Top 100 Coding Problems from Programming Job interviews

Here is my list of 100 frequently asked coding problems from programming job interviews. In order to get most of this list, I suggest to actually solve the problem.

Do it yourself, no matter whether you stuck because that's the only way to learn. After solving a couple of problems, you will gain confidence. I also suggest you look at the solution when you stuck or after you have solved the problem, this way you learn to compare different solution and how to approach a problem from a different angle.

1. **How is a bubble sort algorithm implemented? (solution)**

Bubble Sort Algorithm in Java with Example

Bubble Sort is the first sorting algorithm I learned during my college day, and after so many years it's the one I remember by heart. It's kind of weird that one of the most popular sorting algorithm is also one of the worst performing sorting algorithm. Bubble sort's average case performance is in `O(n^2)`, which means as the size array grows, the time it take to sort that array increases quadratic. Due to this reason, bubble sort is not used in production code, instead quick sort and merge sort are preferred over it. In fact, Java's own `Arrays.sort()` method, which is the easiest way to sort an array in Java also uses two pivot quicksort to sort primitive array and stable mergesort algorithm to sort object arrays.

The reason of slow performance of this algorithm is excessive comparison and swapping, since it compare each element of array to another and swaps if it is on right side.

Due to quadratic performance, bubble sort is best suited for small, almost sorted list e.g. `{1, 2, 4, 3, 5}`, where it just need to do one swapping. Ironically, best case performance of bubble sort, which is `O(n)` beats quicksort's best case performance of `O(NlogN)`.

Someone may argue that why teaching an algorithm which that poor performance, why not teach insertion or selection sort which is as easy as bubble sort, and performs better. IMHO, easiness of algorithm depends upon programmer as much as on algorithm itself.

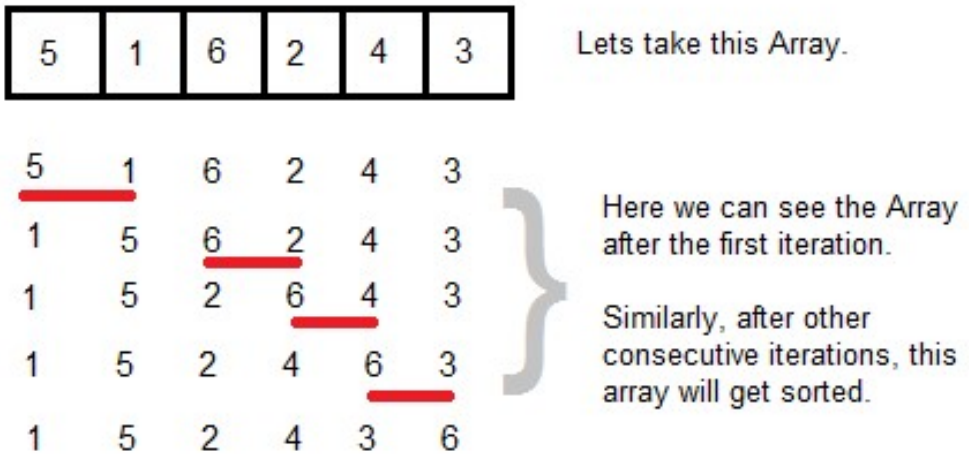
Many programmer will find *insertion sort* easier than *bubble sor*t but again there will be a lot many who will find bubble sort easier to remember, including myself. This is true, despite many of them have used insertion sort unknowingly in real life, e.g. sorting playing cards in hand.

Another reason for learning this sorting algorithm is for comparative analysis, how you improve algorithms, how you come up with different algorithms for same problems. In short, despite of all its shortcomings, bubble sort is still the most popular algorithm.

In this tutorial, we will learn *how bubble sort works*, complexity and performance of bubble sort algorithm, implementation and source code in Java and a step by step example of bubble sort.

How Bubble Sort Algorithm works

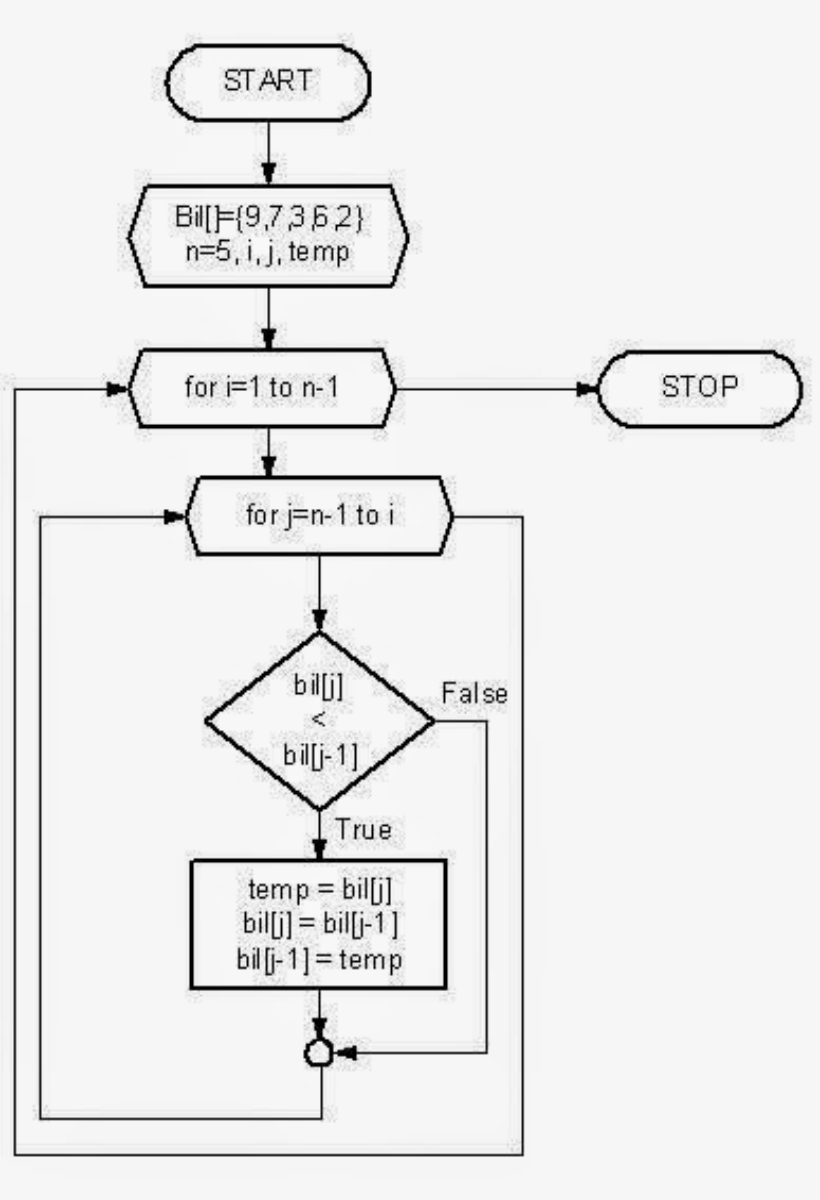
If you are the one who focus on names, then you might have got an idea how bubble sort works. Just like a bubble comes up from water, in bubble sort smallest or largest number, depending upon whether you are sorting array on ascending or descending order, bubbles up towards start or end of the array. We need at least N pass to sort the array completely and at the end of each pass one elements are sorted in its proper position. You can take first element from array, and start comparing it with other element, swapping where it's lesser than the number you are comparing. You can start this comparison from start or from end, as we have compared elements from end in our bubble sort example. It is said that a picture is worth more than a thousand word and it's particularly true in case of understanding sorting algorithm. Let' see an *step by step example to sort array using bubble sort*, as I said after each pass largest number is sorted.



In this array, we start from index 0, which is 5 and starts comparing elements from start to end. So first element we compare 5 is 1, and since 5 is greater than 1 we swap them (because ascending order sorted array will have larger number towards end). Next we compare 5 to 6, here no swapping because 6 is greater than 5 and it's on higher index than 5. Now we compare 6 to 2, again we need swapping to move 6 towards end. At the end of this pass 6 reaches (bubbles up) at the top of the array. In next iteration 5 will be sorted on its position and after n iteration all elements will be sorted. Since we compare each element with another, we need two for loops and that result in complexity of `O(n^2)`.

FlowChart of Bubble Sort Algorithm

Another cool way to understand an algorithm is to draw it's flowchart. It will walk through each iteration in loop and how decisions are made during algorithm execution. Here is flowchart of our bubble sort algorithm, which complements our implementation of this sorting algorithm.



