lastIndexOf(c)){

containsUnique = true;

} else {

containsUnique = false;

}

}

return containsUnique;}

# Count Number Of Words In The String

public class StringDemo

{

static int i,c=0,res;

static int wordcount(String s)

{

char ch[]= new char[s.length()]; //in string especially we have to mention the () after length

for(i=0;i<s.length();i++)

{

ch[i]= s.charAt(i);

if( ((i>0)&&(ch[i]!=' ')&&(ch[i-1]==' ')) || ((ch[0]!=' ')&&(i==0)) )

c++;

}

return c; }

public static void main (String args[])

{

res=StringDemo.wordcount(" manchester united is also known as red devil ");

//string is always passed in double quotes

System.out.println("The number of words in the String are : "+res); }}

#### ****Question 1****

Given an array of integers, you must find a pair within the array that sums up to a given sum. We assume the array is unsorted.

For example:

**Input:**

Array A = {4, 9, 7, 3, 6, 8}

Sum = 11

**Output:**

Pair found at 0 and  2 (4 +  7)

Pair found at 3 and 5 (3 + 8)

[Solution 1: Inefficient Solution](https://www.kindsonthegenius.com/coding/2019/02/05/question-1-find-pair-with-given-sum-in-an-array/#t1)

[Solution 2: Start with Sorting](https://www.kindsonthegenius.com/coding/2019/02/05/question-1-find-pair-with-given-sum-in-an-array/#t2)

[Solution 3: Use a hashmap](https://www.kindsonthegenius.com/coding/2019/02/05/question-1-find-pair-with-given-sum-in-an-array/#t3)

**Solution 1: Inefficient Solution (Quadratic time)**

In this solution, we loop through the array and for each element, we calculate the sum with every other element and compare with the given sum. If the sum equals the given sum. We output the indexes.

The Java program is given below:

//Question 1: Given an unsorted array of integers

// find a pair of integers with the given sum in it

package examples;

public class FindPairSum1{

public static void findPair1(int A[], int sum) {

//loop until element before the last

for(int i = 0; i< A.length-1; i++) {

//loop until the last element

for(int j = i+1; j < A.length; j++) {

//if given sum is found, print the indexes

if(A[i] + A[j] == sum) {

System.out.println("Pair found at " + i + " and " + j);

} } } }//End of findPair1

public static void main(String args[]) {

int ar[] = {3,5,2,8,0,1,11,7};

int sum = 10;

findPair1(ar, sum);

}//End of main

}

**Solution 1:** Inefficient Solution – Quadratic Time

**Solution 2: Start by Sorting (nlog(n) time)**

In this solution, we first sort the array. As such, the largest elements  are are the end of the array while the smallest elements are at the beginning of the array. Then we use two iterators. The first counting from the beginning of the array while the second counting from the end of the array.For each iteration,  we calculate the sum to check if it equals the given sum

The Java program is given below.

Note: The index returned with this method is the index of the elements in the sorted array which is not the same as the indexes in the original array.

//findPair2: Sorts the array. Maintain two indices of the two end-points

//Complexity: O(nlogn)

package examples;

import java.util.Arrays;

public class FindPairSum2 {

public static void findPair2 (int A[], int sum) {

Arrays.sort(A); //First sort the array

int low = 0;

int high = A.length - 1;

while(low < high) {

if(A[low] + A[high] == sum) {

System.out.println("Sum found are " + A[low] + " and " + A[high]);

low++;

}

else if(A[low] + A[high] < sum) {

low = low + 1;

}

else {

high = high - 1;

}

} //end of while loop

}

public static void main(String[] args) {

int ar[] = {3,5,2,8,0,1,11,7};

int sum = 10;

findPair2(ar, sum);

}

}

Solution 2: Maintain two indexes

**Solution 3 – Use of HashMap (Linear time O(n))**

In this solution, we iterate though the array just once and store each element in a hash set.. For each element, we would check if the complement of the element is already  in the hashset. If it is there, then we have  a sum. If it is not there then we insert it.

The Java implementation is given below.

