Java Array Interview Questions

codespaghetti.com/array-interview-question

Arrays

Java Array Interview Questions, Array Algorithm Questions and Java Array Programs to help you ace your next Job interview.

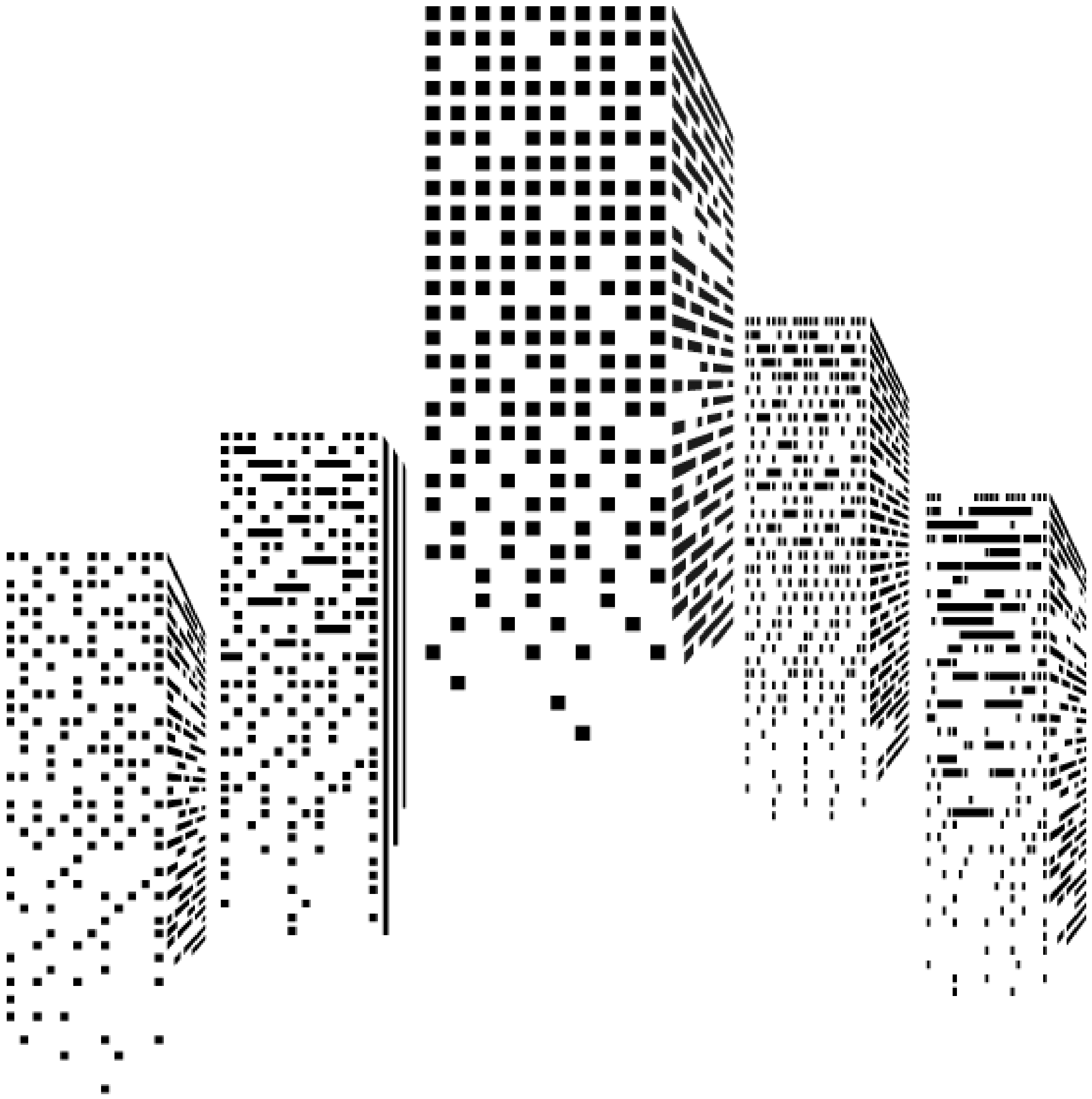


Table of Contents:

Importance of Arrays in Programming Language

You can think, you can read, you can write, you can study, you can dream, you can hope, you can wish or even you can memorize everything about programming languages.

*But you can still never be able to succeed in programming interviews. Until you know**"THE ARRAYS"** from inside out.*
Understand: You need to get really good at manipulating Arrays before you can crack any programming interviews.
You are your own worst enemy if you go to an interview without preparing Java array interview questions.
And
There are WAY too many candidates who gets this wrong.
But
Mastering arrays is simple and easy all you have to do is to understand three basic elements of an array

- How arrays works internally [Java arrays]
- How to manipulate arrays using search and sorting algorithms
- The performance of arrays and its comparisons with other data structures like (Linked lists, array lists and hash maps)

If you're serious about succeeding in your next interview. You need to be verysystematic with how you learn and master arrays.
In the following guide I have collected and categorized over 50 ARRAY interview Questions , Programs and Algorithms, asked in interviews from large to small companies all over the world.
Take your time to go through them and you will increase your chances ofsuccess in

interviews.
But before we continue. Do you know what are the top 5 array questions asked in interviews? If not then checkout the following graph.

Top 5 Array Interview Questions

Array Internal Structure:

In programming interviews(Onsite and Phone interviews) you will be asked a variety of questions about Arrays. The most important one's will be about the basic structure of the array.

The purpose of these questions is to get to know the candidate's understanding about arrays.

How array works internally?

What are the Basic functions of arrays?

And how to use this knowledge to solve real world problems using arrays.

Here are the most important java array interview questions aimed at the internal workings of arrays.

Java Array Facts:

- Arrays are objects which can store collection of same type of elements
- An array has a certain number of elements in a fixed order
- Accessing an invalid array index causes an exception
- Arrays are objects,and are created on the heap, not the stack
- Big-O Complexity of operations **Access** Θ(1), **Search** Θ(n), **Insertion** Θ(n), **Deletion** Θ(n).

Question: What is Array in Java? or how does array works internally?

Answer: Array is a data structure which store multiple variables of the same type. It can hold primitive types as well as object references.

In simple words an array is used to store a collection of data, array is declared like this

```
dataType[] arrayRefVariable = new dataType[arraySize];
```

These are some important facts of array

- Arrays are of fixed length(Static length)
- Array can hold same type of data
- Arrays can even hold the reference variables of other objects
- Arrays are objects,and are created on the heap, not the stack

Question: Can you change size of Array in Java once created?

Answer: Arrays are static, which means that we cannot change the size of array once created. If you need dynamic array, consider using ArrayList class, which can resize itself.

Question: Can you use Generics with Array in Java?

Answer: No, Generics cannot be used with array. That is why sometime List is better choice over array in Java.
Question : Why do length of array is referred as length - 1 ?

Values of an array are accessed by using index and by default array starts from index zero.

so if array lenght is 10 then first value 1 is actually placed at index zero and last value 10 is placed at index 9.so.

We always subtract -1 from array length to point to the last index location.

Question: Difference between Array Index Out OF Bounds and ArrayStoreException?

ArrayIndexOutOfBoundsException: is thrown when the code tries to access an invalid index for a given array e.g. negative index or higher index than length - 1. While,

ArrayStoreException: is thrown when you try to store an element of another type then the type of array.e.g; if array is of type int and we try to add element of type String.

Question: Can you pass the negative number as an Array size?

No. You can’t pass the negative integer as an array size. If you pass, there will be no compile time error but you will get NegativeArraySizeException at run time.

```
public class MainClass
{
    public static void main(String[] args)
    {
        int[] array = new int[-5];  //No compile time error

        //but you will get java.lang.NegativeArraySizeException at run time
    }
}
```

Question: What is an anonymous array in Java? Give example?

Answer: Anonymous array is an array without reference.

Here is an example

```
 public static void main(String[] args) {
//Anonymous Array creation
  System.out.println(new int[]{6,7, 3, 1, 9}.length);
  System.out.println(new int[]{91, 34, 55, 24, 31}[1]);

   }
}
```

Question: Can you assign an Array of 100 elements to an array of 10 elements?

Answer: Yes, In Java an Array of 100 elements can be assigned to an Array of 10 elements. The only conditions is that, they should be of same type. Because while assigning values the compiler checks only type of the array and not the size. **Here is code example written in Java.**

```
public class ArrayCopyClass
{
    public static void main(String[] args)
    {
        int[] arrayWitTen = new int[10];

        int[] arrayWith100 = new int[100];
        arrayWitTen=arrayWith100;
      }
}
```

Question: What are the different ways of copying an array into another array in Java?

Answer: There are four methods available in java to copy an array.

- Using for loop
- Using Arrays.copyOf() method
- Using System.arraycopy() method
- Using clone() method

Question: What are jagged arrays in java? Give example?

Answer: Jagged arrays in java are type of arrays which have different length.

And Jagged arrays are multidimensional Arrays.

Here is an example of jagged Arrays

```
public class JaggedArraysExampleInJava
{
    public static void main(String[] args)
    {
        //One Dimensional Array with length 3
        int[] OneDimensionalArray3 = {1, 2, 3};

        //One Dimensional Array with length 4
        int[] oneDimensionalArray4 = {4, 5, 6, 7};

        //One Dimensional Array with length 5
        int[] oneDimensionalArray5 = {8, 9, 10, 11, 12};

        //Jagged Two Dimensional Array
        int[][] twoDimensionalArray = {OneDimensionalArray3, oneDimensionalArray4,
oneDimensionalArray5};

        //Printing elements of Two Dimensional Array
        for (int i = 0; i < twoDimensionalArray.length; i++)
        {
            for (int j = 0; j < twoDimensionalArray[i].length; j++)
            {
                System.out.print(twoDimensionalArray[i][j]+”");
            }
            System.out.println();
        }
    }
}
```

Question: How do you check the equality of two arrays in java?

Answer: You can use Arrays.equals() method to compare one dimensional arrays and to compare multidimensional arrays, use Arrays.deepEquals() method.

Question: Where does Java Array is stored in memory?

Answer: Array is created in heap space of JVM memory. Since array is object in Java.

Even if you create array locally inside a method or block, object is always allocated memory from heap.

Question: Which access modifiers can be used to declare Arrays in Java?

Answer: In Java Array can be declared as **PRIVATE, PUBLIC, PROTECTED**, and without any modifiers.

Following table gives an overview about the accessibility of array for different access modifiers.

```
            | Class | Package | Subclass | Subclass | World
            |       |         |(same pkg)|(diff pkg)|
────────────+───────+─────────+──────────+──────────+────────
public      |   +   |    +    |    +     |     +    |   +
────────────+───────+─────────+──────────+──────────+────────
protected   |   +   |    +    |    +     |     +    |   o
────────────+───────+─────────+──────────+──────────+────────
no modifier |   +   |    +    |    +     |     o    |   o
────────────+───────+─────────+──────────+──────────+────────
private     |   +   |    o    |    o     |     o    |   o

+ : accessible
o : not accessible
```

Question: Are Array thread safe in Java?