Here we have integer array {9, 7, 3, 6, 2} and start with four variable i, j, temp and array length which is stored in variable n. We have two for loop, outer loop runs from 1 to n-1. Our inner loop runs from n-1 to i. Many programmer make mistake here, if you start outer loop with second element than make sure to use `j>=i` condition on inner loop, or if you start with first element e.g. `i=0`, make sure you use `j>i` to avoid ArrayIndexOutOfBound exception. Now we compare each element and swap them to move smaller element towards front of array. As I said depending upon your navigation direction either largest element will be sorted at highest index in first pass or smallest element will be placed in lowest index. In this case, after first pass, smallest number will be sorted. This loop runs until `j>=i` after than it finishes and i becomes `i + 1`. This whole process repeats until outer loop is finished and that time your array is sorted. In flowchart, a diamond box is used for decision making, which is equivalent of if-else statement in code. You can see here decision box is inside inner loop,

which means we do `N` comparison in each iteration, totals to `NxN` comparisons.

Complexity and Performance of Bubble Sort Algorithm

As I said before compared to other sorting algorithm like quicksort, merge sort or shell sort, bubble sort performs poorly. These algorithm has average case complexity of `O(NLogN)`, while average case complexity of bubble sort `O(n^2)`. Ironically in best case bubble sort do better than quicksort with `O(n)` performance. Bubble sort is three times slower than quicksort or mergesort even for n = 100 but it's easier to implement and remember. here is the summary of bubble sort performance and complexity :

Bubble sort Worst case performance	O(n^2)
Bubble sort Best case performance	O(n)
Bubble sort Average case performance	O(n^2)

You can further explore insertion sort and selection sort, which also does sorting in similar time complexity. By the you can not only sort the array using bubble sort but `ArrayList` or any other collection class as well. Though you should really use `Arrays.sort()` or `Collections.sort()` for those purpose.

Bubble Sort Implementation in Java

here is the Java program to implement bubble sort algorithm using Java programming language. Don't surprise with import of `java.util.Array`, we have not used it's sort method here, instead it is used to print arrays in readable format. I have created a swap function to swap numbers and improve readability of code, if you don't like you can in-line the code in the swap method inside if statement of inner loop. Though I have used main method for testing, as it demonstrate better, I would suggest you to write some unit test case for your bubble sort implementation. If you don't know how to do that, you can see this JUnit tutorial.

```
import java.util.Arrays;

/**
* Java program to implement bubble sort algorithm and sort integer array
using
* that method.
*
```

```
* @author ocj4u
*/
public class BubbleSort{

   public static void main(String args[]) {
        bubbleSort(new int[] { 20, 12, 45, 19, 91, 55 });
        bubbleSort(new int[] { -1, 0, 1 });
        bubbleSort(new int[] { -3, -9, -2, -1 });

    }

    /*
     * This method sort the integer array using bubble sort algorithm
     */
    public static void bubbleSort(int[] numbers) {
        System.out.printf("Unsorted array in Java :%s %n",
Arrays.toString(numbers));

        for (int i = 0; i < numbers.length; i++) {
            for (int j = numbers.length -1; j > i; j--) {
                if (numbers[j] < numbers[j - 1]) {
                    swap(numbers, j, j-1);
                }
            }
        }

        System.out.printf("Sorted Array using Bubble sort algorithm :%s
%n",
                Arrays.toString(numbers));
    }

    /*
     * Utility method to swap two numbers in array
     */
    public static void swap(int[] array, int from, int to){
        int temp = array[from];
        array[from] = array[to];
        array[to] = temp;
```

```
    }

}
```

```
Output
Unsorted array in Java : [20, 12, 45, 19, 91, 55]
Sorted Array using Bubble sort algorithm : [12, 19, 20, 45, 55, 91]
Unsorted array in Java : [-1, 0, 1]
Sorted Array using Bubble sort algorithm : [-1, 0, 1]
Unsorted array in Java : [-3, -9, -2, -1]
Sorted Array using Bubble sort algorithm : [-9, -3, -2, -1]
```

How to improve Bubble Sort Algorithm

In interview one of the popular follow-up question is how do you improve a particular algorithm, and Bubble Sort is no different than that. If you wrote bubble sort like the one we have shown here, interviewer will definitely going to ask about how do you improve your bubble sort method. In order to improve any algorithm, you must understand how each step of that algorithm works, then only you will be able to spot any deficiency in code. If you follow the tutorial, you will find that array is sorted by moving elements to their correct position. In worst case situation if array is reverse sorted then we need to move every element, this will require `n-1` passes, `n-1` comparison in each pass and `n-1` exchanges, but how about best case if array is already sorted, our existing bubble sort method is still going to take n-1 pass, same number of comparison but no exchange. If you observe carefully, you will find that after one pass through the array, the largest element will moved to the end of the array, but many other elements also moved toward their correct positions, as bubbles move toward the water's surface. By leveraging this property, you can deduce that during a pass, if no pair of consecutive entries is out of order, then the array is sorted. Our current algorithm is not taking advantage of this property. If we track exchanges then we can decide whether additional iteration over array is needed or not. Here is an i*mproved version of Bubble Sort algorithm*, which will only take 1 iteration and n-1 comparison in best case, when array is already sorted. This will also improve Bubble sort's average case performance, as compared to our existing method which will always take N - 1 passes.

```
    /*
     * An improved version of Bubble Sort algorithm, which will only do
     * 1 pass and n-1 comparison if array is already sorted.
     */
```

```
public static void bubbleSortImproved(int[] number) {
    boolean swapped = true;
    int last = number.length - 2;

    // only continue if swapping of number has occurred
    while (swapped) {
        swapped = false;

        for (int i = 0; i <= last; i++) {
            if (number[i] > number[i + 1]) {

                // pair is out of order, swap them
                swap(number, i, i + 1);

                swapped = true; // swapping occurred
            }
        }

        // after each pass largest element moved to end of array
        last--;
    }

}
```

Now let's test this method for two input, one in which array is sorted (best case) and other on which only one pair is out of order. If we pass int array `{10, 20, 30, 40, 50, 60}` to this method, initially will go inside while loop and make swapped=false. Then it will go inside for loop. when `i =0` it will compare number[i] to `number[i+1]` i.e. 10 to 20 and check if `10 > 20`, since it's not it will not go inside if block and no swapping will be occurred. When `i=1`, it will compare 20 > 30 again no swapping, next when i =2 30> 40 which is false so no exchange again, next `i =3` so `40> 50`, which is again false, so no swapping. Now last pair comparison `i=4`, it will compare `50 > 60` again this is false, so control will not go inside if block and no exchange will be made. Because of that swapped will remain false and control will not go inside while loop again. So you know that your array is sorted just after one pass.

Now consider another example, where just one pair is out of order, let's say String array names = `{"Ada", "C++", "Lisp", "Java", "Scala"}`, here only one pair is out of

order e.g. "Lisp" should come after "Java". Let's see how our improved bubble sort algorithm work here. In first pass, comparison will continue without exchange until we compare `"Lisp"` to `"Java"`, here `"Lisp".compareTo("Java") > 0` will become true and swapping will occur, which means Java will go to the Lisp place, and Lisp will take Java's place. this will make boolean variable `swapped=true`, Now in last comparison on this pass, we compare `"Lisp"` to `"Scala"` and again no exchange. Now we will reduce last index by 1 because Scala is sorted at last position and will not participate further. But now swapped variable is true, so control will again go inside while loop, and for loop but this time no exchanged will be made so it will not take another pass. Our array is now sorted in just two pass compared to `N-1` pass of earlier implementation. This bubble sort implementation is much better and even perform better than Selection sort algorithm in average case because, now sorting is not proportional to total number of elements but only with number of pairs which are out-of-order.

By the way to sort String array using Bubble Sort, you need to overload `BubbleSortImproved()` method to accept `String[]` and also need to use `compareTo()` method to compare two String object lexicographically. Here is Java program to sort String array using Bubble Sort :

```
import java.util.Arrays;

class BubbleSortImproved {

    public static void main(String args[]) {

        String[] test = {"Ada", "C++", "Lisp", "Java", "Scala"};
        System.out.println("Before Sorting : " + Arrays.toString(test));
        bubbleSortImproved(test);
        System.out.println("After Sorting : " + Arrays.toString(test));
    }

    /*
     * An improved implementation of Bubble Sort algorithm, which will
only do
     * 1 pass and n-1 comparison if array is already sorted.
     */
    public static void bubbleSortImproved(String[] names) {
        boolean swapped = true;
        int last = names.length - 2;
```

```
        // only continue if swapping of number has occurred
        while (swapped) {
            swapped = false;

            for (int i = 0; i <= last; i++) {
                if (names[i].compareTo(names[i + 1]) > 0) {

                    // pair is out of order, swap them
                    swap(names, i, i + 1);

                    swapped = true; // swapping occurred
                }
            }

            // after each pass largest element moved to end of array
            last--;
        }

    }

    public static void swap(String[] names, int fromIdx, int toIdx) {
        String temp = names[fromIdx]; // exchange
        names[fromIdx] = names[toIdx];
        names[toIdx] = temp;
    }

}

Output:
Before Sorting : [Ada, C++, Lisp, Java, Scala]
After Sorting : [Ada, C++, Java, Lisp, Scala]
```

Which one is better Selection Sort vs Bubble Sort?