//Find pair with given sum using a hashma[

package examples;

import java.util.HashMap;

import java.util.Map;

public class FindPairSum3 {

public static void findPair(int A[], int sum) {

Map<Integer, Integer> map = new HashMap<>();

for(int i = 0; i< A.length; i++) {

//if complement is already in map, print the pair

if(map.containsKey(sum-A[i])) {

System.out.println("Pair found at " + map.get(sum-A[i]) + " and " + i);

}

map.put(A[i], i);

}//end of for }//end for function

public static void main(String[] args) {

int ar[] = {3, 20, 5, 6,1,9, 7, 22};

int sum = 29;

findPair(ar, sum); }}

#### **Question 2**

You are given an integer array. You need to check if the array contains sub-array with zero sum.

For example:

**Input**: A = {4, -1, -3, 1}

**Output**: Subarray with zero sum exists

{4, -1, -3}

**Solution 1: Inefficient Solution – Quadratic Time – O(n2)**

In this solution we take running sum of all each element with every other element, and each time we check if the sum is equal to zero. If yes then we return true.

The Java code is given below:

public class ZeroSumSubarray1 {

public static boolean zeroSumSubarray(int A[]) {

int sum;

for(int i = 0; i < A.length; i++) {

sum = 0;

for(int j = i; j < A.length; j++) {

sum = sum + A[j];

if(sum == 0) {

System.out.println("Subarray exists");

return true; }

}//end of for loop

}//end of for loop

System.out.println("Subarray does not exist");

return false;

}//end of zeroSumSubarray

public static void main(String[] args) {

int ar[] = {4, -1, -3, 1};

zeroSumSubarray(ar);

}}

**Solution 2: Using a Hash Table – Linear Time O(n)**

In this solution, start from the first element and take running sum. Then for each element, we store the sum so far in a hash table. Before we store an element, we check the hash table  to see if the sum already exists (ie if it has been seen before). If yes, then we know hat there is  a sub-array with zero sum that ends at the current index.

The Java program is given below:

/\*

\* Written by Kindson the Genius

\* Check is Subarry with zero sum exists

\*/

public class ZeroSumSubarray2 {

public static Boolean ZeroSumSubarray(int A[]) {

Set<Integer> set = new HashSet<Integer>();

int sum = 0;

set.add(sum);

for(int i = 0; i<A.length; i++) {

sum = sum + A[i];

if(set.contains(sum)) {

return true; }

set.add(sum);

}//end of for loop

return false;

}//end of function

public static void main(String[] args) {

int ar[] = {1, -4, 3, 4};

System.out.println(ZeroSumSubarray(ar));

}}

#### ****Question 3****

Given an unsorted binary array, you need to sort it in linear time – O(n) and constant space O(1).

Input: [0, 1, 1, 0, 1, 0, 0]

Output:[0, 0, 0, 0, 1, 1, 1]

**Best Solution**

Since the question clearly specifies that we sort in linear time and constant space, we simply would go directly to the solution.

The approach is this: Iterate through the array and each time you encounter a zero, move it to the beginning. At the same time, keep count of the number of zeroes seen so far. At the end of the iteration, then fill the remaining part of the array with ones.

This means that  the total time taken is linear.

The Python code is given below:

# \*\*\*\* SORT A BINARY ARRAY IN LINEAR TIME AND CONSTANT SPACE \*\*\*\*

def sortBinaryArray(a):

j = 0 # number of zeroes so far

i = 0 # loop variable

while(i != len(a)):

if(a[i] == 0):

a[j] = 0

j = j + 1

i = i + 1

for i in range(j+1, len(a)):

a[i] = 1

ar = [0, 1, 1, 0, 1, 0, 0]

print(ar)

sortBinaryArray(ar)

print(ar)

#### ****Question 4****

You are given an array of size n. One of the elements of the array appears twice in the array. Every other element appears once. You need to find the duplicate element.

For example

**Input:** {3, 6, 9, 0, 4, 7, 6}

**Output:** The duplicate is 6

**Solution 1: Brute Force Approach (Inefficient, O(n2))**

In this approach, you iterate through the array and compare each element with every other element in the array until you find a match.