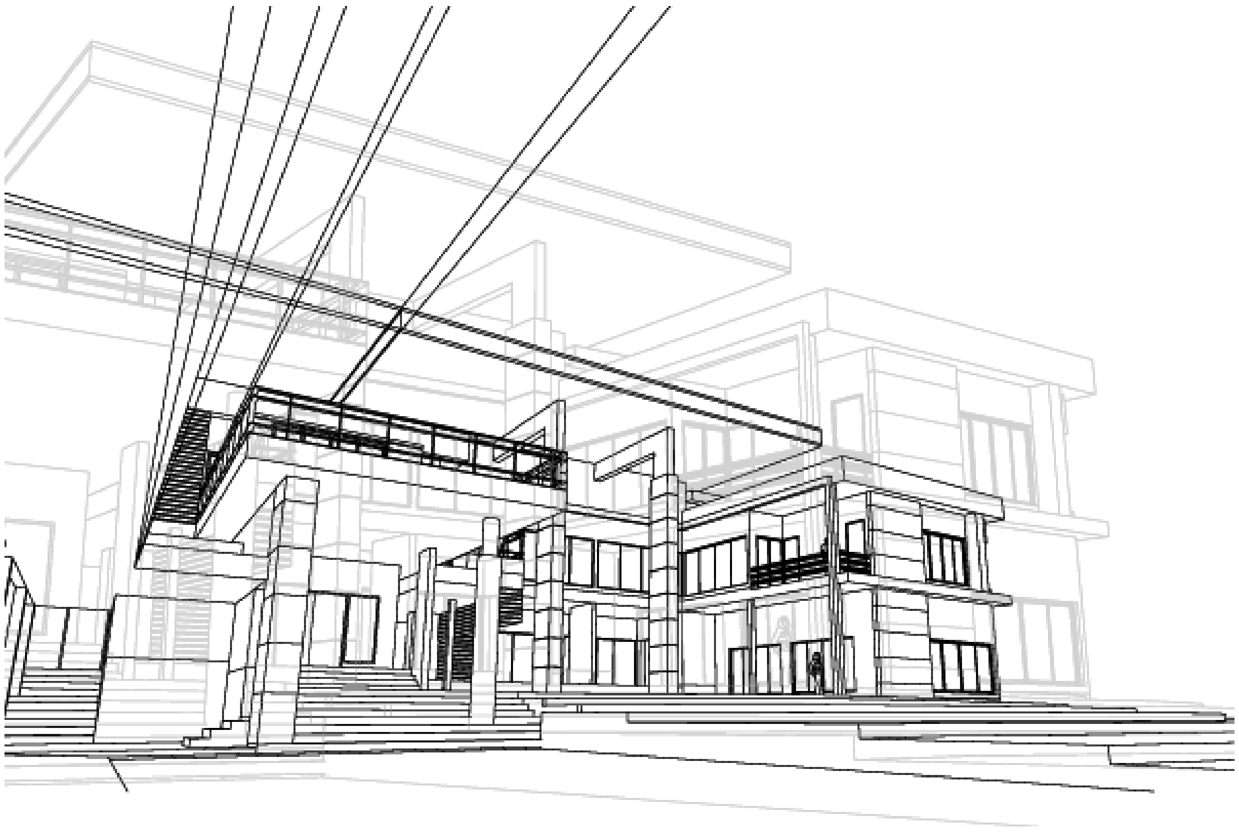
In general reading from array is Thread-Safe Operation but modifying an array its not.

Question: What is time complexity of different Array operations in terms of big o notation?

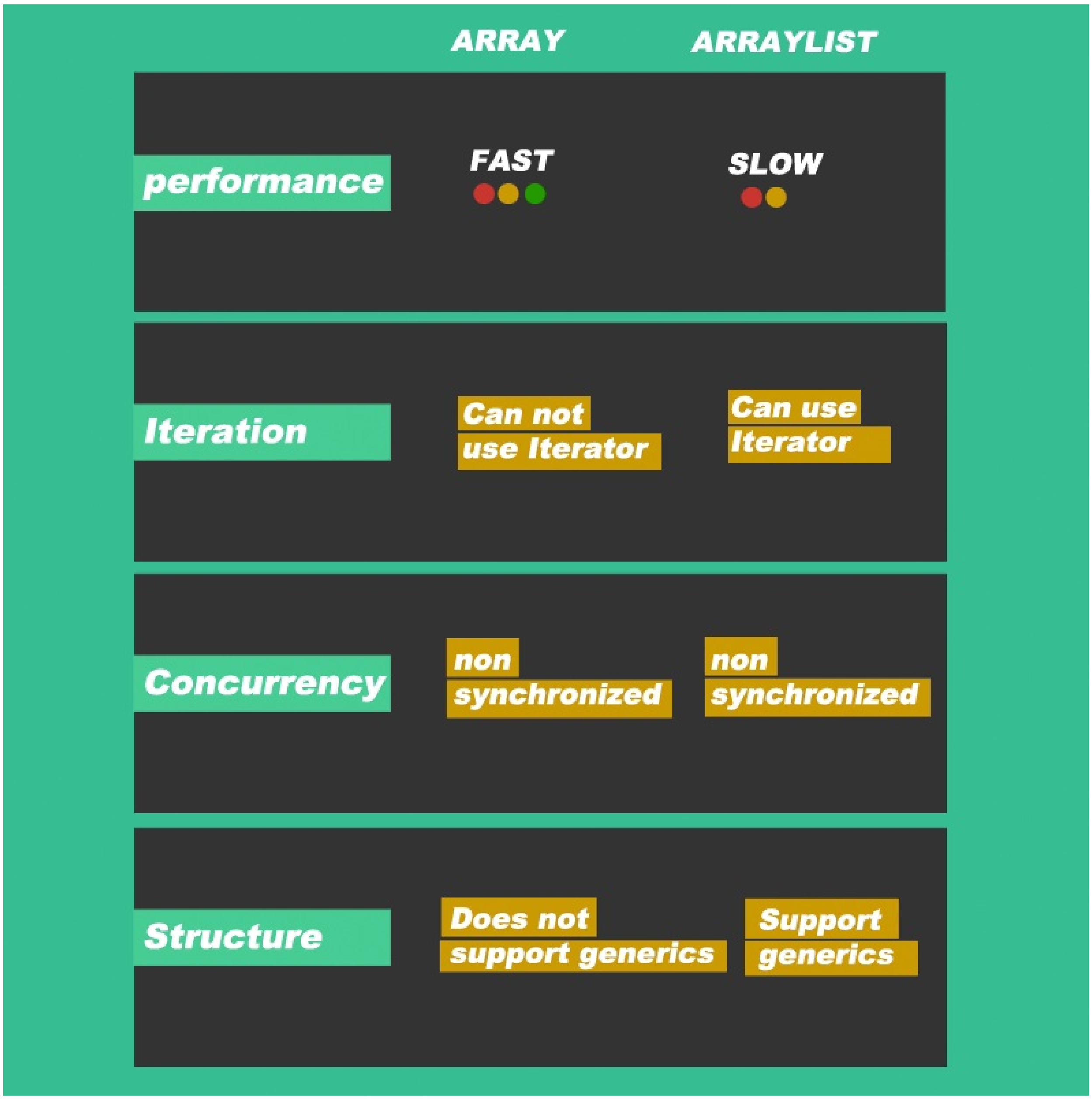
Operation	Performance	Description
Access	Θ(1)	This means very fast because we read from each index
Search	Θ(n)	'n' represents no. of elements in array. And it means search operation is slow as we need to iterate all elements in array to search for one specific element
Insertion	Θ(n)	same as above.

Deletion	Θ(n)	same as above.

How To Compare Array With Other Data Structures?

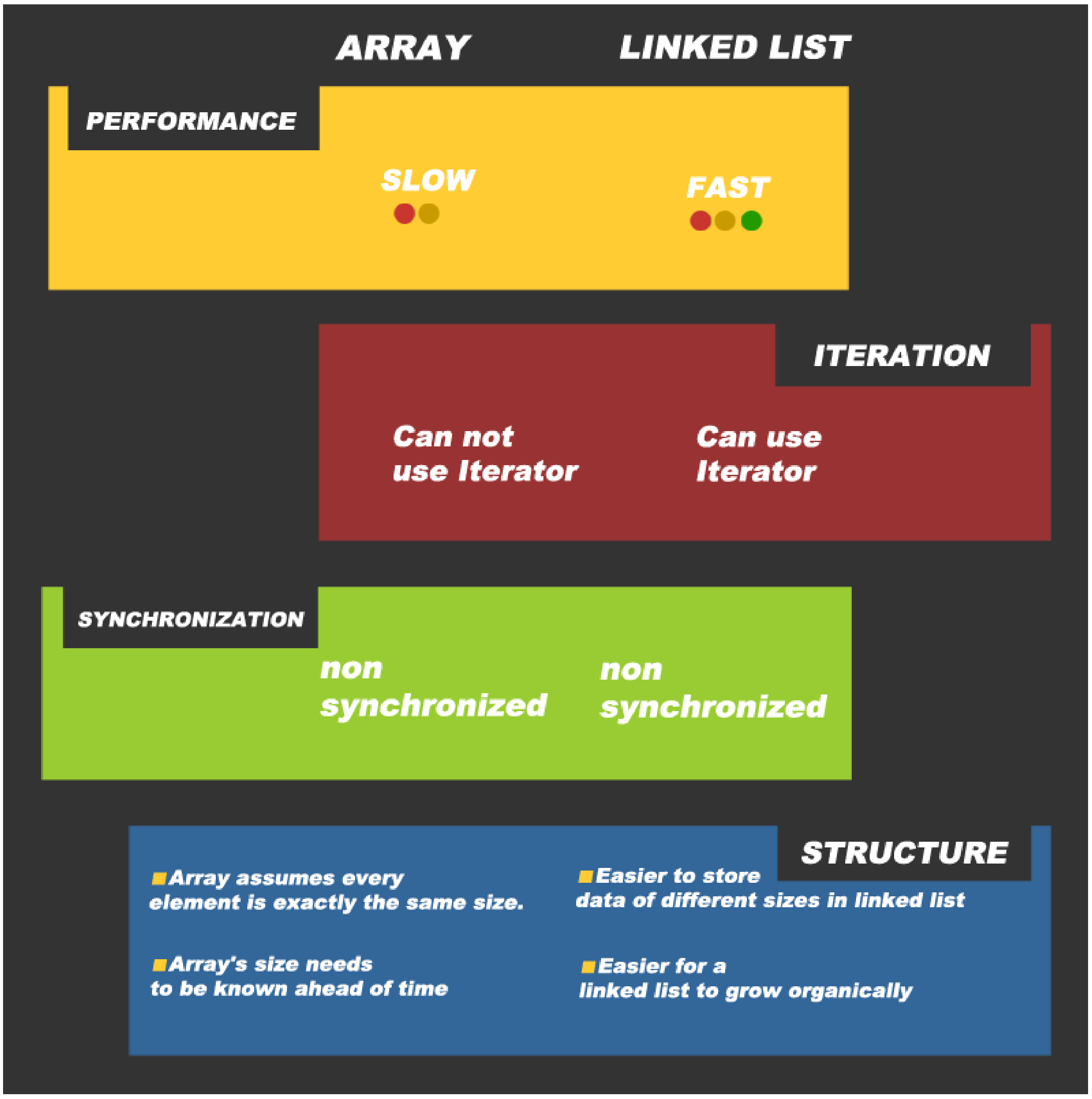


Difference Between Array and Arraylist in Java



	ARRAY	ARRAYLIST
performance	FAST ●●●	SLOW ●●
Iteration	Can not use Iterator	Can use Iterator
Concurrency	non synchronized	non synchronized
Structure	Does not support generics	Support generics

