**Algorithms-Binary Search**

**Objective :** To find an element in an sorted array

**Input:** A sorted array, arrA[] and an key

**Output :** Return true if element is found, else false.

**Approach:** The idea is to compare the middle element of array with the key, if key equal to the middle element , that’s it you have find your element, return true. If key is greater than the middle element, chuck out the first half of the array, you wont find your key in the first half and do the recursive search on the right half of the array and vice versa.

If(mid\_element==key)

return true;

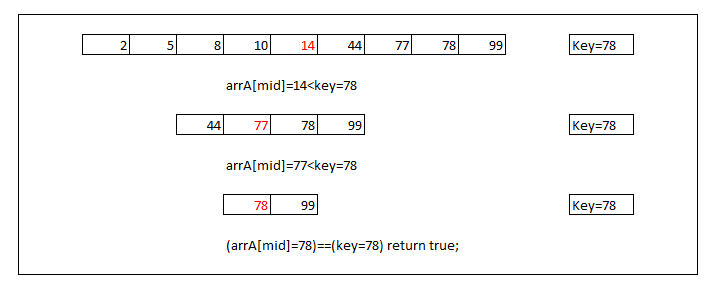
else if (mid<key)

do recursive search on the right half of the array.

else

do recursive search on the left half of the array.

**Time Complexity**: O(logN) –since we are eliminating half of the array with every comparison.



**Complete Code:**

**package** interviewQuestion;

**public** **class** BinarySearch {

**private** **int** [] arrA;

**private** **int** number;

**public** BinarySearch(**int** [] arrA){

**this**.arrA = arrA;

}

**public** Boolean Search(**int** low,**int** high, **int** number){

**if**(low>high){

**return** **false**;

}

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

**if**(arrA[mid]==number)**return** **true**;

**else** **if** (arrA[mid]>number) **return** Search(low,mid-1,number);

**else** **return** Search(mid+1,high,number);

}

**public** **static** **void** main(String args[]){

**int** [] a = {2,5,8,10,14,44,77,78,99};

**int** number = 99;

BinarySearch b = **new** BinarySearch(a);

System.*out*.println("The "+ number + " present in array a ??? :" + b.Search(0, a.length-1, number));

number = 76;

System.*out*.println("The "+ number + " present in array a ??? :" + b.Search(0, a.length-1, number));

}

}

Output:

The 99 present in array a ??? :true

The 76 present in array a ??? :false

Download link:

**Algorithms – Merge Sort**

**Objective :** To sort elements in an array

**Input:** A insorted array, arrA[].

**Output :** A sorted array.

**Approach:**

**Divide and Conquer:** In this approach we divide the main problems into smaller problems, solve them and merge the results to get the final result.

**How Divide and conquer works in Merge Sort:**

We divide the elements into two half’s by middle of the array. We solve the left half and right half recursively and merge the results.

**Merging:**

Once the sorting is done individually on both the half’s, our next task will be merge them. To merge we start with both the arrays at the beginning, pick the smaller one put into array and then compare the next elements and so on.

4

9

10

5

3

6

1

2

6

5

4

3

2

1

10

9

5

4

6

3

2

1

10

9

9

4

6

3

2

1

10

5

4

9

10

5

3

6

1

2

4

9

10

5

3

6

1

2

4

9

10

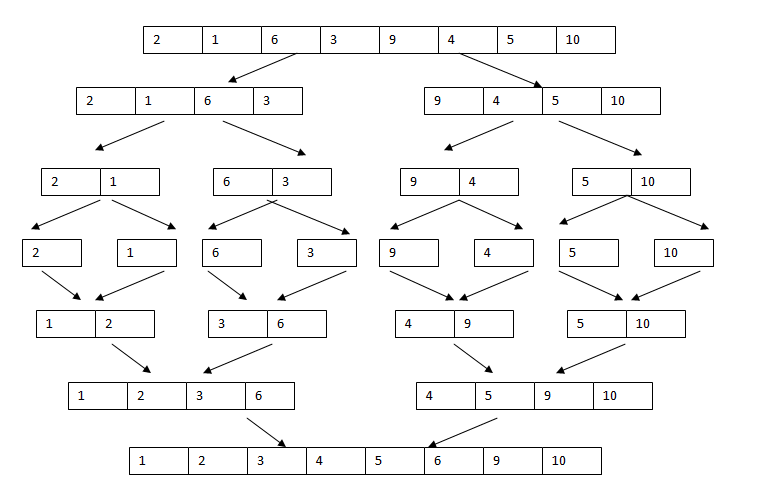
5

3

2

6

1



**Time Complexity : O(nlogn) { O(logn) for dividing and O(n) for merging.**

**Note: we can make merging more efficient by implementing these approaches**

**Using Auxiliary Array with copying data** – In this approach you wont create new array everytime for merging instead you create Auxiliary array. This will save memory for you.

**Alternate Merging Between Primary and Auxiliary Array:** This is the best approach for merging. You don’t copy the entire array to auxiliary array for merging instead you do alternate merging between main array and auxiliary array.