Though both Selection Sort and Bubble sort has complexity of `O(n^2)` in worst case. On average, we expect the bubble sort to perform better than Selection sort, because bubble sort will finish sorting sooner than the selection sort due to more data movements for the same number of comparisons, because *we compare elements in pair on Bubble Sort*. If we

use our improved implementation Bubble Sort then a boolean test to not enter on while loop when array gets sorted will also help. As I said, The worst case of the bubble sort happens when the original array is in descending order, while in best case, if the original array is already sorted, the bubble sort will perform only one pass whereas the selection sort will perform N - 1 passes. Given this, I think on average Bubble sort is better than Selection sort.

That's all about **Bubble Sort in Java**. We have learned *how bubble sort algorithm works* and how do you implement it in Java. As I said, it is one of the simplest sorting algorithm and very easy to remember, but also it doesn't have any practical use apart from academics and in data structure and algorithm training classes. It's worst case performance is quadratic which means it not suitable for large array or list. If you have to use bubble sort, it's best suited for small, already sorted array in which case it has to very few swapping and it's performance is in O(n). If you love algorithms, you can see this problem of finding cycle on linked list.

2. How is a merge sort algorithm implemented? (solution)

The merge sort algorithm is a divide and conquers algorithm. In the divide and conquer paradigm, a problem is broken into smaller problems where each small problem still retains all the properties of the larger problem -- except its size. To solve the original problem, each piece is solved individually; then the pieces are merged back together. For example, imagine you have to sort an array of 200 elements using the bubble sort algorithm. Since selection sort takes O(n^2) time, it would take about 40,000-time units to sort the array. Now imagine splitting the array into ten equal pieces and sorting each piece individually still using selection sort. Now it would take 400-time units to sort each piece; for a grand total of 10*400 = 4000.

Once each piece is sorted, merging them back together would take about 200-time units; for a grand total of 200+4000 = 4,200. Clearly, 4,200 is an impressive improvement over 40,000.

Now think bigger. Imagine splitting the original array into groups of two and then sorting them. In the end, it would take about 1,000-time units to sort the array.

That's how merge sort works. It makes sorting a big array easy and hence its suitable for large integer and string arrays. Time Complexity of the mergesort algorithm is the same in the best, average and worst case and it's equal to O(n*log(n))

Sorting Array using Merge Sort Algorithm:

You have given an unordered list of integers (or any other objects e.g. String), You have to rearrange the integers or objects in their natural order.

Sample Input: {80, 50, 30, 10, 90, 60, 0, 70, 40, 20, 5}

Sample Output: {0, 10, 20, 30, 40, 50, 50, 60, 70, 80, 90}

```java
import java.util.Arrays;

/*
 * Java Program to sort an integer array using merge sort algorithm.
 */

public class Main {

  public static void main(String[] args) {

    System.out.println("mergesort");
    int[] input = { 87, 57, 370, 110, 90, 610, 02, 710, 140, 203, 150
};

    System.out.println("array before sorting");
    System.out.println(Arrays.toString(input));

    // sorting array using MergeSort algorithm
    mergesort(input);

    System.out.println("array after sorting using mergesort
algorithm");
    System.out.println(Arrays.toString(input));

  }

  /**
   * Java function to sort given array using merge sort algorithm
   *
   * @param input
   */
  public static void mergesort(int[] input) {
    mergesort(input, 0, input.length - 1);
  }
```

```java
  /**
   * A Java method to implement MergeSort algorithm using recursion
   *
   * @param input
   *          , integer array to be sorted
   * @param start
   *          index of first element in array
   * @param end
   *          index of last element in array
   */
  private static void mergesort(int[] input, int start, int end) {

    // break problem into smaller structurally identical problems
    int mid = (start + end) / 2;
    if (start < end) {
      mergesort(input, start, mid);
      mergesort(input, mid + 1, end);
    }

    // merge solved pieces to get solution to original problem
    int i = 0, first = start, last = mid + 1;
    int[] tmp = new int[end - start + 1];

    while (first <= mid && last <= end) {
      tmp[i++] = input[first] < input[last] ? input[first++] :
input[last++];
    }
    while (first <= mid) {
      tmp[i++] = input[first++];
    }
    while (last <= end) {
      tmp[i++] = input[last++];
    }
    i = 0;
    while (start <= end) {
      input[start++] = tmp[i++];
    }
```

```
  }
}

Output
mergesort
array before sorting
[87, 57, 370, 110, 90, 610, 2, 710, 140, 203, 150]
array after sorting using mergesort algorithm
[2, 57, 87, 90, 110, 140, 150, 203, 370, 610, 710]
```

You can see that the array is now sorted. The algorithm we have used is a recursive implementation of merge sort and it's also a stable sorting algorithm I mean it maintains the original order of elements in case of a tie.

That's all about how to implement the merge sort algorithm in Java. Along with Quicksort it's an important sorting algorithm and used in many real world, general purpose library. In fact, Java's Collections.sort() and Arrays.sort() method also used a variant of merge sort for sorting objects.

If you are preparing for a programming job interview then you must know how to implement it by hand and find out time and space complexity. You should also be able to explain the difference between merge sort and quicksort if asked.

The key thing to remember here is that even though both merge sort and quick sort has an average case time complexity of O(NlogN), quicksort is better than merge sort because the constant associated with quicksort is lesser than merge sort.

In reality, O(NlogN) is K*O(nLogN), Big O notation ignore that constant because it gets irrelevant when input size increases exponentially but for algorithms with same time complexity, this constant could be a differentiating factor, just like it is in the case of quicksort and mergesort.

You should also remember that it's a recursive algorithm and can be easily implemented using recursion.

3. How do you count the occurrence of a given character in a string? (solution)

Write a program to count the number of occurrences of a character in String is one of the common programming interview questions not just in Java but also in other programming languages like C or C++. As String in a very popular topic on programming interviews and there are lot of good programming exercise on String like "count number of vowels or consonants in String", "count number of characters in String" , How to reverse String in Java using recursion or without using StringBuffer etc, it becomes extremely important to have solid knowledge of String in Java or any other programming language. Though, this question is mostly used to test candidate's coding ability i.e. whether he can convert logic to code or not.

In Interview, most of the time Interviewer will ask you to write a program without using any API method, as Java is very rich and it always some kind of nice method to do the job, But it

also important to know rich Java and open source libraries for writing production quality code.

Anyway, in this question, we will see both API based and non-API based(except few) ways to count a number of occurrences of a character in String on Java.

Java program to count occurrences of a character in String

How to find number of occurence of character or substring in String in JavaIn this Java program, we will see a couple of ways to count, how many times a particular character is present in String. First, we'll see Spring framework's StringUtils class and its static method countOccurrenceOf(String, character) which takes a String and character and returns occurrence of character into that String.

After that, we will see Apache commons StringUtils class for counting occurrence of a character in String. Apache commons StringUtils provide countMatches() method which can be used to count the occurrence of one character or substring.

Finally, we will see the most simple way of counting character using standard for loop and Java 5 enhanced for loop. This solution can be extended not just to finding the occurrence of character but also finding occurrences of a substring.

Btw, if you are solving this question as part of your Java interview preparation, you can also check Cracking the Coding Interview, a collection of 189 programming questions and solutions from various programming job interviews. Your perfect companion for developing coding sense required solving these kinds of problems on interviews.

```
import org.springframework.util.StringUtils;
/**
 * Java program to count the number of occurrence of any
character on String.
 * @author ocj4u
 */
public class CountCharacters {

    public static void main(String args[]) {

        String input = "Today is Monday"; //count number of
"a" on this String.

        //Using Spring framework StringUtils class for finding
occurrence of another String
        int count =
StringUtils.countOccurrencesOf(input, "a");

        System.out.println("count of occurrence of character
'a' on String: " +
                " Today is Monday' using Spring StringUtils
" + count);


        //Using Apache commons lang StringUtils class
```

```
        int number =
org.apache.commons.lang.StringUtils.countMatches(input, "a");
        System.out.println("count of character 'a' on String:
'Today is Monday' using commons StringUtils " + number);

        //counting occurrence of character with loop
        int charCount = 0;
        for(int i =0 ; i<input.length(); i++){
            if(input.charAt(i) == 'a'){
                charCount++;
            }
        }
        System.out.println("count of character 'a' on String:
'Today is Monday' using for loop  " + charCount);

        //a more elegant way of counting occurrence of
character in String using foreach loop

        charCount = 0; //resetting character count
        for(char ch: input.toCharArray()){
            if(ch == 'a'){
                charCount++;
            }
        }
        System.out.println("count of character 'a' on String:
'Today is Monday' using for each loop  " + charCount);
    }
}
```

Output

```
count of occurrence of character 'a' on String: 'Today is
Monday' using Spring StringUtils 2
count of character 'a' on String: 'Today is Monday' using
commons StringUtils 2
count of character 'a' on String: 'Today is
Monday' using for loop  2
count of character 'a' on String: 'Today is
Monday' using for each loop  2
```

Well, the beauty of this questions is that Interviewer can twist it on many ways, they can ask you to write a recursive function to count occurrences of a particular character or they can even ask to count how many times each character has appeared.