Difference Between Linked list and Array in Java



	ARRAY	LINKED LIST
PERFORMANCE	SLOW ●●	FAST ●●●
ITERATION	Can not use Iterator	Can use Iterator
SYNCHRONIZATION	non synchronized	non synchronized
STRUCTURE	■Array assumes every element is exactly the same size. ■Array's size needs to be known ahead of time	■Easier to store data of different sizes in linked list ■Easier for a linked list to grow organically

Bubble Sort Java Interview Questions?

What is Bubble Sort Algorithm?

In the bubble sort, as elements are sorted they gradually "bubble" (or rise) to their proper location in the array.

Just like bubbles rising in a glass of soda. The bubble sort repeatedly compares adjacent elements of an array.

The first and second elements are compared and swapped if out of order. Then the second and third elements are compared and swapped if out of order.

This sorting process continues until the last two elements of the array are compared and swapped if out of order

How does Bubble Sort works ?

```
for i = 1:n,
    swapped = false
    for j = n:i+1,
        if a[j] < a[j-1],
            swap a[j,j-1]
            swapped = true
    → invariant: a[1..i] in final position
    break if not swapped
end
```

The table below You can see an array of numbers before, during, and after a bubble sort for *descending* order.
A "pass" is defined as one full trip through the array comparing and if necessary, swapping, adjacentelements.
Several passes have to be made through the array before it is finally sorted.

Array at beginning:	84	69	76	86	94	91
After Pass #1:	84	76	86	94	91	69
After Pass #2:	84	86	94	91	76	69
After Pass #3:	86	94	91	84	76	69
After Pass #4:	94	91	86	84	76	69
After Pass #5 (done):	94	91	86	84	76	69

The bubble sort is an easy algorithm to program, but it is slower than many other sorts.

With a bubble sort, it is always necessary to make one final "pass" through the array.

To check to see that no swaps are made to ensure that the process is finished. In actuality, the process is finished before this last pass is made.

Bubble Sort Implementation in Java

```
package com.codespaghetti.com;

public class MyBubbleSort {

    public static void bubbleSort(int array[]) {
        int n = array.length;
        int k;
        for (int m = n; m >= 0; m--) {
            for (int i = 0; i < n - 1; i++) { k = i + 1; if (array[i] > array[k]) {
                    swapNumbers(i, k, array);
                }
            }
            printNumbers(array);
        }
    }

    private static void swapNumbers(int i, int j, int[] array) {

        int temp;
        temp = array[i];
        array[i] = array[j];
        array[j] = temp;
    }

    private static void printNumbers(int[] input) {

        for (int i = 0; i < input.length; i++) {
            System.out.print(input[i] + ", ");
        }
        System.out.println("n");
    }

    public static void main(String[] args) {
        int[] input = { 4, 2, 9, 6, 23, 12, 34, 0, 1 };
        bubble_srt(input);

    }
}
```

What are Properties of Bubble Sort?

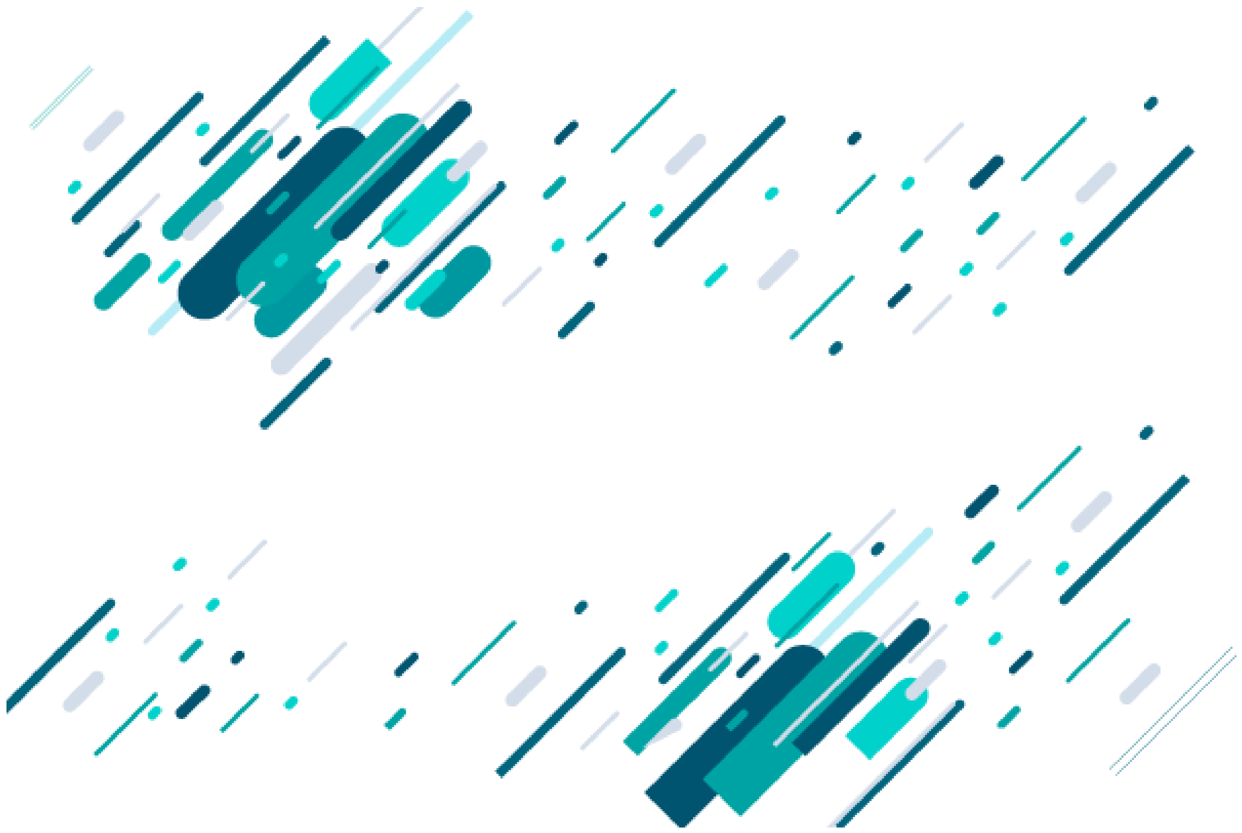
- Stable
- O(1) extra space
- O(n^2) comparisons and swaps
- Adaptive: O(n) when nearly sorted

What is Performance of Bubble Sort?

Worst-case performance	-
Best-case performance	-
Average performance	-

Worst-case space complexity	· auxiliary

Selection Sort Java Interview Questions?



What is selection sort?

The selection sort algorithm is a combination of searching and sorting. It sorts an array by repeatedly finding the minimum/maximum element from unsorted part.

and putting it at the beginning.In selection sort, the inner loop finds the next smallest (or largest) value and the outer loop places that value into its proper location.

How does it work?

```
for i = 1:n,
    k = i
    for j = i+1:n, if a[j] < a[k], k = j
    → invariant: a[k] smallest of a[i..n]
    swap a[i,k]
    → invariant: a[1..i] in final position
end
```

Let's look at following table of elements using a selection sort for descending order. Remember, a "pass" is defined as one full trip through the array comparing and if necessary, swapping elements.

Array at beginning:	84	69	76	86	94	91
After Pass #1:	84	91	76	86	94	69

After Pass #2:	84	91	94	86	76	69
After Pass #3:	86	91	94	84	76	69
After Pass #4:	94	91	86	84	76	69
After Pass #5 (done):	94	91	86	84	76	69

Full implementation of selection sort in Java

```
package codespaghetti.com;

    public class MyLinearSearch {

    public static int linerSearch(int[] arr, int key){

        int size = arr.length;
        for(int i=0;i<size;i++){
            if(arr[i] == key){
                return i;
            }
        }
        return -1;
    }

    public static void main(String a[]){

        int[] arr1= {23,45,21,55,234,1,34,90};
        int searchKey = 34;
        System.out.println("Key "+searchKey+" found at index: "+linerSearch(arr1,
searchKey));
        int[] arr2= {123,445,421,595,2134,41,304,190};
        searchKey = 421;
        System.out.println("Key "+searchKey+" found at index: "+linerSearch(arr2,
searchKey));
    }
}
```

Properties of Selection sort algorithm

- Not stable
- O(1) extra space
- $\Theta(n^2)$ comparisons
- $\Theta(n)$ swaps
- Not adaptive

When to use Selection sort?