Below is the running time comparison between all three approaches

|  |  |  |  |
| --- | --- | --- | --- |
| **Data Size** | **Dynamic Memo Allocation Algo** | **Using Auxillary Array with copying data** | **Alternate Merging Between Primary and Auxillary Array** |
| 1 Million | 600-630 mili sec | 450-470 mili sec | 400-425 mili sec |
| 10 million | 6 secs | 4.2 secs | 2.3 secs |
| 100 million | 56 secs | 46 sec | 18 sec |

**You can find the implementation of all these approaches here –**

<https://github.com/SumitJainUTD/DataStructuresAlgos/tree/master/3%20Different%20Impl%20of%20Merge%20Sort>

**Complete Code:**

**package** interviewQuestion;

**public** **class** MergeSort {

**private** **int** arrSize;

**private** **int** [] arrAux;

**private** **int** [] arrInput;

**public** MergeSort(**int** [] arrInput){

**this**.arrInput = arrInput;

arrSize = arrInput.length;

arrAux = **new** **int** [arrSize];

}

**public** **int**[] mergeSorting(){

sort(0,arrSize-1);

**return** arrInput;

}

**public** **void** sort(**int** low, **int** high){

**if**(low<high){

**int** mid = low+((high-low))/2;

sort(low,mid);

sort(mid+1,high);

merge(low, mid, high);

}

}

**public** **void** merge(**int** low, **int** mid, **int** high){

//copy the entire array in the Auxilary array

**for**(**int** i=low;i<=high;i++){

arrAux[i] = arrInput[i];

}

**int** i = low;

**int** j = mid+1;

**int** k = low;

**while**(i<=mid && j<=high){

**if**(arrAux[i]<=arrAux[j]){

arrInput[k]=arrAux[i];

i++;

}

**else**{

arrInput[k]=arrAux[j];

j++;

}

k++;

}

**while**(i<=mid){

arrInput[k]=arrAux[i];

i++;

k++;

}

}

**public** **void** displayArray(**int** [] b){

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

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

}

}

**public** **static** **void** main(String[] args){

**int** [] a = {2,1,6,3,9,4,5,10};

MergeSort m = **new** MergeSort(a);

**int** [] b = m.mergeSorting();

m.displayArray(b);

}

}

Output :

1 2 3 4 5 6 9 10

**Algorithms - Hash Table Implementation**

**Objective :** To implement a Hash Table

**Input:** A set of pairs of keys and values

**Approach:**

* **Create a Hash Table**
  + Hashtable<Integer, String> ht = new Hashtable<Integer, String>();
* **Insert values in hash table using put(key,value)**
  + ht.put(key, value);
* **Get values from hash table using get(key)** 
  + ht.get(key);

|  |  |
| --- | --- |
| Key | Value |
| 1 | Sumit |
| 2 | Raghav |
| 3 | Rishi |

hashTable Object  
Advantage : The search time for any element is O(1) since it uses key to find an element so it takes constant time. But drawback is that it takes extra space.

**Complete Code:**

**package** interviewQuestion;

**import** java.util.Hashtable;

**public** **class** SimpleHashTable {

**int** [] a = **new** **int**[5];

String [] arrNames = **new** String[]{"Sumit","Jain","Raghav","Garg","Gaurav","Rishi"};

Hashtable<Integer, String> ht = **new** Hashtable<Integer, String>();

**public** **void** insertValues(){

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

ht.put(i+1,arrNames[i]);

}

}

**public** String getValue(**int** key){

**return** ht.get(key);

}

**public** **static** **void** main (String [] args){

SimpleHashTable sht = **new** SimpleHashTable();

sht.insertValues();

System.*out*.println("All values inserted");

System.*out*.println("Employee with ID 1 is "+ sht.getValue(1));

System.*out*.println("Employee with ID 3 is "+ sht.getValue(6));

}

}

Output:

All values inserted

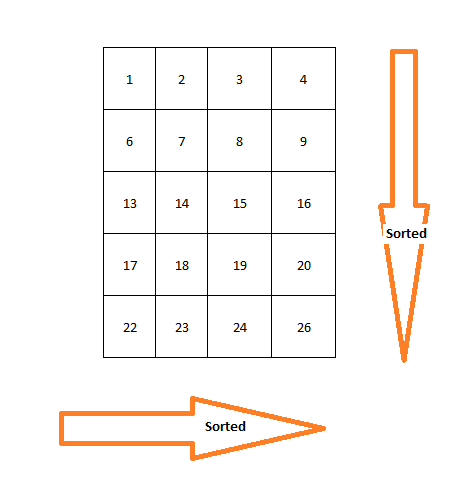
Employee with ID 1 is Sumit

Employee with ID 3 is Rishi

**Algorithms – Find an Element in 2 dimensional sorted array**

**Objective :** To **Find an Element in 2 dimensional array where rows and columns are sorted respectively.**

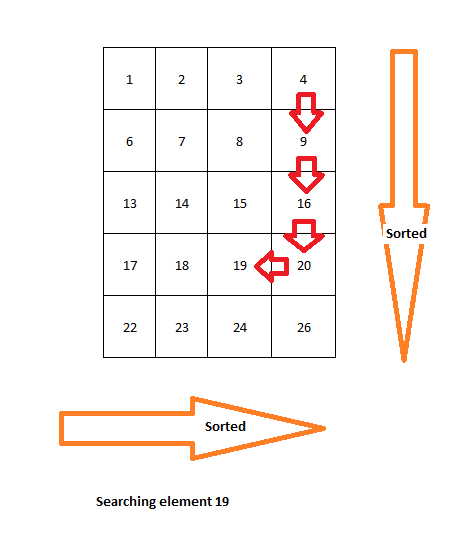
**Input:** A two dimensianl sorted array, arrA[][].