So if a String contains multiple characters and you need to store count of each character, consider using HashMap for storing character as key and number of occurrence as value. Though there are other ways of doing it as well but I like the HashMap way of counting character for simplicity.

4. How do you print the first non-repeated character from a string? (solution)

3 ways to Find First Non Repeated Character in a String - Java Programming Problem

Write a Java program to find the first non-repeated character in a String is a common question on coding tests. Since String is a popular topic in various programming interviews, It's better to prepare well with some well-known questions like reversing String using recursion, or checking if a String is a palindrome or not. This question is also in the same league. Before jumping into solution, let's first understand this question. You need to write a function, which will accept a String and return first non-repeated character, for example in the world "hello", except 'l' all are non-repeated, but 'h' is the first non-repeated character. Similarly, in word "swiss" 'w' is the first non-repeated character. One way to solve this problem is creating a table to store count of each character, and then picking the first entry which is not repeated. The key thing to remember is order, your code must return first non-repeated letter.

By the way, In this article, we will see 3 examples to find the first non-repeated character from a String. Our first solution uses LinkedHashMap to store character count since LinkedHashMap maintains insertion order and we are inserting character in the order they appear in String, once we scanned String, we just need to iterate through LinkedHashMap and choose the entry with value 1. Yes, this solution require one LinkedHashMap and two for loops.

Our second solution is a trade-off between time and space, to find first non repeated character in one pass. This time, we have used one Set and one List to keep repeating and non-repeating character separately. Once we finish scanning through String, which is O(n), we can get the magic character by accessing List which is O(1) operator. Since List is an ordered collection get(0) returns first element.

Our third solution is also similar, but this time we have used HashMap instead of LinkedHashMap and we loop through String again to find first non-repeated character. In next section, we will the code example and unit test for this programming question. You can also see my list of String interview Questions for more of such problems and questions from Java programming language.

How to find First Non-Repeated Character from String

How to find first non-repeated character in a String in JavaHere is the full code sample of finding first duplicate character in a given String. This program has three method to find first non-repeated character. Each uses their own algorithm to do this programming task. First algorithm is implemented in getFirstNonRepeatedChar(String str) method. It first gets character array from given String and loop through it to build a hash table with character as key and their count as value. In next step, It loop through LinkedHashMap to find an entry with value 1, that's your first non-repeated character, because LinkedHashMap maintains insertion order, and we iterate through character array from beginning to end. Bad part is it requires two iteration, first one is proportional to number of character in String, and second is proportional to number of duplicate characters in String. In worst case, where String contains non-repeated character at end, it will take 2*N time to solve this problem.

Second way to find first non-repeated or unique character is coded on firstNonRepeatingChar(String word) ,this solution finds first non repeated character in a String in just one pass. It applies classical space-time trade-off technique. It uses two storage to cut down one iteration, standard space vs time trade-off. Since we store repeated and non-

repeated characters separately, at the end of iteration, first element from List is our first non repeated character from String. This one is slightly better than previous one, though it's your choice to return null or empty string if there is no non-repeated character in the String. Third way to solve this programming question is implemented in firstNonRepeatedCharacter(String word) method. It's very similar to first one except the fact that instead of LinkedHashMap, we have used HashMap. Since later doesn't guarantee any order, we have to rely on original String for finding first non repeated character. Here is the algorithm of this third solution. First step : Scan String and store count of each character in HashMap. Second Step : traverse String and get count for each character from Map. Since we are going through String from first to last character, when count for any character is 1, we break, it's the first non repeated character. Here order is achieved by going through String again.

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Set;

/**
 * Java Program to find first duplicate, non-repeated character in a
String.
 * It demonstrate three simple example to do this programming problem.
 *
 * @author ocj4u
 */
public class Programming {

    /*
     * Using LinkedHashMap to find first non repeated character of String
     * Algorithm :
     *            Step 1: get character array and loop through it to build
a
     *                    hash table with char and their count.
     *            Step 2: loop through LinkedHashMap to find an entry with
     *                    value 1, that's your first non-repeated
character,
     *                    as LinkedHashMap maintains insertion order.
     */
    public static char getFirstNonRepeatedChar(String str) {
        Map<Character,Integer> counts = new LinkedHashMap<>(str.length());

        for (char c : str.toCharArray()) {
            counts.put(c, counts.containsKey(c) ? counts.get(c) + 1 : 1);
        }

        for (Entry<Character,Integer> entry : counts.entrySet()) {
            if (entry.getValue() == 1) {
                return entry.getKey();
            }
        }
```

```
        throw new RuntimeException("didn't find any non repeated
Character");
    }


    /*
     * Finds first non repeated character in a String in just one pass.
     * It uses two storage to cut down one iteration, standard space vs
time
     * trade-off.Since we store repeated and non-repeated character
separately,
     * at the end of iteration, first element from List is our first non
     * repeated character from String.
     */
    public static char firstNonRepeatingChar(String word) {
        Set<Character> repeating = new HashSet<>();
        List<Character> nonRepeating = new ArrayList<>();
        for (int i = 0; i < word.length(); i++) {
            char letter = word.charAt(i);
            if (repeating.contains(letter)) {
                continue;
            }
            if (nonRepeating.contains(letter)) {
                nonRepeating.remove((Character) letter);
                repeating.add(letter);
            } else {
                nonRepeating.add(letter);
            }
        }
        return nonRepeating.get(0);
    }


    /*
     * Using HashMap to find first non-repeated character from String in
Java.
     * Algorithm :
     * Step 1 : Scan String and store count of each character in HashMap
     * Step 2 : traverse String and get count for each character from Map.
     *          Since we are going through String from first to last
character,
     *          when count for any character is 1, we break, it's the first
     *          non repeated character. Here order is achieved by going
     *          through String again.
     */
    public static char firstNonRepeatedCharacter(String word) {
        HashMap<Character,Integer> scoreboard = new HashMap<>();
        // build table [char -> count]
        for (int i = 0; i < word.length(); i++) {
            char c = word.charAt(i);
            if (scoreboard.containsKey(c)) {
                scoreboard.put(c, scoreboard.get(c) + 1);
            } else {
                scoreboard.put(c, 1);
            }
        }
        // since HashMap doesn't maintain order, going through string again
```

```
        for (int i = 0; i < word.length(); i++) {
            char c = word.charAt(i);
            if (scoreboard.get(c) == 1) {
                return c;
            }
        }
        throw new RuntimeException("Undefined behaviour");
    }

}
```

JUnit Test to find First Unique Character

Here are some JUnit test cases to test each of this method. We test different kind of inputs, one which contains duplicates, and other which doesn't contain duplicates. Since program has not defined what to do in case of empty String, null String and what to return if only contains duplicates, you are feel free to do in a way which make sense.

```
import static org.junit.Assert.*;
import org.junit.Test;

public class ProgrammingTest {

  @Test
  public void testFirstNonRepeatedCharacter() {
    assertEquals('b', Programming.firstNonRepeatedCharacter("abcdefghija"));
    assertEquals('h', Programming.firstNonRepeatedCharacter("hello"));
    assertEquals('J', Programming.firstNonRepeatedCharacter("Java"));
    assertEquals('i', Programming.firstNonRepeatedCharacter("simplest"));
  }

  @Test
  public void testFirstNonRepeatingChar() {
    assertEquals('b', Programming.firstNonRepeatingChar("abcdefghija"));
    assertEquals('h', Programming.firstNonRepeatingChar("hello"));
    assertEquals('J', Programming.firstNonRepeatingChar("Java"));
    assertEquals('i', Programming.firstNonRepeatingChar("simplest"));
  }

  @Test
  public void testGetFirstNonRepeatedChar() {
    assertEquals('b', Programming.getFirstNonRepeatedChar("abcdefghija"));
    assertEquals('h', Programming.getFirstNonRepeatedChar("hello"));
    assertEquals('J', Programming.getFirstNonRepeatedChar("Java"));
    assertEquals('i', Programming.getFirstNonRepeatedChar("simplest"));
  }
}
```

If you can enhance this test cases to check more scenario, just go for it. There is no better way to impress interviewer then writing detailed, creative test cases, which many programmer can't think of or just don't put effort to come up.

That's all on How to find first non-repeated character of a String in Java. We have seen three ways to solve this problem, although they use pretty much similar logic, they are different from each other. This program is also very good for beginners to master Java Collection framework. It gives you an opportunity to explore different Map implementations and understand difference between HashMap and LinkedHashMap to decide when to use them.

By the way, if you know any other way to solve this problem, feel free to share. You can also share your interview experience, If you have faced this question on Interviews.

5. How do you convert a given String into int like the atoi()? (solution)

How to Convert String to Integer to String in Java with Example

Converting String to integer and Integer to String is one of the basic tasks of Java and most people learned about it when they learn Java programming. Even though String to integer and Integer to String conversion is basic stuff but same time its most useful also because of its frequent need given that String and Integer are two most widely used type in all sort of program and you often gets data between any of these formats. One of the common tasks of programming is converting one data type another e.g. Converting Enum to String or Converting Double to String, Which is similar to converting String to Integer. Some programmers asked a question that why not Autoboxing can be used to Convert String to int primitive or Integer Object?

Remember autoboxing only converts primitive to Object it doesn't convert one data type to other. Few days back I had to convert a binary String into integer number and then I thought about this post to document all the way I know to convert Integer to String Object and String to Integer object.

Here is my way of converting String to Integer in Java with example :

String to Integer Conversion in Java

here are four different ways of converting String object into int or Integer in Java :

1) By using Intger.parseInt(String string-to-be-converted) method

This is my preferred way of converting an String to int in Java, Extremely easy and most flexible way of converting String to Integer. Let see an example of String to int conversion:

```
//using Integer.parseInt

int i = Integer.parseInt("123");

System.out.println("i: " + i);
```

Integer.parseInt() method will throw NumberFormatException if String provided is not a proper number. Same technique can be used to convert other data type like float and Double to String in Java. Java API provides static methods like Float.parseFloat() and Double.parseDouble() to perform data type conversion.

2) Integer.valueOf() method

There is another way of converting String into Integer which was hidden to me for long time mostly because I was satisfied with Integer.parseInt() method. This is an example of Factory method design pattern in Java and known as Integer.valueOf(), this is also a static method like main and can be used as utility for string to int conversion. Let's see an example of using Integer.valueOf() to convert String into int in java.

```
//How to convert numeric string = "000000081" into Integer value = 81

int i = Integer.parseInt("000000081");

System.out.println("i: " + i);
```

It will ignore the leading zeros and convert the string into int. This method also throws NumberFormatException if string provided does not represent actual number. Another interesting point about static valueOf() method is that it is used to create instance of wrapper class during Autoboxing in Java and can cause subtle issues while comparing primitive to Object e.g. int to Integer using equality operator (==), because caches Integer instance in the range -128 to 127.

How to convert Integer to String in Java

String to integer conversion, Int to string exampleIn the previous example of this String to int conversion we have seen changing String value into int primitive type and this part of Java tutorial we will see opposite i.e. conversion of Integer Object to String. In my opinion this is simpler than previous one. You can simply concatenate any number with empty String and it will create a new String. Under the carpet + operator uses either StringBuffer or StringBuilder to concatenate String in Java. anyway there are couple of more ways to convert int into String and we will see those here with examples.

Int to String in Java using "+" operator

Anything could not be more easy and simple than this. You don't have to do anything special just use "+" concatenation operator with String to convert int variable into String object as shown in the following example:

```
String price = "" + 123;
```

Simplicity aside, Using String concatenation for converting int to String is also one of the most poor way of doing it. I know it's temptation because, it's also the most easiest way to do and that's the main reason of it polluting code. When you write code like "" + 10 to convert numeric 10 to String, your code is translated into following :

```
new StringBuilder().append( "" ).append( 10 ).toString();
```

StringBuilder(String) constructor allocates a buffer containing 16 characters. So , appending upto to 16 characters to that StringBuilder will not require buffer reallocation, but appending more than 16 character will expand StringBuider buffer. Though it's not going to happen because Integer.MAX_VALUE is 2147483647, which is less than 16 character. At the end, StringBuilder.toString() will create a new String object with a copy of StringBuider buffer. This means for converting a single integer value to String you will need to allocate: one StringBuilder, one char array char[16], one String and one char[] of appropriate size to fit your input value. If you use String.vauleOf() will not only benefit from cached set of values but also you will at least avoid creating a StringBuilder. To learn more about these two, see my post StringBuffer vs StringBuilder vs String

One more example of converting int to String

There are many ways to convert an int variable into String, In case if you don't like above example of string conversion than here is one more example of doing same thing. In this example of converting Integer to String we have used String.valueOf() method which is another static utility method to convert any integer value to String. In-fact String.valueOf()

method is overloaded to accept almost all primitive type so you can use it convert char, double, float or any other data type into String. Static binding is used to call corresponding method in Java. Here is an example of int to String using String.valueOf()

String price = String.valueOf(123);

After execution of above line Integer 123 will be converted into String “123”.

Int to string using String. format()

This is a new way of converting an int primitive to String object and introduced in JDK 1.5 along-with several other important features like Enum, Generics and Variable argument methods. String.format() is even more powerful and can be used in variety of way to format String in Java. This is just another use case of String.format() method for displaying int as string. Here is an example of converting int to String using String.format method:

String price = String.format ("%d", 123);

Indeed conversion of String to Integer object or int primitive to String is pretty basic stuff but I thought let's document for quick reference of anyone who might forget it or just wanted to refresh it. By the way if you know any other way of string-int-string conversion than please let us know and I will include it here.

6. How do you implement a bucket sort algorithm? (solution)

In recent years, one of the questions I have increasingly seen in programming job interviews is about constant time sorting algorithms like do *you know any O(n) sorting algorithm?* how do they work? When I first encountered this question, I had no idea whether we can sort in constant time because even some of the fastest sorting algorithms like QuickSort or MergeSort takes **O(N log N)** time for sorting on their average case
The idea of bucket sort is quite simple, you distribute the elements of an array into a number of buckets and then sort those individual buckets by a different sorting algorithm or by recursively applying the bucket sorting algorithm.

You might have remembered, how shopkeepers or bank cashiers used to prepare/sort bundles of notes. They usually have a bucket load of cash with different denominations like 5, 10, 20, 50, or 100. They just put all 5 Rs notes into one place, all Rs 10 notes into another place and so on.

This way all notes are sorted in `O(n)` time because you don't need to sort individual buckets, they are all same notes there.

Anyway, for bucket sort to work at its super fast speed, there are multiple prerequisites. First, the hash function that is used to partition the elements must be very good and must produce ordered hash: if the `i < j` then `hash(i) < hash(j)`.

If this is not the case then you cannot concatenate individual buckets in O(n) time. Second, the elements to be sorted must be **uniformly distributed.** If you still have trouble understanding Bucket sort or want to explore other linear time sorting algorithms.

Bucket Sort vs Counting Sort

If you keep these prerequisites aside, **bucket sort** is actually very good considering that counting sort is reasonably equal to its upper bound and counting sort is also super fast.

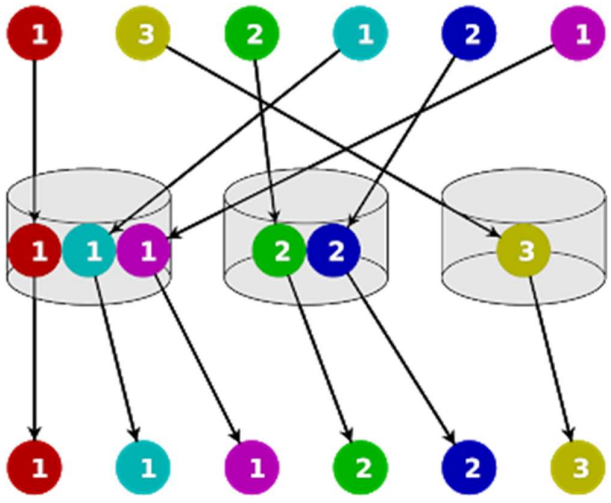
The particular distinction for bucket sort is that it uses a hash function to partition the keys of the input array, so that multiple keys may hash to the same bucket. Hence each bucket must effectively be a growable list; similar to Radix sort, another O(n) sorting algorithms.

Many programmers get confused between Counting Sort and Bucket sort as they work quite similar, but the most important difference between them is that Bucket sort uses a hash function to distribute keys while Counting sort creates a bucket for each key.

In our example, I have used JDK's `Collections.sort()` method to sort each bucket. This is to indicate that the bucket sort algorithm does not specify which sorting algorithm should be used to sort individual buckets.

A programmer may choose to recursively use bucket sort on each bucket until the input array is sorted, similar to radix sort. Anyway, whichever sorting method you use to sort individual buckets, the time complexity of bucket sort will still tend towards `O(n)`.

I think there are perhaps greater similarities between radix sort and bucket sort than there are between counting sort and bucket sort. You can also read **Introduction to Algorithm** by Thomas H. Cormen to learn more about the difference between these two constant time sorting algorithms.



Bucket Sort - FAQ

Now, let's see some of the frequently asked questions about Bucket sort in programming job interviews.

1. Is the bucket sort, a stable algorithm?

Bucket sort is not a stable sorting algorithm. If you remember, A sorting algorithm is said to be stable if two objects with equal keys appear in the same order in sorted output as they appear in the input array to be sorted. Some sorting algorithms are stable by nature like Insertion sort, Merge Sort, or Bubble Sort, etc. In the case of Bucket sort, input sorts themselves by moving into a bucket and their order are not guaranteed to be retained.

2. Is the bucket sort in-place algorithm?

Bucket sort is also not an in-place sorting algorithm because you create buckets which can be array, linked list, or hashtable and they take additional spaces. In the worst of the good cases (sequential values, but no repetition) the additional space needed is as big as the original array.

3. What is the time complexity of Bucket sort?

The time complexity of bucket sort in the `O(n)` in the best and average case and `O(n^2)` in the worst case. The creation of bucket will take O(1) time and assume moving elements into bucket will take O(1) time e.g. in the case of hashtable and linked list.

The main step is to sort elements on individual elements. This step also takes O(n) time on average if all numbers are uniformly distributed. The last step to concatenate all bucket will also take O(n) time as there will be n items in all buckets

4. What is the time complexity of Bucket sort in the worst case?

The time complexity of bucket sort is `O(n^2)` in the worst case i.e. when all elements at allocated to the same bucket. Since individual buckets are sorted using another algorithm, if only a single bucket needs to be sorted, bucket sort will take on the complexity of the inner sorting algorithm. This is why bucket sort is only useful when the input is uniformly distributed in a range. This way they will not end up in the same bucket.

5. What is the space complexity of Bucket sort?

The space complexity of the bucket sort algorithm is O(n) because even in the worst of the good cases (sequential values, but no repetition) the additional space needed is as big as the original array.

Java Program to implement Bucket Sort in Java

Problem Statement:
Given an unordered list of integers, rearrange them in the natural order.

Sample Input: {8,5,3,1,9,6,0,7,4,2,5}
Sample Output: {0,1,2,3,4,5,6,7,8,9,5}

Here is our complete Java program to implement bucket sort in Java. This program sorts an integer array using bucket sort algorithm.

Program to implement Bucket sort in Java

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;

/*
 * Java Program sort an integer array using radix sort algorithm.
 * input: [80, 50, 30, 10, 90, 60, 0, 70, 40, 20, 50]
 * output: [0, 10, 20, 30, 40, 50, 50, 60, 70, 80, 90]
 *
 * Time Complexity of Solution:
 *     Best Case O(n); Average Case O(n); Worst Case O(n^2).
 *
 */
```

```java
public class BuckeSort {

  public static void main(String[] args) {

    System.out.println("Bucket sort in Java");
    int[] input = { 80, 50, 30, 10, 90, 60, 0, 70, 40, 20, 50 };

    System.out.println("integer array before sorting");
    System.out.println(Arrays.toString(input));

    // sorting array using radix Sort Algorithm
    bucketSort(input);

    System.out.println("integer array after sorting using bucket sort
algorithm");
    System.out.println(Arrays.toString(input));

  }

  /**
   *
   * @param input
   */
  public static void bucketSort(int[] input) {
    // get hash codes
    final int[] code = hash(input);

    // create and initialize buckets to ArrayList: O(n)
    List[] buckets = new List[code[1]];
    for (int i = 0; i < code[1]; i++) {
      buckets[i] = new ArrayList();
    }

    // distribute data into buckets: O(n)
    for (int i : input) {
      buckets[hash(i, code)].add(i);
    }
```

```
    // sort each bucket O(n)
    for (List bucket : buckets) {
      Collections.sort(bucket);
    }

    int ndx = 0;
    // merge the buckets: O(n)
    for (int b = 0; b < buckets.length; b++) {
      for (int v : buckets[b]) {
        input[ndx++] = v;
      }
    }
  }

  /**
   *
   * @param input
   * @return an array containing hash of input
   */
  private static int[] hash(int[] input) {
    int m = input[0];
    for (int i = 1; i < input.length; i++) {
      if (m < input[i]) {
        m = input[i];
      }
    }
    return new int[] { m, (int) Math.sqrt(input.length) };
  }

  /**
   *
   * @param i
   * @param code
   * @return
   */
  private static int hash(int i, int[] code) {
    return (int) ((double) i / code[0] * (code[1] - 1));
```

```
  }

}
```

```
Output
Bucket sort in Java
integer array before sorting
[80, 50, 30, 10, 90, 60, 0, 70, 40, 20, 50]
integer array after sorting using bucket sort algorithm
[0, 10, 20, 30, 40, 50, 50, 60, 70, 80, 90]
```