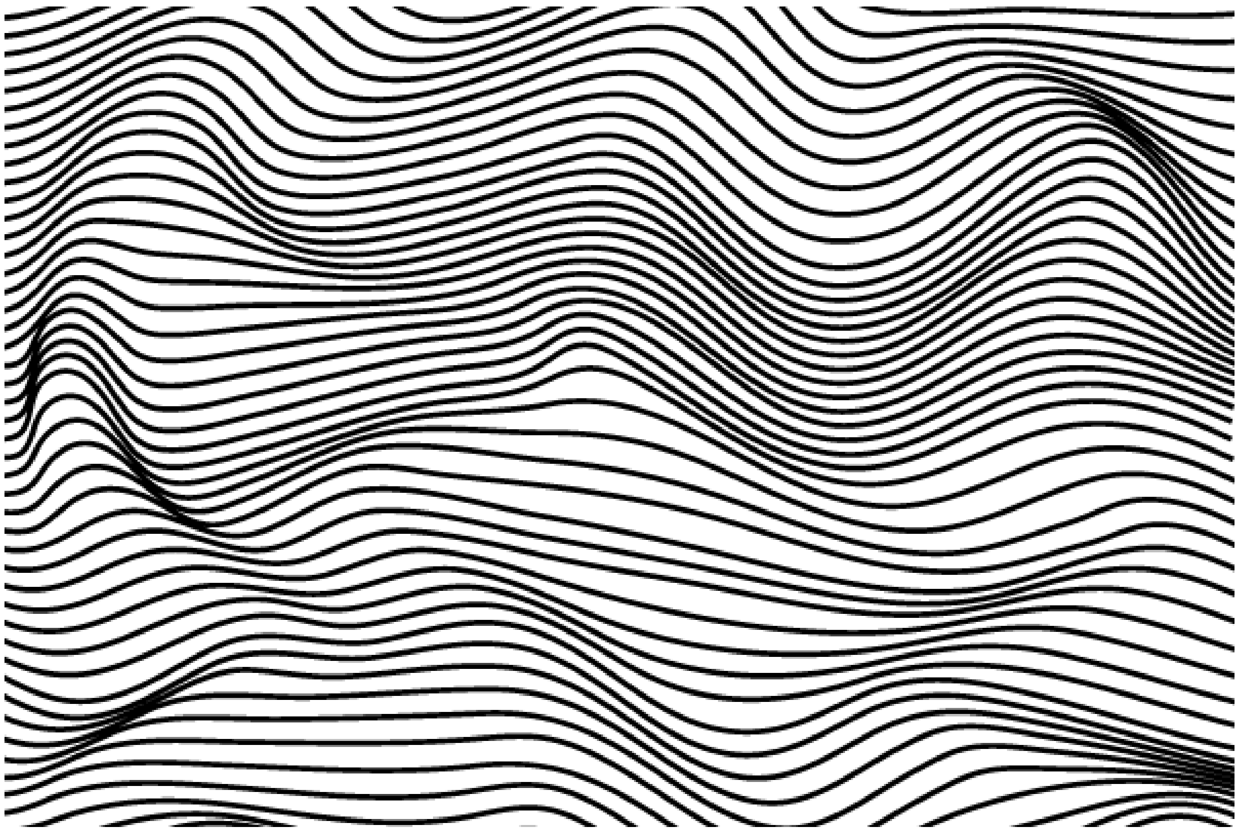
In general Selection sort should never be used. It does not adapt to the data in any way , so its runtime is always quadratic.

However, selection sort has the property of minimizing the number of swaps. In applications where the cost of swapping items is high, selection sort very well may be the algorithm of choice.

What is performance of selection sort in Big'O?

Worst-case performance	O(n^2)
Best-case performance	O(n^2)
Average performance	O(n^2)
Worst-case space Complexity	O(n) total, O(1) auxiliary

Heap Sort Java Interview Questions?



What is Heap sort algorithm?

Heap sort is a comparison-based sorting algorithm. Heap sort can be thought of as an improved selection sort.

Like that algorithm, it divides its input into a sorted and an unsorted region, and it iteratively shrinks the unsorted region by extracting the largest element and moving that to the sorted region.

The improvement consists of the use of a heap data structure rather than a linear-time search to find the maximum

How does it work ?

ALGORITHM:

```
# heapify
for i = n/2:1, sink(a,i,n)
→ invariant: a[1,n] in heap order

# sortdown
for i = 1:n,
    swap a[1,n-i+1]
    sink(a,1,n-i)
    → invariant: a[n-i+1,n] in final position
end

# sink from i in a[1..n]
function sink(a,i,n):
    # {lc,rc,mc} = {left,right,max} child index
    lc = 2*i
    if lc > n, return # no children
    rc = lc + 1
    mc = (rc > n) ? lc : (a[lc] > a[rc]) ? lc : rc
    if a[i] >= a[mc], return # heap ordered
    swap a[i,mc]
    sink(a,mc,n)
```

Heap sort implementation in Java

```java
/*
 * Java Program to Implement Heap Sort
 */

import java.util.Scanner;

/* Class HeapSort */
public class HeapSort
{
    private static int N;
    /* Sort Function */
    public static void sort(int arr[])
    {
        heapify(arr);
        for (int i = N; i > 0; i--)
        {
            swap(arr,0, i);
            N = N-1;
            maxheap(arr, 0);
        }
    }
    /* Function to build a heap */
    public static void heapify(int arr[])
    {
        N = arr.length-1;
        for (int i = N/2; i >= 0; i--)
            maxheap(arr, i);
    }
    /* Function to swap largest element in heap */
    public static void maxheap(int arr[], int i)
    {
        int left = 2*i ;
        int right = 2*i + 1;
        int max = i;
        if (left <= N && arr[left] > arr[i])
            max = left;
        if (right <= N && arr[right] > arr[max])
            max = right;

        if (max != i)
        {
            swap(arr, i, max);
            maxheap(arr, max);
        }
    }
    /* Function to swap two numbers in an array */
    public static void swap(int arr[], int i, int j)
    {
        int tmp = arr[i];
        arr[i] = arr[j];
        arr[j] = tmp;
    }
    /* Main method */
    public static void main(String[] args)
    {
        Scanner scan = new Scanner( System.in );
        System.out.println("Heap Sort Testn");
        int n, i;
        /* Accept number of elements */
        System.out.println("Enter number of integer elements");
        n = scan.nextInt();
        /* Make array of n elements */
        int arr[] = new int[ n ];
        /* Accept elements */
        System.out.println("nEnter "+ n +" integer elements");
        for (i = 0; i < n; i++)
            arr[i] = scan.nextInt();
        /* Call method sort */
        sort(arr);
        /* Print sorted Array */
        System.out.println("nElements after sorting ");
        for (i = 0; i < n; i++)
            System.out.print(arr[i]+" ");
        System.out.println();
    }
}
```

What are properties of Heap sort?

- Not stable
- O(1) extra space (see discussion)

- O(n·lg(n)) time
- Not really adaptive

When to use Heap sort?

Heap sort is simple to implement, performs an O(n·lg(n)) in-place sort, but is not stable.

The first loop, the Θ(n) “heapify” phase, puts the array into heap order. The second loop, the O(n·lg(n)) “sortdown” phase, repeatedly extracts the maximum and restores heap order.

The sink function is written recursively for clarity. Thus, as shown, the code requires Θ(lg(n)) space for the recursive call stack. However, the tail recursion in sink() is easily converted to iteration, which yields the O(1) space bound.

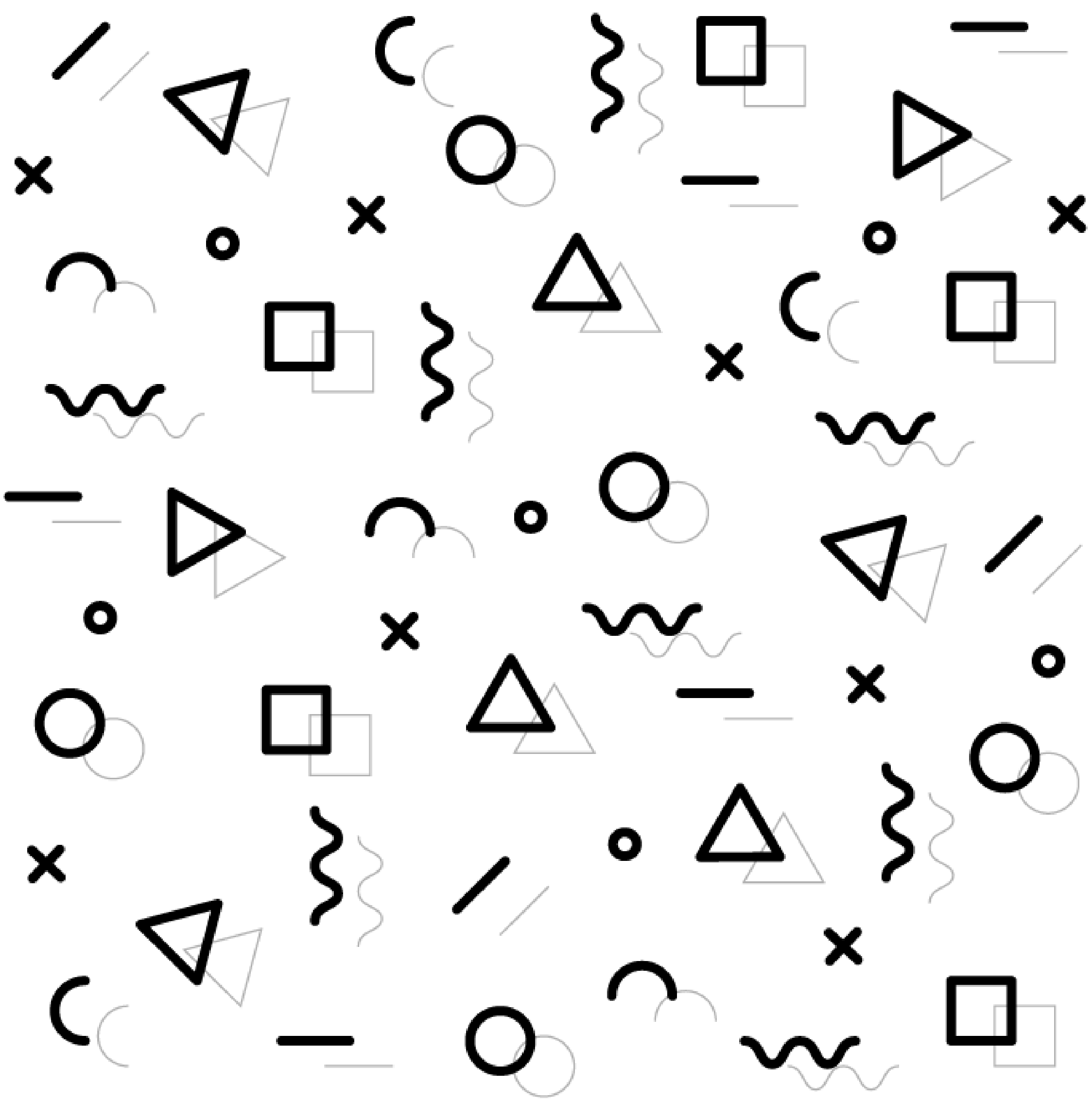
Both phases are slightly adaptive, though not in any particularly useful manner. In the nearly sorted case, the heapify phase destroys the original order.

In the reversed case, the heapify phase is as fast as possible since the array starts in heap order, but then the sortdown phase is typical. In the few unique keys case, there is some speedup but not as much as in shell sort or 3-way quicksort.

What is performance of Heap sort?

Worst-case performance	-
Best-case performance	-
Average performance	-
Worst-case space complexity	- auxiliary

Insertion Sort Java Interview Questions?



What is Insertion Sort Algorithm?

Insertion sort is a simple sorting algorithm, it builds the final sorted array one item at a time. It is much less efficient on large lists than other sort algorithms.