**Output :** True or false based on whether element exists and its location

**Approach:**

* Start from the left top corner, say ele;
* If ele>number -> move left
* If ele<number -> move right
* If you cant move further to find the number , return false



Complete Code:

**package** interviewQuestion;

//Here objective is find an element in two dimensional array

//all rows and columns of an array are sorted in ascending order respectively

**public** **class** FindElementInSorted2DArray {

**public** Boolean findElement(**int** [][] arrA, **int** number){

//start from the left top corner, say ele;

//if ele>number -> move left

//if ele<number -> move right

//if you cant move further to find the number , return false

**int** row = 0;

**int** col = arrA[1].length-1;

**boolean** numberFound = **false**;

System.*out*.print("The Movement : " );

**while**(numberFound==**false**){

**int** ele = arrA[row][col];

System.*out*.print(ele + "->" );

**if**(ele==number){

**return** **true**;

}

**else** **if**(ele>number)col--;

**else** **if** (ele<number)row++;

**if**(row>arrA[0].length-1||col<0)**return** **false**;

}

**return** **false**;

}

**public** **static** **void** main(String args[]){

**int** [][] a = {{1,2,3,4},{6,7,8,9},{13,14,15,16},{17,18,19,20},{22,23,24,26}};

**int** number1 = 15;

**int** number2 = 5;

**int** number3 = 19;

**int** number4 = 25;

FindElementInSorted2DArray f = **new** FindElementInSorted2DArray();

System.*out*.println("The "+ number1 + " present in 2D array a ??? :" + f.findElement(a, number1));

System.*out*.println("The "+ number2 + " present in 2D array a ??? :" + f.findElement(a, number2));

System.*out*.println("The "+ number3 + " present in 2D array a ??? :" + f.findElement(a, number3));

System.*out*.println("The "+ number4 + " present in 2D array a ??? :" + f.findElement(a, number4));

}

}

Output:

The Movement : 4->9->16->15->The 15 present in 2D array a ??? :true

The Movement : 4->9->8->7->6->The 5 present in 2D array a ??? :false

The Movement : 4->9->16->20->19->The 19 present in 2D array a ??? :true

The Movement : 4->9->16->20->The 25 present in 2D array a ??? :false

**Algorithms – Find a Missing Number From a Sequence of Consecutive Numbers**

**Objective :** Find a Missing Number From a Sequence of Consecutive Numbers

**Input:** Array, arrA[] with a missing number and Range

**Output :** missing number

**Approach:**

* Approach is very simple, Add all the given numbers say S
* Calculate sum of N numbers by formula n(n+1)/2 , say N
* Find missing number m = N-S

Example : suppose array given is {1,2,3,4,5,6,8,9,10} and range is 10.

So N will be sum of 1 to 10 = 10(10+1)/2 = 55

S will be sum of all the array elements which is = 48

So missing number will be = 55-48 = 7

**Complete Code:**

**package** interviewQuestion;

//find the missing number from the sequence of consecutive number

//Approach is very simple, Add all the given numbers say S

//Calculate sum of N numbers by formula n(n+1)/2 , say N

//Find missing number m = N-S

**public** **class** FindMissingNumber {

**int** Sum;

**int** Sum\_N;

**public** **int** missingNumber(**int** [] arrA, **int** size){

Sum\_N = size\*(size+1)/2;

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

Sum +=arrA[i];

}

**return** Sum\_N-Sum;

}

**public** **static** **void** main(String args[]){

**int** [] arrA = {1,2,3,4,5,7,8,9,10};

System.*out*.println("Missing number is :" + (**new** FindMissingNumber()).missingNumber(arrA,10));

}

}

Output :

Missing number is :6

**Algorithms – Find two Missing Numbers in a Sequence of Consecutive Numbers**

**Objective :** Find two Missing Numbers in a Sequence of Consecutive Numbers

**Input:** Array, arrA[] with two missing numbers and Range

**Output :** Two missing numbers

**Approach:**

* Approach is very simple, Add all the given numbers say S
* Calculate sum of N numbers by formula n(n+1)/2 , say N
* Find sum of two missing numbers a+b = N-S
* Now take the product of all given numbers say P
* Now take the product of N numbers , say Np;
* Find the product of two missing numbers ab = Np-P
* Now we have a+b and ab , we can easily calculate a and b

**Example :**

Given array : {10,2,3,5,7,8,9,1}; Range : 10

N (Sum of 1 to 10 ) = 55

S (Sum of given elements ) = 45

a+b = 10------------------------------------(1)

Np(Product of 1 to 10) = 3628800

P(Product of given elements) = 151200

So a\*b = 24---------------------------------(2)

Now we have two equations and two variables, if we solve we will get values 6 and 4.