Bucket Sort - Important points to remember

Here are some important points about bucket sort you should remember, this is useful from both interview and understanding point of view and Interviewer expects that you know about them when you say that you know bucket sort.

1) Bucket Sort is also known as bin sort because you create bins or buckets to sort inputs.

2) Bucket sort is only useful when the input is uniformly distributed over a range like coins, numbers 1 to 100 etc.

3) You can use a linked list or array as a bucket. The choice of data structure will affect the insertion time e.g. if you use linked list then adding on the head could take O(1) time. You can also use hash tables as buckets.

4) The bucket sort is one of the rare O(n) sorting algorithm i.e. time complexity of Bucket sort is the liner in best and average case and not NLogN like Quicksort or Mergesort.

5) Bucket sort is not a stable sorting algorithm because in a stable algorithm if two input is same they retain their place in sorted order and in the bucket it depends upon how you sort the individual bucket. Though, bucket sort can be made stable, known as radix sort, which we'll see in future articles.

6) You can sort the elements inside individual buckets either by recursively calling the bucket sort or using a separate sorting algorithm like insertion sort, bubble sort, or quicksort.

7) Is bucket sort an in-place sorting algorithm? No, it's not an in-place sorting algorithm. The whole idea is that input sorts themselves as they are moved to the buckets. In the worst of the good cases (sequential values, but no repetition) the additional space needed is as big as the original array.

8) The worst case complexity of bucket sort, when all elements will end up in the same bucket is O(n^2) because then it has to be sorted by a different sorting algorithm.

9) The space complexity of bucket sort is O(n) because even to sort sequential values, without repetition, you need an array as big as the original array.

That's all about bucket sort in Java. It's one of the interesting sorting algorithms which gives you O(n) performance in the best and average case. Bucket sort should only be used when the input is uniformly distributed over a range e.g. numbers up to 1 to 1000.

You should also remember that Bucket sort is not a stable algorithm hence it's not guaranteed that equal keys on input array will retain their places.

It is also not an in-place algorithm, as it will require additional space as big as the original array in the worst case. You can also refer following resources for more details on bucket sort and other O(n) sorting algorithms like Counting sort, Radix sort etc.

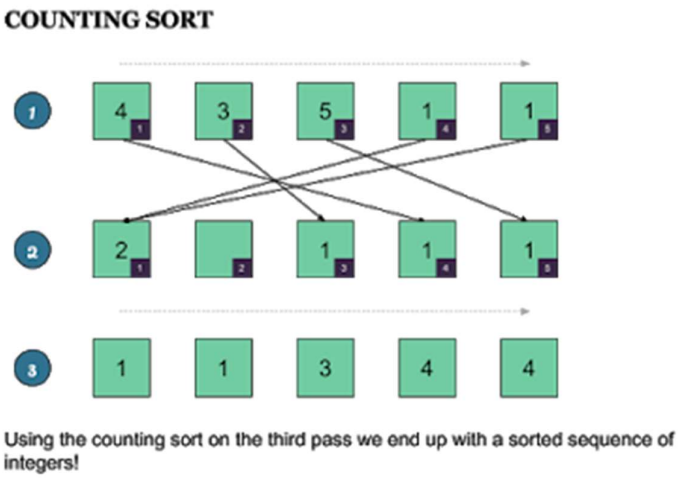
7. How do you implement a counting sort algorithm? (solution)

The Counting sort algorithm, like Radix sort and Bucket sort, is an integer based algorithm (i.e. the values of the input array are assumed to be integers), non-comparison and linear sorting algorithm. Hence counting sort is among the fastest sorting algorithms around, in theory. It is also one of the few linear sorting algorithms or O(n) sorting algorithm. It's quite common in Java interviews nowadays to ask, whether do you know any O(n) sorting algorithm or not. If you face this question in future, you can mention Radix sort, Bucket sort, or Counting sort algorithms. How does the counting sort algorithm works? Well, counting sort creates a bucket for each value and keep a counter in each bucket. Then each time a value is encountered in the input collection, the appropriate counter is incremented.

Because Counting sort algorithm creates a bucket for each value, an imposing restriction is that the maximum value in the input array is known beforehand. Once every value is inserted into the bucket, you just go through count array and print them up depending upon their frequency.

For example, if the input array contains 0 (zero) five times then at the zeroth index of count array you would have 5. Now, you can print zero 5 times before printing 1 depending upon its count. This way, you get a sorted array.

Btw, there is a large number of counting sort algorithm implementation code on the Internet, including on university websites, that erroneously claim to be bucket sort.



Using the counting sort on the third pass we end up with a sorted sequence of integers!