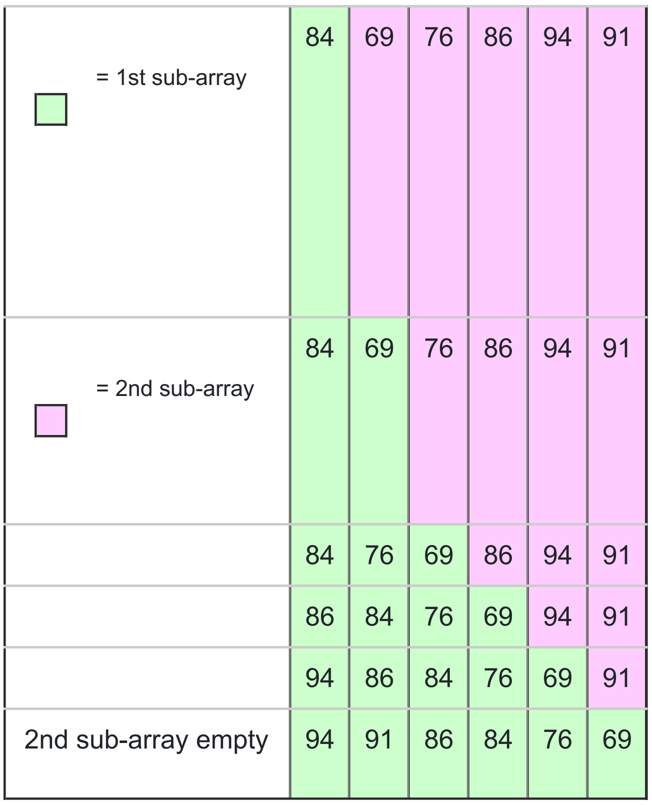
How does it work?

ALGORITHM:

```
for i = 2:n,
    for (k = i; k > 1 and a[k] < a[k-1]; k--)
        swap a[k,k-1]
    → invariant: a[1..i] is sorted
end
```

Let's look at following example using the insertion sort for descending order.

Array at beginning:	84	69	76	86	94	91



🟩 = 1st sub-array	84	69	76	86	94	91
🟪 = 2nd sub-array	84	69	76	86	94	91
	84	76	69	86	94	91
	86	84	76	69	94	91
	94	86	84	76	69	91
2nd sub-array empty	94	91	86	84	76	69

Insertion sort implementation in Java

```
package codespaghetti.com;

public class MyInsertionSort {

    public static void main(String a[]){
        int[] arr1 = {10,34,2,56,7,67,88,42};
        int[] arr2 = doInsertionSort(arr1);
        for(int i:arr2){
            System.out.print(i);
            System.out.print(", ");
        }
    }

    public static int[] doInsertionSort(int[] input){

        int temp;
        for (int i = 1; i < input.length; i++) { for(int j = i ; j > 0 ; j--){
                if(input[j] < input[j-1]){
                    temp = input[j];
                    input[j] = input[j-1];
                    input[j-1] = temp;
                }
            }
        }
        return input;
    }
}
```

Properties of Insertion sort?

- Stable
- O(1) extra space
- O(n^2) comparisons and swaps
- Adaptive: O(n) time when nearly sorted
- Very low overhead

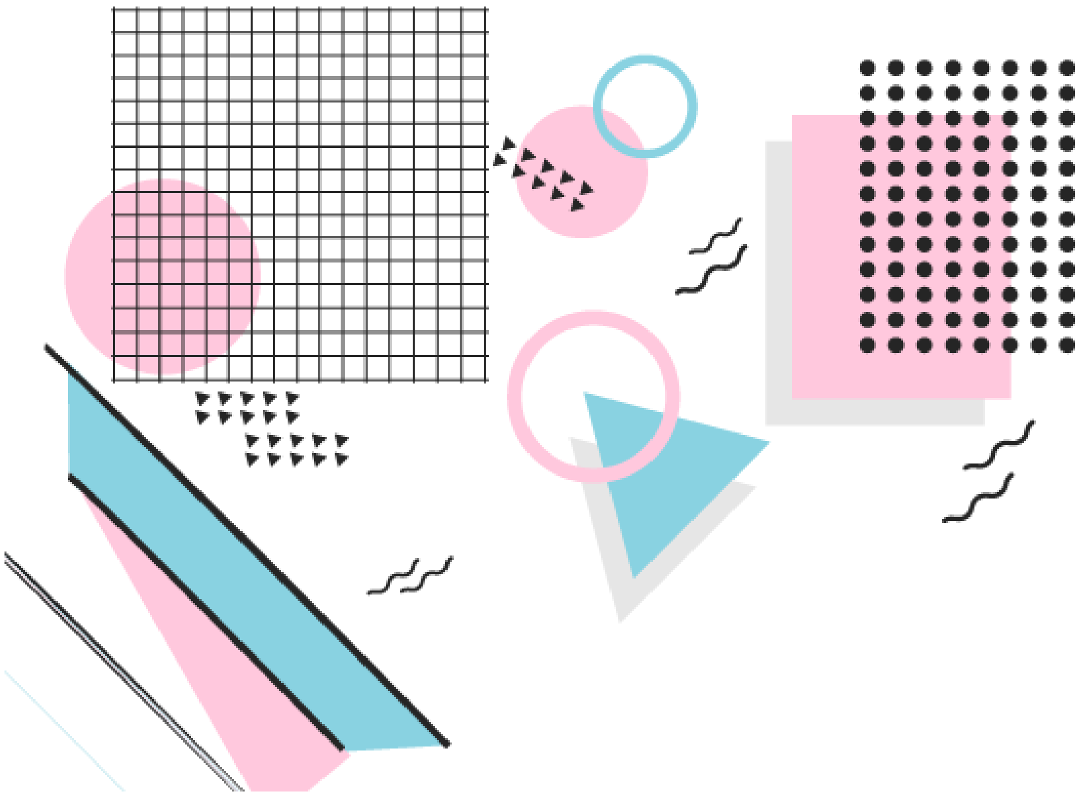
Advantages Selection sort?

- It is very simple.
- It is very efficient for small data sets.
- It is stable; i.e., it does not change the relative order of elements with equal keys.
- In-place; i.e., only requires a constant amount O(1) of additional memory space.

What is Performance of insertion sort?

Worst-case performance	O(n^2) comparisons, swaps
Best-case performance	O(n) comparisons, O(1) swaps
Average performance	O(n^2) comparisons, swaps
Worst-case space complexity	O(n) total, O(1) auxiliary

Merge Sort Java Interview Questions?



What is merge sort?

Merge sort is a sorting technique based on divide and conquer rule. With worst-case time complexity being O(n log n), it is one of the most respected algorithms.

Merge sort first divides the array into equal halves and then combines them in a sorted manner.

How does it work?

```
# split in half
m = n / 2

# recursive sorts
sort a[1..m]
sort a[m+1..n]

# merge sorted sub-arrays using temp array
b = copy of a[1..m]
i = 1, j = m+1, k = 1
while i <= m and j <= n,
    a[k++] = (a[j] < b[i]) ? a[j++] : b[i++]
    → invariant: a[1..k] in final position
while i <= m,
    a[k++] = b[i++]
    → invariant: a[1..k] in final position
```

Merge sort implementation in Java

```java
package codespaghetti.com;

public class MyMergeSort {

    private int[] array;
    private int[] tempMergArr;
    private int length;

    public static void main(String a[]){

        int[] inputArr = {45,23,11,89,77,98,4,28,65,43};
        MyMergeSort mms = new MyMergeSort();
        mms.sort(inputArr);
        for(int i:inputArr){
            System.out.print(i);
            System.out.print(" ");
        }
    }

    public void sort(int inputArr[]) {
        this.array = inputArr;
        this.length = inputArr.length;
        this.tempMergArr = new int[length];
        doMergeSort(0, length - 1);
    }
    private void doMergeSort(int lowerIndex, int higherIndex) {

        if (lowerIndex < higherIndex) {
            int middle = lowerIndex + (higherIndex - lowerIndex) / 2;
            // Below step sorts the left side of the array
            doMergeSort(lowerIndex, middle);
            // Below step sorts the right side of the array
            doMergeSort(middle + 1, higherIndex);
            // Now merge both sides
            mergeParts(lowerIndex, middle, higherIndex);
        }
    }

    private void mergeParts(int lowerIndex, int middle, int higherIndex) {

        for (int i = lowerIndex; i <= higherIndex; i++) {
            tempMergArr[i] = array[i];
        }
        int i = lowerIndex;
        int j = middle + 1;
        int k = lowerIndex;
        while (i <= middle && j <= higherIndex) {
            if (tempMergArr[i] <= tempMergArr[j]) {
                array[k] = tempMergArr[i];
                i++;
            } else {
                array[k] = tempMergArr[j];
                j++;
            }
            k++;
        }
        while (i <= middle) {
            array[k] = tempMergArr[i];
            k++;
            i++;
        }

    }
}
```

What are properties of merge sort?

- Stable
- Θ(n) extra space for arrays (as shown)
- Θ(lg(n)) extra space for linked lists
- Θ(n·lg(n)) time
- Not adaptive
- Does not require random access to data

When to use Merge sort?

Merge sort is very predictable. It makes between 0.5*lg(n) and lg(n) comparisons per element, and between lg(n) and 1.5*lg(n) swaps per element. The minima are achieved for already sorted data.

The maxima are achieved, on average, for random data. If using Θ(n) extra space is of no concern, then merge sort is an excellent choice.

It is simple to implement, and it is the only stable O(n·lg(n)) sorting algorithm. Note that when sorting linked lists, merge sort requires only Θ(lg(n)) extra space (for recursion).

Merge sort is the algorithm of choice for a variety of situations: when stability is required, when sorting linked lists, and when random access is much more expensive than sequential access (for example, external sorting on tape).

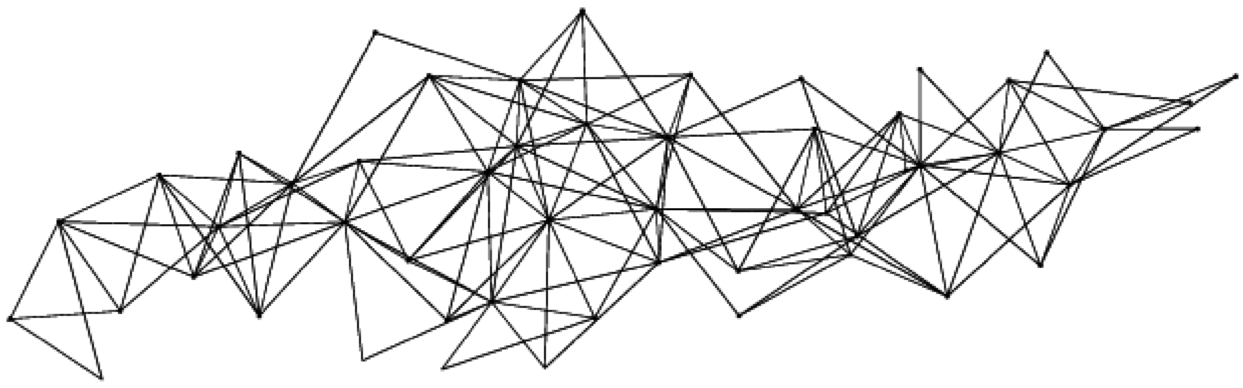
There do exist linear time *in-place* merge algorithms for the last step of the algorithm, but they are both expensive and complex.