**Complete Code:**

**package** interviewQuestion;

//find the two missing numbers from the sequence of consecutive number

//Approach is very simple, Add all the given numbers say S

//Calculate sum of N numbers by formula n(n+1)/2 , say N

//Find sum of two missing numbers a+b = N-S

//Now take the product of all given numbers say P

//now take the product of N numbers , say Np;

//find the product of two missing numbers ab = Np-P

//now we have a+b and ab , we can easily calculate a and b

**public** **class** FindTwoMissingNumbers {

**int** Sum;

**int** SumN;

**int** P=1;

**int** Np=1;

**int** a,b;

**public** **int** [] missingNumbers(**int** [] arrA, **int** range){

SumN = range\*(range+1)/2;

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

Sum +=arrA[i];

}

**int** s= SumN-Sum;

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

P \*=arrA[i];

}

**for**(**int** i=1;i<=range;i++){

Np \*=i;

}

**int** product = Np/P;

// System.out.println(product);

**int** diffSqr = (**int**)Math.*sqrt*(s\*s-4\*product); // (a-b)^2 = (a+b)^2-4ab

a = (s+diffSqr)/2;

b= s-a;

**int** [] result = {a,b};

**return** result;

}

**public** **static** **void** main(String args[]){

**int** [] arrA = {10,2,3,5,7,8,9,1};

FindTwoMissingNumbers f = **new** FindTwoMissingNumbers();

**int** [] results = f.missingNumbers(arrA, 10);

System.*out*.println("Missing numbers are :" + results[0] + " and " + results[1]);

}

}

**Output:**

Missing numbers are :6 and 4

**Algorithms – Find a peak element in a Given Array**

**Objective :** Find a peak element in a Given Array, where peak element is the one.

**Peak Element:** peak element is the element which is greater than or equal to both of its neighbors.

**Input:** Array, arrA[] .

**Output:** A peak element and its index

**Approach:**

A simple approach is to do a linear scan to a array and using few variables we can find a peak element. But the Time Complexity will be O(n) but real question is, Can we do better?

The Answer is yes, by using Binary Search techniques.

* If middle element is the peak element, return it
* If middle element is smaller than its left element , we will get our peak element on the left half
* If middle element is the smaller than its right element, we will our peak element on the right.

**Notes:**

1. If array has all the same elements, every element is a peak element.
2. Every array has a peak element.
3. Array might have has many peak elements but we are finding only one.
4. If array is in ascending or descending order then last element or the first element of the array will be the peak element respectively.

**Complete Code:**

**package** interviewQuestion;

//we will use binary search techniques

//if middle element is the peak element, return it

//if middle element is smaller than its left element , we will get our peak element on the left half

//if middle element is the smaller than its right element, we will our peak element on the right.

**public** **class** PeakElement {

**public** **int** peak(**int** [] arrA,**int** low, **int** high, **int** size){

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

**if**((mid==0||arrA[mid]>=arrA[mid-1]) && (arrA[mid]>=arrA[mid+1]||mid==size-1)){

**return** mid;

}

**else** **if**(mid>0 && arrA[mid]<arrA[mid-1]) **return** peak(arrA,low,mid-1,size);

**else** **return** peak(arrA,mid+1,high,size);

}

**public** **static** **void** main(String args[]){

PeakElement pe = **new** PeakElement();

**int** arrA[] = { 1,2,3,4,0,1,5,4,3,2,1};

**int** peakEle = pe.peak(arrA, 0, arrA.length-1, arrA.length);

System.*out*.println("Peak Element is found at index [" + peakEle +"] = "+ arrA[peakEle]);

}

}

**Output:**

Peak Element is found at index [6] = 5

**Algorithms – Find Whether Given String is palindrome or Not.**

**Objective :** Find Whether Given String is palindrome or Not.

**Input:** A String,

**Output:** true or false on whether string is palindrome or not

**Approach:**

* Use recursive approach
* Compare first and last characters if they are not same- return false
* If they are same make, remove the first and last characters and make a recursive call.

**Example:**

Jain niaJ => compare ‘J’ with ‘J’ =>returns true

ain nia => compare ‘a’ with ‘a’ =>returns true

in ni => compare ‘i’ with ‘i’ =>returns true

n n => compare ‘n’ with ‘n’ =>returns true

string length <2 => returns true

**Complete Code:**

**package** interviewQuestion;

//Use recursive approach

//Compare first and last characters if they are not same- return false

//If they are same make, remove the first and last characters and make a recursive call.