How to implement Counting Sorting in Java

You can follow below steps to implement counting sort algorithm in Java:

1. Since the values range from 0 to k, create k+1 buckets. For example, if your array contains 0 to 10 then create 11 buckets for storing the frequency of each number. This array is also called a frequency array or count array.

2. To fill the buckets, iterate through the input array and each time a value appears, increment the counter in its bucket.

3. Now fill the input array with the compressed data in the buckets. Each bucket's key represents a value in the array. So for each bucket, from smallest key to largest, add the index of the bucket to the input array and decrease the counter in the said bucket by one; until the counter is zero.

Java Program to sort Integer array using Counting Sort Algorithm

Problem Statement:

You have given an unordered list of repeated integers, write a Java program to arrange them in ascending order.

Sample Input: {60, 40, 30, 20, 10, 40, 30, 60, 60, 20, 40, 30, 40}

Sample Output: {10, 20, 20, 30, 30, 30, 40, 40, 40, 40, 60, 60, 60}

```
import java.util.Arrays;

/*
 * Java Program sort an integer array using counting sort algorithm.
 * input: [60, 40, 30, 20, 10, 40, 30, 60, 60, 20, 40, 30, 40]
 * output: [10, 20, 20, 30, 30, 30, 40, 40, 40, 40, 60, 60, 60]
 *
 * Time Complexity of Counting Sort Algorithm:
 *   Best Case O(n+k); Average Case O(n+k); Worst Case O(n+k),
 *   where n is the size of the input array and k means the
 *   values range from 0 to k.
 *
 */

public class CountiSorter{

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

```
    System.out.println("Counting sort in Java");
    int[] input = { 60, 40, 30, 20, 10, 40, 30, 60, 60, 20, 40, 30, 40 };
    int k = 60;

    System.out.println("integer array before sorting");
    System.out.println(Arrays.toString(input));

    // sorting array using Counting Sort Algorithm
    countingSort(input, k);

    System.out.println("integer array after sorting using counting sort
                            algorithm");
    System.out.println(Arrays.toString(input));

  }

  public static void countingSort(int[] input, int k) {
    // create buckets
    int counter[] = new int[k + 1];

    // fill buckets
    for (int i : input) {
      counter[i]++;
    }

    // sort array
    int ndx = 0;
    for (int i = 0; i < counter.length; i++) {
      while (0 < counter[i]) {
        input[ndx++] = i;
        counter[i]--;
      }
    }
  }

}
```

```
Output
Counting sort in Java
integer array before sorting
[60, 40, 30, 20, 10, 40, 30, 60, 60, 20, 40, 30, 40]
integer array after sorting using counting sort algorithm
[10, 20, 20, 30, 30, 30, 40, 40, 40, 40, 60, 60, 60]
```

You can see that our final array is now sorted in the increasing order using Counting sort algorithm. This is very useful if you have an integer array where values are not in a relatively small range.

Counting Sort FAQ

Here is some of the frequently asked question about Counting sort algorithm on interviews:

1. Is counting sort stable algorithm?

Yes, The counting sort is a stable sort like multiple keys with the same value are placed in the sorted array in the same order that they appear in the original input array. See here to learn more about the difference between stable and unstable sorting algorithms like Mergesort (a stable sorting algorithm) and Quicksort (unstable sorting algorithm).

2. When do you use counting sort algorithm?

In practice, we usually use counting sort algorithm when having k = O(n), in which case running time is O(n).

3. Is counting sort in place Algorithm?

It is possible to modify the counting sort algorithm so that it places the numbers into sorted order within the same array that was given to it as the input, using only the count array as auxiliary storage; however, the modified in-place version of counting sort is not stable. See a good algorithm course like Data Structures and Algorithms: Deep Dive Using Java on Udemy to learn about them

4. How does counting sort works?

As I said before, it first creates a count or frequency array, where each index represents the value in the input array. Hence you need a count array of k+1 to sort values in the range 0 to k, where k is the maximum value in the array. So, in order to sort an array of 1 to 100, you need an array of size 101.

After creating a count array or frequency array you just go through input array and increment counter in the respective index, which serves as a key.

For example, if 23 appears 3 times in the input array then the index 23 will contain 3. Once you create frequency array, just go through it and print the number as many times they appear in count array. You are done, the integer array is sorted now.

Here is a diagram which explains this beautifully:

Counting Sort in Java - Example

5. Is counting sort, a comparison based algorithm?

No, the counting sort is not a comparison based algorithm. It's actually a non-comparison sorting algorithm. See here to learn more about the difference between comparison and non-comparison based sorting algorithm.

6. Can you use counting sort to sort an array of String?

No, counting sort is an integer based sorting algorithm, it can only sort an integer array or number array like short, byte or char array.

That's all about Counting sort in Java. This is one of the useful O(n) sorting algorithm for sorting integer array. the linear sorting algorithm is a useful concept to remember, they are very popular nowadays on interviews. A couple of other linear sorting algorithms are Bucket sort and Radix sort. Just remember that you can only sort integer array using counting sort algorithm and you need to know the maximum value in the input array beforehand.

8. How do you remove duplicates from an array in place? (solution)

How to Remove Duplicates from Array Without Using Java Collection API

This is a coding question recently asked to one of my readers in a Java Technical interview. Question was to remove duplicates from an integer array without using any collection API classes like Set or LinkedHashSet, which can make this task trivial. In general, if you need to do this for any project work, I suggest better using Set interface, particularly LinkedHashSet, because that also keep the order on which elements are inserted into Set. Only for technical interview perspective, you need to do this using either loops or recursion, depending upon what is your strongest area. In this article, I am sharing a naive solution, which has lots of limitation to be considered as production quality code, It's not the best solution but still a solution.

The main problem, while dealing with an array is not finding duplicates, it's about removing them. Since an array is a static, fixed length data structure, you can not change its length. This means, deleting an element from an array requires creating a new array and copying content into that array.

If your input array contains lots of duplicates then this may result in lots of temporary arrays. It also increases cost of copying contents, which can be very bad. Given this restriction, you need to come out with a strategy to minimize both memory and CPU requirements.

Java Program to remove duplicates from integer array without Collection

Java Program to remove duplicates from Integer array without CollectionIn this program, we have not used any collection class to remove duplicates, earlier, I had shown you a way to remove duplicates from ArrayList, which was using LinkedHashSet. You can still use that solution if the interviewer doesn't mention without Collection specifically.

All you need to do is to convert your array into ArrayList first then subsequently create a LinkedHashSet from that ArrayList. In this example, we are removing duplicates from the array by not copying them into result array, this solution is not actually deleting duplicates instead it replacing it with default value i.e. zero.

```
import java.util.Arrays;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * Java program to remove duplicates from this array. You don't
 * need to physically delete duplicate elements, replacing with
null, or
 * empty or default value is ok.
 *
 * @author http://javarevisited.blogspot.com
 */
public class TechnicalInterviewTest {

    private static final Logger logger =
LoggerFactory.getLogger(TechnicalInterviewTest.class);

    public static void main(String args[]) {

        int[][] test = new int[][]{
            {1, 1, 2, 2, 3, 4, 5},
            {1, 1, 1, 1, 1, 1, 1},
            {1, 2, 3, 4, 5, 6, 7},
            {1, 2, 1, 1, 1, 1, 1},};

        for (int[] input : test) {
            System.out.println("Array with Duplicates       : " +
Arrays.toString(input));
            System.out.println("After removing duplicates   : " +
Arrays.toString(removeDuplicates(input)));
        }
    }

    /*
     * Method to remove duplicates from array in Java, without using
     * Collection classes e.g. Set or ArrayList. Algorithm for this
     * method is simple, it first sort the array and then compare
adjacent
     * objects, leaving out duplicates, which is already in the
result.
     */
    public static int[] removeDuplicates(int[]
numbersWithDuplicates) {

        // Sorting array to bring duplicates together
        Arrays.sort(numbersWithDuplicates);

        int[] result = new int[numbersWithDuplicates.length];
        int previous = numbersWithDuplicates[0];
```

```
        result[0] = previous;

        for (int i = 1; i < numbersWithDuplicates.length; i++) {
            int ch = numbersWithDuplicates[i];

            if (previous != ch) {
                result[i] = ch;
            }
            previous = ch;
        }
        return result;