The complexity is justified for applications such as external sorting when Θ(n) extra space is not available.

What is performance of merge sort?

Worst-case performance	O(*n* log *n*)
Best-case performance	O(*n* log *n*) typical, O(*n*) natural variant
Average performance	O(*n* log *n*)
Worst-case space complexity	O(*n*) total, O(*n*) auxiliary

Array Algorithm Java Interview Questions?



Question: Find Duplicate Numbers In Integer Array in Java [Google, phone]

Answer: There are various ways in Java to find duplicate numbers in a given array, each solution has its own positive and negative points.

But keep in mind the interviewer will be checking your ability to present a solution where you know the pros and cons of each proposed solution, like performance and other characteristics.

Following are the two most common ways

Solution 1: Loop and Compare

Loop over an array and compare each element to every other element. For doing this, we are using two loops, inner loop, and outer loop.

We are also making sure that we are ignoring comparing of elements to itself by checking for i != j before printing duplicates.

```
 for (int i = 0; i < numbers.length; i++) {
     for (int j = i + 1 ; j < numbers.length; j++) {
          if (names[i].equals(number[j])) {
           System.out.println("This is a duplicate element:"+names[i]);
         }
       }
     }
```

Performance

Since we are comparing every element to every other element, this solution has quadratic time complexity

```
O(n^2)
```

Solution 2 : Add to Set

Since Set interface doesn't allow duplicates in Java. Which means if you have added an element into Set and trying to insert duplicate element again.

It will not be allowed. In Java, you can use HashSet class to solve this problem. Just loop over array elements, insert them into HashSet using add() method and check return value.

If add() returns false it means that element is not allowed in the Set and that is your duplicate. Here is the code sample to do this :

```
for (String name : names) {
if (set.add(name) == false) {
// your duplicate element
}
}
```

Performance

Complexity of this solution is

```
O(n)
```

because you are only going through array one time, but it also has space complexity of O(n) because of HashSet data structure.

which contains your unique elements. So if an array contains 1 million elements, in worst case you would need an HashSet to store those 1 million elements.

Question: Find Intersection of Two Sorted Arrays in Java [Microsoft]

Example:

```
int[] a = { 1, 2, 3, 6, 8, 10 };
int[] b = { 4, 5, 6, 11, 15, 20 };
Output: Intersection point is : 6
```

There are two approaches to find the intersection among two arrays

Solution 1:

Use two for loops and compare each elements in both array and as soon as you find the intersection point, return it

Time complexity

```
O(n2)
```

Solution 2:

Let us consider Arrays are arrA[] and arrB[] and indexes for navigation are x and y respectively.

Since arrays are sorted, compare the first element of both the arrays.(x=0, y=0)

If both elements are same, we have our intersection point, return it.

Else if element of arrA[x] > element of arrB[y], increase the arrB[] index, y++.

Else if element of arrA[x] < element of arrB[y], increase the arrA[] index, x++.

If any of the array gets over that means you have not found the intersection point. return –1.

Time Complexity

```
 O(n)
```

Here is complete code

```
package codespaghetti.com;
public class IntersecionPoint2Arrays {
 int intersectionPoint = -1;
 int x;
 int y;

 public int intersection(int[] arrA, int[] arrB) {
  while (x < arrA.length && y < arrB.length) { if (arrA[x] > arrB[y])
    y++;
   else if (arrA[x] < arrB[y])
    x++;
   else {
    intersectionPoint = arrA[x];
    return intersectionPoint;
   }
  }
  return intersectionPoint;
 }

 public static void main(String[] args) throws java.lang.Exception {
  int[] a = { 1, 2, 3, 6, 8, 10 };
  int[] b = { 4, 5, 6, 11, 15, 20 };
  IntersecionPoint2Arrays i = new IntersecionPoint2Arrays();
  System.out.println("Intersection point is : " + i.intersection(a, b));

 }
}
```

Output

```
Intersection point is : 6
```

Alternative Questions

Based on your answer, interviewer will normally ask some other related questions to the original question.

Which can be used to further test your knowledge. Following is a list of possible questions they can ask. so make sure you know these as well

Question	Answer
What is performance	O(n) for solution 2
Find Intersection Point in Two Linked List	Check This
Find all common numbers in given three sorted arrays	Check This
What if both arrays contain over million numbers?	It will require over million iterations and will consume alot of time.

Question: Find Largest And Smallest Numbers In Unsorted Array in Java[Amazon Phone]

Answer: In order to find the largest or smallest number we will pick first element and then iterate the array and check if each number is smaller or greater then switch the numbers Here is pseudo code for the solution `for (int i : numbers) { if (i < smallest) { smallest = i; } // end finding smallest else if (i > largest) { largest = i; } // end finding largest number } // end finding largest and smallest values`
Here is complete solution

```
package codespaghetti.com;
import java.util.Arrays;

public class FindLargestAndSmallestNumbers {
 public static void main(String args[]) {
  largestAndSmallest(new int[] { -20, 34, 21, -87, 92, Integer.MAX_VALUE });
  largestAndSmallest(new int[] { 10, Integer.MIN_VALUE, -2 });
  largestAndSmallest(new int[] { Integer.MAX_VALUE, 40, Integer.MAX_VALUE });
  largestAndSmallest(new int[] { 1, -1, 0 });
 }

 public static void largestAndSmallest(int[] numbers) {
  int largest = Integer.MIN_VALUE;
  int smallest = Integer.MAX_VALUE;
  for (int number : numbers) {
   if (number > largest) {
    largest = number;
   } else if (number < smallest) {
    smallest = number;
   }
  }
  System.out.println("Given integer array : " + Arrays.toString(numbers));
  System.out.println("Largest number in array is : " + largest);
  System.out.println("Smallest number in array is : " + smallest);
 }

}
```

Output

```
Given integer array : [-20, 34, 21, -87, 92, 2147483647]
Largest number in array is : 2147483647
Smallest number in array is : -87
Given integer array : [10, -2147483648, -2]
Largest number in array is : 10
Smallest number in array is : -2147483648
Given integer array : [2147483647, 40, 2147483647]
Largest number in array is : 2147483647
Smallest number in array is : 40
Given integer array : [1, -1, 0]
Largest number in array is : 1
Smallest number in array is : -1
```

Alternative Questions

Based on your answer, interviewer will normally ask some other related questions to the original question. Which can be used to further test your knowledge. Following is a list of possible questions they can ask. so make sure you know these as well

Question	Answer
What is performance of the solution	It's O(n). In the worst case all members of the array have to be visited and compared.
Does an algorithm exist that finds the maximum of an unsorted array in O(log n) time? (This question is asked a lot)	Simple answer is No.Think about it mathematically. Unless the array is sorted, there is nothing to "cut in half" to give you the `log(n)` behavior.
How many iterations will the for loop make	Until all members of the array have been visited and compared.
How you can find the second largest or smallest number in an array	check this

Question: Find Missing Number In Array [Facebook]

Note: You can download the fully functional example at the end

Answer: An array can have one or more numbers missing and we can find them by following two approaches

1) Sum of the series: Formula: n (n+1)/2(but only work for one missing number)

2) Use BitSet, if an array has more than one missing number in array n one missing elements.

Below is the implementation of the second solution

```
package codespaghetti.com;
import java.util.Arrays;
import java.util.BitSet;
public class MissingNumber {
 public static void main(String args[]) {

  // one missing number
  printMissingNumber(new int[] { 1, 2, 3, 4, 6 }, 6);

  // two missing number
  printMissingNumber(new int[] { 1, 2, 3, 4, 6, 7, 9, 8, 10 }, 10);

  // three missing number
  printMissingNumber(new int[] { 1, 2, 3, 4, 6, 9, 8 }, 10);

  // four missing number
  printMissingNumber(new int[] { 1, 2, 3, 4, 9, 8 }, 10);

  // Only one missing number in array
  int[] iArray = new int[] { 1, 2, 3, 5 };
  int missing = getMissingNumber(iArray, 5);
  System.out.printf("Missing number in array %s is %d %n", Arrays.toString(iArray),
missing);
 }
 /**
  * A general method to find missing values from an integer array in Java.
  * This method will work even if array has more than one missing element.
  */
 private static void printMissingNumber(int[] numbers, int count) {
  int missingCount = count - numbers.length;
  BitSet bitSet = new BitSet(count);

  for (int number : numbers) {
   bitSet.set(number - 1);
  }

  System.out.printf("Missing numbers in integer array %s, with total number %d is
%n", Arrays.toString(numbers),
    count);
  int lastMissingIndex = 0;

  for (int i = 0; i < missingCount; i++) {
   lastMissingIndex = bitSet.nextClearBit(lastMissingIndex);
   System.out.println(++lastMissingIndex);
  }

 }

 private static int getMissingNumber(int[] numbers, int totalCount) {
  int expectedSum = totalCount * ((totalCount + 1) / 2);
  int actualSum = 0;
  for (int i : numbers) {
   actualSum += i;
  }
  return expectedSum - actualSum;
 }

}
```

Output `Missing numbers in integer array [1, 2, 3, 4, 6], with total number 6 is 5 Missing numbers in integer array [1, 2, 3, 4, 6, 7, 9, 8, 10], with total number 10 is 5 Missing numbers in integer array [1, 2, 3, 4, 6, 9, 8], with total number 10 is 5 7 10 Missing numbers in integer array [1, 2, 3, 4, 9, 8], with total number 10 is 5 6 7 10 Missing number in array [1, 2, 3, 5] is 4`

Question: There is an array with every element repeated twice except one. Find that element?

As an example we may consider [1, 1, 2, 3, 3, 5, 5] and the element that we look for is 2. Can we achieve it in O(log n) and in place?

Let’s consider what we know already about the problem.

All the elements appears twice, except one: it means that the size of the array is an odd number.we can see that the non repeating number is located in the odd-size part of the array.

The binary search always divides the search space into two pieces. Dividing an odd size space, gives two subspaces – one of the even size, and the second one of the odd size.

Unfortunately, dividing the array into two subarrays doesn’t give as any information which half is the odd size, and which is the even size.

But we can divide the array arbitrary, so that the first half is always even size.

Then comparing the last element L of the left subarray to the first element R of the right subarray, is sufficient to establish, in which half the extra element is located.

If L != R then we need to check the second half of the array, otherwise the left one.