**public** **class** Palindrome {

**public** Boolean isPalindrome(String strX){

**if**(strX.length()<2) **return** **true**;

**if**(strX.charAt(0)==strX.charAt(strX.length()-1)){

isPalindrome(strX.substring(0, strX.length()-2));

}

**else** **return** **false**;

**return** **true**;

}

**public** **static** **void** main(String args[]){

String S1 = "Sumit";

String S2 = "SumuS";

String S3 = "ABCDEFGHGFEDCBA";

String S4 = "Jain niaJ";

Palindrome p = **new** Palindrome();

System.*out*.println("Is "+ S1 + " Palindrome ??? :" + p.isPalindrome(S1));

System.*out*.println("Is "+ S2 + " Palindrome ??? :" + p.isPalindrome(S2));

System.*out*.println("Is "+ S3 + " Palindrome ??? :" + p.isPalindrome(S3));

System.*out*.println("Is "+ S4 + " Palindrome ??? :" + p.isPalindrome(S4));

}

}

**Output:**

Is Sumit Palindrome ??? :false

Is SumuS Palindrome ??? :true

Is ABCDEFGHGFEDCBA Palindrome ??? :true

Is Jain niaJ Palindrome ??? :true

**Algorithms – Find Whether Given the Sequence of parentheses are well formed.**

**Objective:** Find Whether Given the Sequence of parentheses are well formed.

**Input:** A String contains a sequence of parentheses

**Output:** true or false on whether parentheses are well formed or not

**Approach:**

* Idea is to have two counters, one for open parentheses '{' and one for close '}'
* Read one character at a time and increment one of the counters
* If any given point of time count of close parentheses is greater than the open one, return false
* If at the end both counters are equal, return true

**Example: { { } { } } – Well formed**

**{ { } { = Not well formed**

**Complete Code:**

**package** interviewQuestion;

**public** **class** WellFormedParentheses {

**public** Boolean isWellFormed(String strParentheses){

**if**(strParentheses==**null**){

**return** **false**;

}

//Idea is to have two counters, one for open parentheses '{' and one for close '}'

//Read one character at a time and increment one of the counters

//If any given point of time count of close parentheses is greater than the open one, return false

//If at the end both counters are equal, return true

**int** openParenCounter=0;

**int** closeParenCounter=0;

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

**char** x = strParentheses.charAt(i);

**if**(x=='{') openParenCounter++;

**else** **if**(x=='}') closeParenCounter++;

**if**(closeParenCounter>openParenCounter){

**return** **false**;

}

}

**if**(openParenCounter==closeParenCounter)**return** **true**;

**else** **return** **false**;

}

**public** **static** **void** main(String args[]){

String x1 = "{{{{}}}}{}{}{}{}{}{}{}{}{}{}{{{}}}";

String x2 = "{{{{}}}}{}{}{}{{}{}{}{}{}{}{}{{{}}}";

String x3 = "{}{";

String x4 = "}{";

String x5 = "{{{{{{{{}}}}}}}}";

WellFormedParentheses w = **new** WellFormedParentheses();

System.*out*.println("Is "+ x1 + " well formed ??? :" + w.isWellFormed(x1));

System.*out*.println("Is "+ x2 + " well formed ??? :" + w.isWellFormed(x2));

System.*out*.println("Is "+ x3 + " well formed ??? :" + w.isWellFormed(x3));

System.*out*.println("Is "+ x4 + " well formed ??? :" + w.isWellFormed(x4));

System.*out*.println("Is "+ x5 + " well formed ??? :" + w.isWellFormed(x5));

}

}

**Output**

Is {{{{}}}}{}{}{}{}{}{}{}{}{}{}{{{}}} well formed ??? :true

Is {{{{}}}}{}{}{}{{}{}{}{}{}{}{}{{{}}} well formed ??? :false

Is {}{ well formed ??? :false

Is }{ well formed ??? :false

Is {{{{{{{{}}}}}}}} well formed ??? :true

**Algorithms – Rearrange Positive and Negative Numbers of Array On Each Side in O(nlogn)**

**Objective:** Rearrange Positive and Negative Numbers of an Array so that one side you have positive numbers and other side with negative Integers without changing their respective order.

Example : Input : 1 -2 3 -4 5 -6 7 -8 9 -10

ReArranged Output : -2 -4 -6 -8 -10 1 3 5 7 9

**Input:** An array with positive and negative numbers

**Output:** Modified array with positive numbers and negative numbers are on each side of the array.

**Approach:**

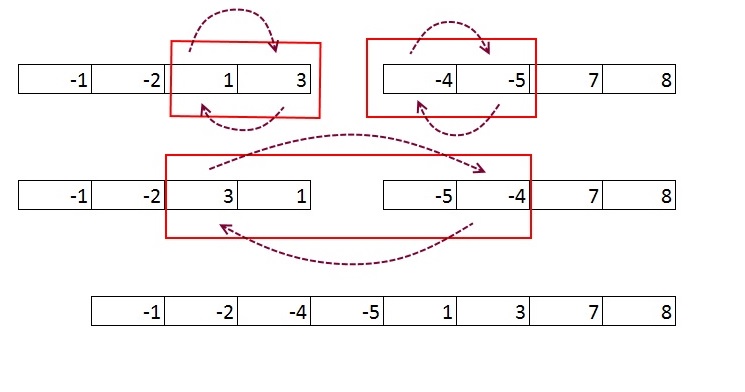
**Method 1.** One naive approach is to have another array of same size and navigate the input array and one scan place all the negative numbers to the new array and in second scan place all the positive numbers to new array. Here the Space complexity will be O(n). We have a better solution which can solve this in O(1) space.