The Java code is given below

public class FindDuplicateElement1 {

public static void findDuplicateElement(int[] A) {

for(int i = 0; i < A.length-1; i++) {

for(int j = i+1; j < A.length; j++ ) {

if(A[i] == A[j]) {

System.out.println("Duplicate at " + i + " and " + j);

return;

} }

}//end of for loop

System.out.println("No duplicates found");

}//end of function

//Main method to test the function

public static void main(String[] args) {

int ar[] = {4, 5, 9,4};

findDuplicateElement(ar);

}}

**Solution 2 – Using Hashing – O(n)**

In this solution, we iterate through the array once. For each iteration, we check if the value is in the hash table. If yes, then it’s a duplicate. If not, then we insert it into the hash table.

The Java code is given below:

public class FindDuplicateElement {

//Return indexes of duplicate

public static void findDuplicateElements(int A[]) {

Map<Integer,Integer> map = new HashMap<Integer, Integer>();

for(int i = 0; i < A.length; i++) {

if(map.containsKey(A[i])) {

System.out.println("Duplicates at " + map.get(A[i]) + " and " + i);

return;

}

map.put(A[i], i); }

System.out.println("No duplicate found");

}

public static void main(String[] args) {

int[] ar = {5, 7, 6, 5, 2};

findDuplicateElements(ar);

}}

**Solution 3 – Using Sorting**

In this approach, we first sort the array. Sorting takes logarithmic time. Then the duplicate element would then be side by side.

Then we loop through the array comparing each element with the immediate next element. Once there is a match then we return. But the challenge with this approach is that the index of the elements after sorting would change. So we only return the duplicate element and not the index.

The Java code is given below:

public class FindDuplicateElement2 {

public static void findDuplicateElement(int A[]) {

//First sort the array

Arrays.sort(A);

if(A.length <= 1) {

System.out.println("Unable to proceed");

return;

}

for(int i = 0; i < A.length-1; i++) {

if(A[i] == A[i+1]) {

System.out.println("Duplicate element is " + A[i]);

return;

}

System.out.println("No duplicated found"); }

public static void main(String[] args) {

int ar[] = {3,4, 5, 6, 6};

findDuplicateElement(ar); }

}

#### Question 5

You are given an unsorted array of integers. You  need to find the largest sub-array formed by consecutive numbers. The sub-array should only contain unique values. For example

**Input:**{2, 7, 2, 1, 4, 3, 5, 0}

**Output:**The largest subarray is from index 2 to 7

The sub-array is {2, 1, 4, 3, 5}

**Solution**

The approach is to iterate through the array. Then we keep track of the largest sub-array seen so far. We also make sure that the integers in the sub-array is consecutive.

For the sub-array elements to be consecutive, then:

the difference between the largest and smallest element must be equal to the length of the sub-array minus one

we check the the elements are unique track tracking the already visited element in another array.

The Java code is given below:

public class LargestSubarray {

//helper function to check if sub-array is consecutive

static boolean isConsecutive(int A[], int i, int j, int min, int max)

{

if(max - min != j - i) {

return false;

}

//track the visited elements

boolean visited[] = new boolean[j - i + 1];

for(int k = i; k <= j; k++) {

if(visited[A[k] - min]) {

return false;

}

visited[A[k] - min] = true;

} return true; }

public static void findMaxSubarray(int[] A) {

int len = 1;

int start = 0;

int end = 0;

//Loop through all the elements of the array

for(int i = 0; i < A.length - 1; i++) {

int min\_val = A[i];

int max\_val = A[i];

//Loop through the current sub-array

for(int j = i; j < A.length; j++)

{ min\_val = Math.min(min\_val, A[j]);

max\_val = Math.max(max\_val, A[j]);

//check the the elements are consecutive

if(isConsecutive(A, i, j, min\_val, max\_val)) {

if(len < max\_val - min\_val + 1) {

len = max\_val - min\_val + 1;

start = i;

end = j;

} } }

System.out.println("The largest subarray is from index " + start + " to " + end);

}//Test the program

public static void main(String[] args) {

int[] A = {2, 7, 2, 1, 4, 3, 5, 0};

findMaxSubarray(A); }}