On the base of the previous paragraph we can develop an algorithm described in the pseudocode below.

```
int findOneElement(int array[], int size)
{
 if(size == 1)
     return array[0];

 int medium = size/2;
 // make the size even number
 medium = medium % 2 == 0 ? medium : medium + 1;

 if(array == array)
 {
     // look in the left subarray
     return findOneElement(array, medium - 1);
 }
 else
 {
    // look in the right subarray
    return findOneElement(array + medium + 1, size - (medium + 1));
 }

}
```

The complexity is obviously

```
O(log n)
```

as we are using binary search. Moreover we don’t use any extra memory, so we get O(1). All the requirements are met. Here is full implementation

```
package array;

public class ElementThatAppearsOnceInSortedArray {

 public static void main(String[] args) {
  new ElementThatAppearsOnceInSortedArray();
 }

 public ElementThatAppearsOnceInSortedArray() {
  int arr[] = {1, 1, 2, 2, 3, 3, 4};
  int value = getElementAppearedOnce(arr, 0, arr.length-1);
  if(value==-1){
   System.out.println("There is no element that appeared once in given sorted
array.");
  }else{
   System.out.println("Element that appeared once in given sorted array is
:"+value);
  }
 }

 private int getElementAppearedOnce(int arr[], int start, int end) {

  if(start>end){
   return -1;
  }

  //This case is will appear when input is {1, 1, 2}
  if(start==end){
   return arr[start];
  }

  int mid = (start + end)/2;

  if(mid%2==0){
   //EVEN
   if(arr[mid] == arr[mid+1]){
    return getElementAppearedOnce(arr, mid+2, end);
   }else{
    return getElementAppearedOnce(arr, start, mid);
   }

  }else{
   //ODD
   if(arr[mid] == arr[mid-1]){
    return getElementAppearedOnce(arr, mid+1, end);
   }else{
    return getElementAppearedOnce(arr, start, mid);
   }
  }
 }

}
```

Question: Given a max-heap represented as an array, Return the kth largest element without modifying the heap.

Answer: Full Implementation in Java:

```
public static void findKthElementFromHeap(int[] heap, int k) {
  PriorityQueue q = new PriorityQueue(
    new HeapComparator(heap));
  int val = -1;
  q.add(0);
  int i = 0;
  while (!q.isEmpty()) {
   i++;
   int temp = q.poll();
   if (i == k) {
    val = heap[temp];
    break;
   }
   int n = (2 * temp) + 1;
   if (n < heap.length) {
    q.add(n);
   }
   n = (2 * temp) + 2;
   if (n < heap.length) {
    q.add(n);
   }
  }
  System.out.println(val);
 }
 public static void main(String[] args) {

  int[] heap = {15, 12, 10, 8, 9, 5, 6, 6, 7, 8, 5};
  findKthElementFromHeap(heap,7);


 }
 static class HeapComparator implements Comparator{

  int[] heap;

  public HeapComparator(int[] heap){
   this.heap = heap;
  }

  @Override
  public int compare(Integer o1, Integer o2) {
   return Integer.compare(heap[o2], heap[o1]);
  }

 }
```

Question: Given a sorted array, find all the numbers that occur more than n/4 times.

Full Implementation in Java:

```
public Set findNumbers(int[] arr, double k) {
        Set result = new HashSet<>();
        double size = arr.length / k;
        if (size <= 1) {
            return result;
        }
        int step = (int) size / 2;
        step = step < 1 ? 1 : step;
        for (int i = 0; i < arr.length - step; i += step) {
            if (arr[i] == arr[i + step]) {
                int start = binarySearch(i - step, i, arr);
                int end = start + (int) size;
                if (end < arr.length && arr[end] == arr[i]) {
                    result.add(arr[i]);
                }
            }
        }
        return result;
    }

    private int binarySearch(int start, int end, int[] arr) {
        if (start < 0) {
            return 0;
        }
        int target = arr[end];
        while (start < end) {
            int mid = (start + end) / 2;
            if (arr[mid] == target) {
                end = mid - 1;
            } else {
                start = mid + 1;
            }
        }
        return start;
    }
```

Question: How to randomly select a number in an array in Java?

Example:

array: [15, 2, 4, 5, 1, -2, 0]

Java implementation :

```
static public int pickRandom(int[] array, int[] freq) {
        int[] sums = new int[array.length];
        int randValue = 0;
        int sum = 0;
        int randIndex = 0;
        Random random = new Random();

        for (int i = 0; i < array.length; i++) {
            sums[i] = sum + freq[i];
            randValue += random.nextInt(freq[i] + 1);
            sum += freq[i];
            while(randIndex < (array.length - 1) && randValue >= sums[randIndex]
                     && randIndex <= i ) {
                randIndex++;
            }
        }
        return array[randIndex];
    }
```

Question: Remove duplicates from Array in Java [Google Phone]

Answer: Java program to remove duplicates from a sorted array.

```
package codespaghetti.com;

public class DuplicateElements {

public static int[] removeDuplicates(int[] input){

int j = 0;
int i = 1;
//return if the array length is less than 2
if(input.length < 2){
return input;
}
while(i < input.length){
if(input[i] == input[j]){
i++;
}else{
input[++j] = input[i++];
}
}
int[] output = new int[j+1];
for(int k=0; k<output.length; k++){
output[k] = input[k];
}

return output;
}

public static void main(String a[]){
int[] input1 = {2,3,6,6,8,9,10,10,10,12,12};
int[] output = removeDuplicates(input1);
for(int i:output){
System.out.print(i+" ");
}
}
}
```

Question: You are given array A and arrayB, write a function to shuffle arrayA and so you can get countA > countB[Google]

Analysis:

There are two integer array arrayA and arrayB in the same size and two integer countA and countB.

If arrayA[i] > arrayB[i], then we increase countA by 1.

If arrayB[i]>arrayA[i], then we increase countB by 1. We will do nothing otherwise.

Now you are given arrayA and arrayB, write a function to shuffle arrayA and so you can get countA > countB. Assume the input array are always valid, not empty and the input is guaranteed to have answer.

Example:

arrayA = [12, 24, 8, 32] arrayB = [13, 25, 32, 11]

After shuffle:

arrayA = [24, 32, 8, 12] arrayB = [13, 25, 32, 11]

Full Implementation in Java:

```
public void shuffle(int[] a, int[] b){
 int k = a.length % 2 == 0 ? a.length / 2 - 1: a.length /2;
 partitionAsc(a,k);
 int i = 0;
 int j = b.length - 1;
 while(i < j){
  if(b[i] < a[k]){
   swap(b,i,j);
   j--;
  }
  i++;
 }

}

private void partitionAsc(int[] a,int k){
 int i = 0;
 int j = a.length - 1;
 Random rnd = new Random();
 while(i < j){
  int piv = i + rnd.nextInt(j - i + 1);
  piv = partitionHelp(a,piv,i,j);
  if(piv == k){
   break;
  }
  if(piv < k){
   i = piv + 1;
  }else{
   j = piv - 1;
  }

 }
}

private int partitionHelp(int[] a, int piv, int i, int j){
 swap(a,piv,j);
 piv = j--;
 while(i <= j){ if(a[i] > a[piv]){

   swap(a,i,j);
   j--;
  }
  i++;
 }
 swap(a,piv,i);
 return i;

}
```

Question: Write 2 functions to serialize and deserialize an array of strings

Java implementation:

```
public class ArraySerializerDeserializer {

 public static String serialize(String[] a) {
  StringBuilder output = new StringBuilder();
  int maxLenght = 0;
  for (String s : a)
   if (s.length() > maxLenght)
    maxLenght = s.length();
  maxLenght++;
  output.append(maxLenght).append(":");
  String delimiter = generateRandString(maxLenght);
  for (String s : a)
   output.append(delimiter).append(s.length()).append(":").append(s);
  System.out.println(output.toString());
  return output.toString();
 }

 public static String[] deserialize(String s, int size) {
  String[] output = new String[size];
  StringBuilder sb = new StringBuilder();
  StringBuilder num = new StringBuilder();
  int i = 0;
  while (s.charAt(i) != ':') {
   num.append(s.charAt(i));
   i++;
  }
  i++;

  int maxWordSize = Integer.valueOf(num.toString());
  num = new StringBuilder();

  boolean parsingNum = false;
  boolean parsingDelimiter = true;
  int charCount = 0;
  int nextWordLenght = 0;
  int wordCount = 0;
  while (i < s.length()) {
   if (parsingDelimiter) {
    while (charCount < maxWordSize) { i++; charCount++; } parsingDelimiter = false;
parsingNum = true; charCount = 0; } else if (parsingNum) { while (s.charAt(i) !=
':') { num.append(s.charAt(i)); i++; } parsingNum = false; nextWordLenght =
Integer.valueOf(num.toString()); num = new StringBuilder(); // Emptying. i++; } else
{ while (nextWordLenght > 0) {
     sb.append(s.charAt(i));
     i++;
     nextWordLenght--;
    }
    parsingDelimiter = true;
    output[wordCount] = sb.toString();
    wordCount++;
    sb = new StringBuilder(); // Emptying.
   }
  }
  return output;
 }

 private static String generateRandString(int size) {
  StringBuilder sb = new StringBuilder();
  for (int i = 0; i < size; i++) {
   sb.append((char) (65 + (26 * Math.random())));
  }
  return sb.toString();
 }

 public static void main(String[] args) {
  String[] a = { "this", "is", "very", "nice", "I", "like" };
  String s = serialize(a);
  String[] output = deserialize(s, a.length);
  for (String out : output)
   System.out.print(out + " ");
 }

}
```

Question: Find the smallest range that includes at least one number from each of the k lists.

Example:

List 1: [4, 10, 15, 24, 26] List 2: [0, 9, 12, 20] List 3: [5, 18, 22, 30] The smallest range here would be [20, 24] as it contains 24 from list 1, 20 from list 2, and 22 from list 3.