**Method 2: Divide and Conquer**

* This approach is similar to merge sort.
* Divide the array into two half, Solve them individually and merge them.
* Tricky part is in merging.

Merging: (Negative on left side and positives on the right side)

* Navigate the left half of array till you won't find a positive integer and reverse the array after that point.(Say that part is called 'A')
* Navigate the right half of array till you won't find a negative integer and reverse the array after that point. (Say that part is called 'B')
* Now reverse the those parts from both the array (A and B), See example for better explanations



Complete Code:

**package** interviewQuestion;

**public** **class** RearrageArrayPositiveNegative {

**int** [] arrA;

**public** RearrageArrayPositiveNegative(**int** [] arrA){

**this**.arrA = arrA;

}

**public** **void** divideGroups(**int** low, **int** high){

**if**(low>=high) **return**;

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

divideGroups(low, mid);

divideGroups(mid+1, high);

merge(low,mid,high);

}

**public** **void** merge(**int** low, **int** mid, **int** high){

**int** l = low;

**int** k = mid+1;

**while**(l<=mid && arrA[l]<=0)l++;

**while**(k<=high && arrA[k]<=0)k++;

reverse(l,mid);

reverse(mid+1,k-1);

reverse(l,k-1);

}

**public** **void** reverse(**int** x, **int** y){

**while**(y>x){

**int** temp = arrA[x];

arrA[x]=arrA[y];

arrA[y]=temp;

x++;

y--;

}

}

**public** **void** display(){

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

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

}

}

**public** **static** **void** main(String args[]){

**int** [] a = {1,-2,3,-4,5,-6,7,-8,9,-10};

RearrageArrayPositiveNegative r = **new** RearrageArrayPositiveNegative(a);

System.*out*.print("Input : ");r.display();

r.divideGroups(0, a.length-1);

System.*out*.println("");

System.*out*.print("ReArranged Output : ");r.display();

}

}

Output:

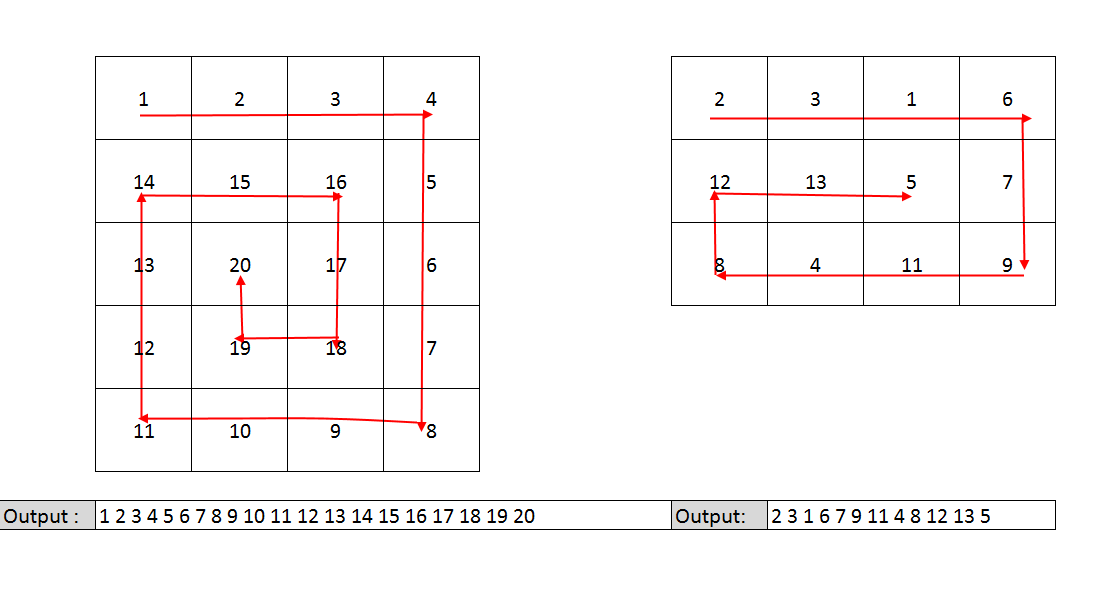
Input : 1 -2 -3 -4 5 -6 7 -8 9 -10 -11 -12 20

ReArranged Output : -2 -3 -4 -6 -8 -10 -11 -12 1 5 7 9 20

**Algorithms – Print All Elements of Two Dimensional Array in Spiral**

**Objective:** Print all the elements of two dimensional array in spiral.

Example :



**Input:** Two dimensional array

**Output:** All array elements printed in spiral.

**Approach:**

* Start printing from first row.
* Print row and columns, forward and backward alternatively
* With every iteration of (either row or column), reduce the size of an row or column by 1
* Call recursively

Complete Code:

**package** interviewQuestion;

**public** **class** Print2DArrayInSpiral {

**public** **int** arrA[][] = { { 1, 2, 3, 4, 5 }, { 18, 19, 20, 21, 6 },

{ 17, 28, 29, 22, 7 }, { 16, 27, 30, 23, 8 },

{ 15, 26, 25, 24, 9 }, { 14, 13, 12, 11, 10 } };