Full Implementation in Java:

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.PriorityQueue;
import java.util.SortedSet;
import java.util.TreeSet;

public class GoogleProblem {

 public static void main(String[] args) {

  List<List> lists = new ArrayList<List>();
  List list1 = new ArrayList();
  list1.add(4);
  list1.add(10);
  list1.add(15);
  list1.add(24);
  list1.add(26);
  List list2 = new ArrayList();
  list2.add(0);
  list2.add(9);
  list2.add(12);
  list2.add(20);
  List list3 = new ArrayList();
  list3.add(5);
  list3.add(18);
  list3.add(22);
  list3.add(30);

  lists.add(list1);
  lists.add(list2);
  lists.add(list3);

  Result result = findCoveringRange(lists);
  System.out.println(result.startRange + ", " + result.endRange);
 }

 public static Result findCoveringRange(List<List> lists) {
  Result result = null;

  int start = -1, end = -1;
  int rDiff = Integer.MAX_VALUE;
  int k = lists.size();

  PriorityQueue pQueue = new PriorityQueue();
  SortedSet entries = new TreeSet();
  Map<Integer, Data> listNoAndEntry = new HashMap<Integer, Data>();

  for (int i = 0; i < k; i++) pQueue.add(new Data(lists.get(i).remove(0), i)); while
(!pQueue.isEmpty()) { Data minData = pQueue.remove(); if
(lists.get(minData.listNo).size() > 0)
    pQueue.add(new Data(lists.get(minData.listNo).remove(0),
      minData.listNo));

   if (listNoAndEntry.size() == k) {

    Data first = entries.first();
    if ((entries.last().data - first.data) + 1 < rDiff) {
     start = first.data;
     end = entries.last().data;
    }
    entries.remove(first);
    listNoAndEntry.remove(first.listNo);
   }

   if (listNoAndEntry.containsKey(minData.listNo))
    entries.remove(listNoAndEntry.remove(minData.listNo));

   listNoAndEntry.put(minData.listNo, minData);
   entries.add(minData);
  }

  if (listNoAndEntry.size() == k) {

   Data first = entries.first();
   if ((entries.last().data - first.data) + 1 < rDiff) {
    start = first.data;
    end = entries.last().data;
   }
   entries.remove(first);
   listNoAndEntry.remove(first.listNo);
  }

  result = new Result(start, end);
  return result;
 }

}

class Result {
 public final int startRange, endRange;

 public Result(int startRange, int endRange) {
  this.startRange = startRange;
  this.endRange = endRange;
 }
}

class Data implements Comparable {
 public final int data;
 public final int listNo;

 public Data(int data, int listNo) {
  this.data = data;
  this.listNo = listNo;
 }

 @Override
 public int compareTo(Data o) {
  return data - o.data;
 }
}
```

Question: Check if array can represent preorder traversal of binary search tree[Google]

Given an array of numbers, return true if given array can represent preorder traversal of a Binary Search Tree, else return false. Expected time complexity is O(n).

Examples:

```
Input:  pre[] = {2, 4, 3}
Output: true
Given array can represent preorder traversal
of below tree
    2


      4
     /
    3

Input:  pre[] = {2, 4, 1}
Output: false
Given array cannot represent preorder traversal
of a Binary Search Tree.

Input:  pre[] = {40, 30, 35, 80, 100}
Output: true
Given array can represent preorder traversal
of below tree
     40
   /
 30    80

  35     100


Input:  pre[] = {40, 30, 35, 20, 80, 100}
Output: false
Given array cannot represent preorder traversal
of a Binary Search Tree.
```

A **Simple Solution** is to do following for every node pre[i] starting from first one.

```
1) Find the first greater value on right side of current node.
   Let the index of this node be j. Return true if following
   conditions hold. Else return false
    (i)  All values after the above found greater value are
         greater than current node.
    (ii) Recursive calls for the subarrays pre[i+1..j-1] and
         pre[j+1..n-1] also return true.
```

Time Complexity of the above solution is O(n^2)

An **Efficient Solution** can solve this problem in O(n) time. The idea is to use a stack. Here we find next greater element and after finding next greater, if we find a smaller element, then return false.

```
1) Create an empty stack.
2) Initialize root as INT_MIN.
3) Do following for every element pre[i]
     a) If pre[i] is smaller than current root, return false.
     b) Keep removing elements from stack while pre[i] is greater
        then stack top. Make the last removed item as new root (to
        be compared next).
        At this point, pre[i] is greater than the removed root
        (That is why if we see a smaller element in step a), we
        return false)
     c) push pre[i] to stack (All elements in stack are in decreasing
        order)
```

Full Implementation:

```java
// Java program for an efficient solution to check if
// a given array can represent Preorder traversal of
// a Binary Search Tree
import java.util.Stack;

class BinarySearchTree {

    boolean canRepresentBST(int pre[], int n) {
        // Create an empty stack
        Stack s = new Stack();

        // Initialize current root as minimum possible
        // value
        int root = Integer.MIN_VALUE;

        // Traverse given array
        for (int i = 0; i < n; i++) {
            // If we find a node who is on right side
            // and smaller than root, return false
            if (pre[i] < root) {
                return false;
            }

            // If pre[i] is in right subtree of stack top,
            // Keep removing items smaller than pre[i]
            // and make the last removed item as new
            // root.
            while (!s.empty() && s.peek() < pre[i]) {
                root = s.peek();
                s.pop();
            }

            // At this point either stack is empty or
            // pre[i] is smaller than root, push pre[i]
            s.push(pre[i]);
        }
        return true;
    }

    public static void main(String args[]) {
        BinarySearchTree bst = new BinarySearchTree();
        int[] pre1 = new int[]{40, 30, 35, 80, 100};
        int n = pre1.length;
        if (bst.canRepresentBST(pre1, n) == true) {
            System.out.println("true");
        } else {
            System.out.println("false");
        }
        int[] pre2 = new int[]{40, 30, 35, 20, 80, 100};
        int n1 = pre2.length;
        if (bst.canRepresentBST(pre2, n) == true) {
            System.out.println("true");
        } else {
            System.out.println("false");
        }
    }
}
```

Output:

```
true
false
```

Question: How to merge two sorted Arrays in Java?

Algorithm:

If we have two arrays and both are sorted in ascending order and we want resulting array to maintain the same order. Algorithm to merge two arrays A[0..m-1] and B[0..n-1] into an array C[0..m+n-1] is as following: e.g. **arr1={4,6, 9, 20, 56} , arr2={1, 7, 25, 45, 70}result = {1,4,6,7, 9,20, 25, 45, 56, 70}**

1. Introduce read-indices **i**, **j** to traverse arrays A and B, accordingly. Introduce write-index **k** to store position of the first free cell in the resulting array. By default **i** = **j** = **k** = 0.
2. At each step: if both indices are in range (**i** < m and **j** < n), choose minimum of (A[**i**], B[**j**]) and write it to C[**k**]. Otherwise go to step 4.
3. Increase **k** and index of the array, algorithm located minimal value at, by one. Repeat step 2.
4. Copy the rest values from the array, which index is still in range, to the resulting array.

Enhancements

Algorithm could be enhanced in many ways. For instance, it is reasonable to check, if A[m - 1] < B[0] or B[n - 1] < A[0]. In any of those cases.

There is no need to do more comparisons. Algorithm could just copy source arrays in the resulting one in the right order.

More complicated enhancements may include searching for interleaving parts and run merge algorithm for them only. It could save up much time.

When sizes of merged arrays differ in scores of times.

Complexity analysis

Merge algorithm's time complexity is O(n + m). Additionally, it requires O(n + m) additional space to store resulting array.

Code snippets

Java implementation

```
public class MergeTwoSortedArrays {
 public static void main(String[] args) {
int[] ar1 = { 1, 3, 7, 11 };
        int[] ar2 = { 2, 5, 8, 22 };
        System.out.print("Display ar1 : "  );
        for (int i = 0; i < ar1.length; i++)
              System.out.print(ar1[i] +" ");
        System.out.print("nDisplay ar2 : "  );
        for (int i = 0; i < ar2.length; i++)
              System.out.print(ar2[i] +" ");
        int mergedArray[]=merging(ar1, ar2); //merging both arrays.
        System.out.print("nDisplay merged array: "  );
        for (int i = 0; i < mergedArray.length; i++)
System.out.print(mergedArray[i] +" ");
  }
  /*
  * Method merges two sorted arrays in java.
  */
  static int[] merging(int[] ar1, int ar2[]) {
        int mergedArray[]=new int[ar1.length+ar2.length];
        int ar1Index=0, ar2Index=0, mergedArrayIndex=0;
        while (ar1Index < ar1.length  && ar2Index < ar2.length)
              if (ar1[ar1Index] < ar2[ar2Index])
                   mergedArray[mergedArrayIndex++] = ar1[ar1Index++];
              else
                   mergedArray[mergedArrayIndex++] = ar2[ar2Index++];
        while (ar1Index < ar1.length )
              mergedArray[mergedArrayIndex++] = ar1[ar1Index++];
        while (ar2Index < ar2.length)
              mergedArray[mergedArrayIndex++] = ar2[ar2Index++];
        return mergedArray;
    }
}
```

Output: `Display ar1 : 1 3 7 11 Display ar2 : 2 5 8 22 Display merged array: 1 2 3 5 7 8 11 22`

Question: You are presented with following code examples and asked to review it and improve?

```
public int private printFirstThreeValues(){

public int someArray[] = new int[1000];

someArray[0]=1;
someArray[1]=2;
someArray[2]=3;
System.out.println(someArray[1]+someArray[2]+someArray[3] )
}
Answer:Remember we are focusing on the use of arrays in this program there are
following things that can be improved
```

Answer:Remember we are focusing on the use of arrays in this program there are following things that can be improved

1. **Access modifier:** Since array is in a private method no need to declare it as public
2. **name of Array:** Remember clean code principles, always use intention revealing names, name can be improved to e.g; integerArray
3. **Size of array:** We are printing first three values of arrays and only populating these three values so no need to creat an array of 1000

Download the Source Code for Java Examples

---

find largest smallest number

find missing number in array

Resources: Array Interview Questions From Around the Web

Although this guide has cover 99.99% of Array interview questions. But still if you want to consult some more excellent resources.

Then in this section I have collected high quality articles, guides and questions related to arrays.

Best Array Interview Resources:

- Interview Cake: Java Arrays
- Geeks for Geeks: Array Data Structures
- Career Cup: Array Interview Questions
- Career Guru: top 50 Array Interview Questions

C# Array Interview Questions:

- Arrays in C#
- Toptal: C# Array interview questions
- C# Array Interview Questions and Answers
- Basic C# Array Questions and Answers

Keys To Interview Success

Arrays are important from interview point of view. Because arrays are the fundamental data structure in any programming language.

In real world a lot of programming problems are solved by using arrays.

If you want to increase your chances of success in real interview be prepared there will be at least 1-3 questions from arrays and sorting algorithm is almost a must.

ARRAY

Basics

1 Arrays are objects which can store collection of same type of elements

2 An array has a certain number of elements in a fixed order

3 Accessing an invalid array index causes an exception

4 Arrays are objects,and are created on the heap, not the stack

5 Big-O Complexity of operations Access $\Theta(1)$, Search $\Theta(n)$, Insertion $\Theta(n)$, Deletion $\Theta(n)$.

6 43% Interviewers ask questions about Arrays in interviews

www.codespaghetti.com

About The Author

References

- http://introcs.cs.princeton.edu/java/14array/
- https://www.amazon.com/Cracking-Coding-Interview-Programming-Questions/dp/098478280X
- https://en.wikipedia.org/wiki/Search_algorithm
- https://www.quora.com/
- https://betterexplained.com/articles/sorting-algorithms/
- https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html