**public** **int** printSpiral(**int** row\_S, **int** row\_E, **int** col\_S, **int** col\_E,**boolean** reverse, **boolean** rowPrint) {

**if** (row\_S > row\_E && col\_S>col\_E) {

**return** 1;

}

**if** (rowPrint == **false**) {

**if** (reverse == **false**) {

**for** (**int** i = col\_S; i <= col\_E; i++) {

System.*out*.print(" " + arrA[row\_S][i]);

}

}

row\_S++;

rowPrint = **true**;

reverse = **false**;

}

**if** (rowPrint == **true**) {

**if** (reverse == **false**) {

**for** (**int** i = row\_S; i <= row\_E; i++) {

System.*out*.print(" " + arrA[i][col\_E]);

}

}

col\_E--;

rowPrint = **false**;

reverse = **true**;

}

**if** (rowPrint == **false**) {

**if** (reverse == **true**) {

**for** (**int** i = col\_E; i >= col\_S; i--) {

System.*out*.print(" " + arrA[row\_E][i]);

}

}

row\_E--;

rowPrint = **true**;

reverse = **true**;

}

**if** (rowPrint == **true**) {

**if** (reverse == **true**) {

**for** (**int** i = row\_E; i >= row\_S; i--) {

System.*out*.print(" " + arrA[i][col\_S]);

}

}

col\_S++;

rowPrint = **false**;

reverse = **false**;

}

printSpiral(row\_S, row\_E, col\_S, col\_E, reverse, rowPrint);

**return** 0;

}

**public** **static** **void** main(String args[]) {

Print2DArrayInSpiral p = **new** Print2DArrayInSpiral();

p.printSpiral(0, 5, 0, 4, **false**, **false**);

}

}

Output:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

**Algorithms – Find Out Whether String Contains All The Unique Characters**

**Objective:** find out whether in a given string contains all the unique characters

**Input:** A String

**Output:** True or false based upon whether string contains all the unique characters or not

**Approach:**

**Method 1.**

**When characters are not ASCII but could be anything alphabets or special characters**

* Create a boolean array of size 256, and put false at every index.
* Navigate the input string one character at a time, say 'char a'
* Check array position of array[a], if it is false, make it true.
* If it is false, update it as true.

**Method 2:**

Sort the array and do the linear scan to find out whether string contains unique elements or not.

**Complete Code:**

**package** interviewQuestion;

//This Program is to find out whether String contains all the unique characters

//With out using any additional data structures

**public** **class** UniqueCharString {

**private** String ip;

**public** UniqueCharString(String ip){

**this**.ip = ip;

}

//method 1 : When characters are not ASCII but could be anything alphabets or special characters

//Time Complexity : O(n)

//Space Complexity : O(1)

//

**public** Boolean UniChars(){

Boolean [] bln = **new** Boolean[256];

**for**(**int** i=0;i<256;i++){

bln[i]=**false**;

}

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

**char** a = ip.charAt(i);

**if**(bln[a]==**true**){

**return** **false**;

}

**else**{

bln[a]=**true**;

}

}

**return** **true**;

}

//method 2: Sort the array and do the linear scan to find out whether string

//contains unique elements or not

//Time Complexity : O(nLogn)

//Space Complexity : O(n)

**public** Boolean UniqueCharSorting(){

**char** [] a = ip.toCharArray();

java.util.Arrays.*sort*(a);

String tmp = **new** String(a);

**for**(**int** i=1;i<tmp.length();i++){

**char** t = tmp.charAt(i-1);

**if**(t==tmp.charAt(i)){

**return** **false**;

}

}

**return** **true**;

}

**public** **static** **void** main(String args[]){

String a = "Sumit\_Jain";

UniqueCharString u = **new** UniqueCharString(a);

System.*out*.println("Method 1 : Does String ' " + a +" ' has all unique characters :" + u.UniChars());

a = "Sumit";

u = **new** UniqueCharString(a);

System.*out*.println("Method 1 : Does String ' " + a +" ' has all unique characters :" + u.UniChars());

a = "Sumit\_Jain";

u = **new** UniqueCharString(a);

System.*out*.println("Method 2 : Does String ' " + a +" ' has all unique characters :" + u.UniqueCharSorting());

a = "Sumit";

u = **new** UniqueCharString(a);

System.*out*.println("Method 2 : Does String ' " + a +" ' has all unique characters :" + u.UniqueCharSorting());

}

}

Output:

Method 1 : Does String ' Sumit\_Jain ' has all unique characters :false

Method 1 : Does String ' Sumit ' has all unique characters :true

Method 2 : Does String ' Sumit\_Jain ' has all unique characters :false

Method 2 : Does String ' Sumit ' has all unique characters :true

**Algorithms – Print All The Permutations Of a String**

**Objective:** find out whether in a given string contains all the unique characters

**Input:** A String

**Output:** Print all the permutations of a string

Example:

Input : abc

**Output:** abc acb bac bca cba cab

**Approach:**

**Algorithms – Quick Sort Implementation**

**Objective:** To sort an array in increasing or decreasing order using Quick Sort.

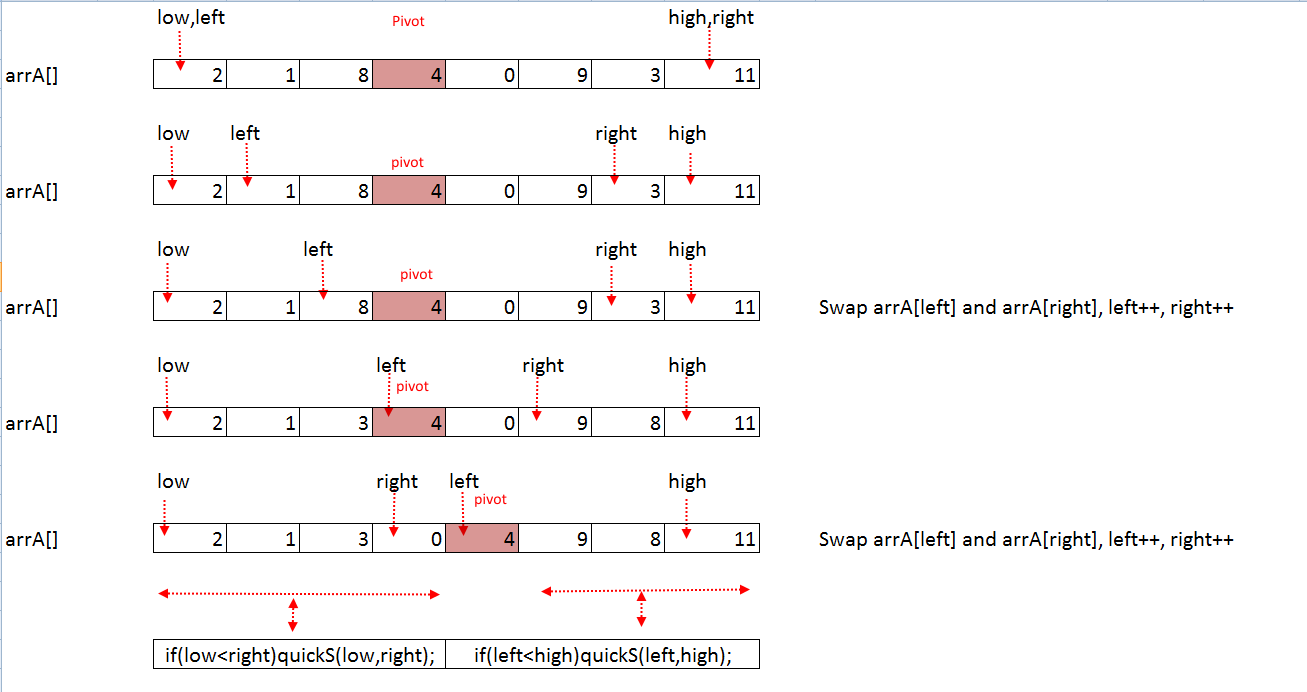
**Input:** An Array arrA[]

**Output:** A sorted array.

**Approach:**

* Choose any element from the array and call it as pivot element, Example here we have selected middle element as pivot
* Place all the elements smaller than pivot in the left side of pivot.
* Place all the elements greater than pivot in the right side of pivot.
* Sort left side and right side recursively.

Example:



**Complete Code:**

**package** interviewQuestion;

**public** **class** QuickSort

{

**private** **int** [] arrA;

**public** QuickSort(**int** [] arrA){

**this**.arrA = arrA;

}

**public** **void** quickS(**int** low, **int** high){

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

**int** left = low;

**int** right = high;

**int** pivot = arrA[mid]; //select middle element as pivot

**while**(left<=right){

**while**(arrA[left]<pivot) left++;//find element which is greater than pivot

**while**(arrA[right]>pivot)right--;////find element which is smaller than pivot

//System.out.println(arrA[left] + " " + pivot + " " + arrA[right] );

//if we found the element on the left side which is greater than pivot

//and element on the right side which is smaller than pivot

//Swap them, and increase the left and right

**if**(left<=right){

**int** temp = arrA[left];

arrA[left] = arrA[right];

arrA[right]= temp;

left++;

right--;

}

}

//Recursion on left and right of the pivot

**if**(low<right)quickS(low,right);

**if**(left<high)quickS(left,high);

}

**public** **void** display(){

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

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

}

}

**public** **static** **void** main (String[] args) **throws** java.lang.Exception

{

// your code goes here

**int** a [] = { 2,1,8,4,0,9,3,11};

QuickSort i = **new** QuickSort(a);

System.*out*.print("UnSorted : ");

i.display();

i.quickS(0,a.length-1);

System.*out*.println("");

System.*out*.print("Quick Sorted : ");

i.display();

}

}

Output:

UnSorted : 2 1 8 4 0 9 3 11

Quick Sorted : 0 1 2 3 4 8 9 11

Time Complexity : O(n^2) worst case and O(nlogn) average case analysis

Space Complexity : O(1)