    }
}
```

```
Output :
Array with Duplicates       : [1, 1, 2, 2, 3, 4, 5]
After removing duplicates   : [1, 0, 2, 0, 3, 4, 5]
Array with Duplicates       : [1, 1, 1, 1, 1, 1, 1]
After removing duplicates   : [1, 0, 0, 0, 0, 0, 0]
Array with Duplicates       : [1, 2, 3, 4, 5, 6, 7]
After removing duplicates   : [1, 2, 3, 4, 5, 6, 7]
Array with Duplicates       : [1, 2, 1, 1, 1, 1, 1]
After removing duplicates   : [1, 0, 0, 0, 0, 0, 2]
```

That's it about how to remove duplicates from an array in Java without using Collection class. As I said before, this solution is not perfect and has some serious limitation, which is an exercise for you to find out. One hint I can give is that array itself can contain default value as duplicates e.g. 0 for int, even if you use any Magic number e.g. Integer.MAX_VALUE, you can not be certain that they will not be part of the input.

Regarding removing duplicate permanently from result array, one approach could be to count a number of duplicates and then create an array of right size i.e. length - duplicates, and then copying content from intermediate result array to final array, leaving out elements which are marked duplicate.

9. How do you reverse an array in place in Java? (solution)

How to Reverse an Array in Java - Integer and String Array Example

This Java tips is about, how to reverse array in Java, mostly primitive types e.g. int, long, double and String arrays. Despite of Java's rich Collection API, use of array is quite common, but standard JDK doesn't has great utility classes for Java. It's difficult to convert between Java Collection e.g. List, Set to primitive arrays. Java's array utility class java.util.Arrays, though offers some of critical functionalities like comparing arrays in Java and support to print arrays, It lacks lot of common features, such as combining two arrays and reverse primitive or object array. Thankfully, Apache commons lang, an open source library from Apache software foundation offers one interesting class ArrayUtils, which can be used in conjunction with java.util.Arrays class to play with both primitive and object array in Java. This API offers convenient overloaded method to reverse different kinds of array in Java e.g. int, double, float, log or Object arrays. On a similar note, you can also write your own utility method to reverse Array in Java, and this is even a good programming question. For production usage, I personally prefer tried and tested library methods instead of reinventing

wheel. Apache commons lang fits the bill, as it offer other convenient API to complement JDK. In this Java tutorial, we will reverse int and String array in Java using ArrayUtils to show How to reverse primitive and object array in Java.

Reverse int and String array in Java - Example

Java program to reverse int and String array in Java with ExampleHere is the code example to reverse any array in Java. We have declared two arrays, iArray which is an int array and sArray which stores String objects. We have also included commons-lang-2.6.jar to use org.apache.commons.lang.ArrayUtils class to reverse Array in Java. As discussed in our last post How to print array element in Java, We are using Arrays.toString() to print content of array.

```
import java.util.Arrays;

import org.apache.commons.lang.ArrayUtils;



/**

 *

 * Java program to reverse array using Apache commons ArrayUtils class.

 * In this example we will reverse both int array and String array to

 * show how to reverse both primitive and object array in Java.

 *

 * @author

 */

public class ReverseArrayExample {



   public static void main(String args[]) {



      int[] iArray = new int[] {101,102,103,104,105};

      String[] sArray = new String[] {"one", "two", "three", "four", "five"};



      // reverse int array using Apache commons ArrayUtils.reverse() method
```

```
        System.out.println("Original int array : " + Arrays.toString(iArray));

        ArrayUtils.reverse(iArray);

        System.out.println("reversed int array : " + Arrays.toString(iArray));


        // reverse String array using ArrayUtis class

        System.out.println("Original String array : " + Arrays.toString(sArray));

        ArrayUtils.reverse(sArray);

        System.out.println("reversed String array in Java : " + Arrays.toString(sArray));

    }


}
```

Output:

```
Original int array : [101, 102, 103, 104, 105]

reversed int array : [105, 104, 103, 102, 101]

Original String array : [one, two, three, four, five]

reversed String array in Java : [five, four, three, two, one]
```

That's all on How to reverse Array in Java. In this Java program, we have seen examples to reverse String and int array using Apache commons ArrayUtils class.By the way there are couple of other ways to reverse array as well, e.g. by converting Array to List and than using Collections.reverse() method or by using brute force and reverse array using algorithm. It depends upon, which method you like. If you are doing practice and learning Java, try to do this exercise using loops.

10. How are duplicates removed from an array without using any library? (solution)

This is a coding question recently asked to one of my readers in a Java Technical interview. Question was to remove duplicates from an integer array without using any collection API classes like Set or LinkedHashSet, which can make this task trivial. In general, if you need to do this for any project work, I suggest better using Set interface, particularly LinkedHashSet, because that also keep the order on which elements are inserted into Set. Only for technical interview perspective, you need to do this using either loops or recursion, depending upon what is your strongest area. In this article, I am sharing a naive solution, which has lots of limitation to be considered as production quality code, It's not the best solution but still a solution.

The main problem, while dealing with an array is not finding duplicates, it's about removing them. Since an array is a static, fixed length data structure, you can not change its length. This means, deleting an element from an array requires creating a new array and copying content into that array.

If your input array contains lots of duplicates then this may result in lots of temporary arrays. It also increases cost of copying contents, which can be very bad. Given this restriction, you need to come out with a strategy to minimize both memory and CPU requirements.

Java Program to remove duplicates from integer array without Collection

Java Program to remove duplicates from Integer array without CollectionIn this program, we have not used any collection class to remove duplicates, earlier, I had shown you a way to remove duplicates from ArrayList, which was using LinkedHashSet. You can still use that solution if the interviewer doesn't mention without Collection specifically.

All you need to do is to convert your array into ArrayList first then subsequently create a LinkedHashSet from that ArrayList. In this example, we are removing duplicates from the array by not copying them into result array, this solution is not actually deleting duplicates instead it replacing it with default value i.e. zero.

```
import java.util.Arrays;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * Java program to remove duplicates from this array. You don't
 * need to physically delete duplicate elements, replacing with
null, or
 * empty or default value is ok.
 *
 * @author http://javarevisited.blogspot.com
 */
public class TechnicalInterviewTest {

    private static final Logger logger =
LoggerFactory.getLogger(TechnicalInterviewTest.class);

    public static void main(String args[]) {

        int[][] test = new int[][]{
            {1, 1, 2, 2, 3, 4, 5},
            {1, 1, 1, 1, 1, 1, 1},
            {1, 2, 3, 4, 5, 6, 7},
            {1, 2, 1, 1, 1, 1, 1},};

        for (int[] input : test) {
            System.out.println("Array with Duplicates       : " +
Arrays.toString(input));
            System.out.println("After removing duplicates   : " +
Arrays.toString(removeDuplicates(input)));
        }
    }
```

```
    /*
     * Method to remove duplicates from array in Java, without using
     * Collection classes e.g. Set or ArrayList. Algorithm for this
     * method is simple, it first sort the array and then compare
adjacent
     * objects, leaving out duplicates, which is already in the
result.
     */
    public static int[] removeDuplicates(int[]
numbersWithDuplicates) {

        // Sorting array to bring duplicates together
        Arrays.sort(numbersWithDuplicates);

        int[] result = new int[numbersWithDuplicates.length];
        int previous = numbersWithDuplicates[0];
        result[0] = previous;

        for (int i = 1; i < numbersWithDuplicates.length; i++) {
            int ch = numbersWithDuplicates[i];

            if (previous != ch) {
                result[i] = ch;
            }
            previous = ch;
        }
        return result;

    }
}
```

Output :

```
Array with Duplicates       : [1, 1, 2, 2, 3, 4, 5]
After removing duplicates   : [1, 0, 2, 0, 3, 4, 5]
Array with Duplicates       : [1, 1, 1, 1, 1, 1, 1]
After removing duplicates   : [1, 0, 0, 0, 0, 0, 0]
Array with Duplicates       : [1, 2, 3, 4, 5, 6, 7]
After removing duplicates   : [1, 2, 3, 4, 5, 6, 7]
Array with Duplicates       : [1, 2, 1, 1, 1, 1, 1]
After removing duplicates   : [1, 0, 0, 0, 0, 0, 2]
```

That's it about how to remove duplicates from an array in Java without using Collection class. As I said before, this solution is not perfect and has some serious limitation, which is an exercise for you to find out. One hint I can give is that array itself can contain default value as duplicates e.g. 0 for int, even if you use any Magic number e.g. Integer.MAX_VALUE, you can not be certain that they will not be part of the input.

Regarding removing duplicate permanently from result array, one approach could be to count a number of duplicates and then create an array of right size i.e. length - duplicates, and then copying content from intermediate result array to final array, leaving out elements which are marked duplicate.

11. How is a radix sort algorithm implemented? (solution)

The Radix sort, like counting sort and bucket sort, is an integer based algorithm (I mean the values of the input array are assumed to be integers). Hence radix sort is among the fastest sorting algorithms around, in theory. It is also one of the few O(n) or linear time sorting algorithm along with the Bucket and Counting sort. The particular distinction for radix sort is that it creates a bucket for each cipher (i.e. digit); as such, similar to bucket sort, each bucket in radix sort must be a growable list that may admit different keys.

For decimal values, the number of buckets is 10, as the decimal system has 10 numerals/cyphers (i.e. 0,1,2,3,4,5,6,7,8,9). Then the keys are continuously sorted by significant digits.

Time Complexity of radix sort in the best case, average case and worst case is O(k*n) where k is the length of the longest number and n is the size of the input array.

Note: if k is greater than log(n) then a n*log(n) algorithm would be a better fit. In reality, we can always change the Radix to make k less than log(n).

Java program to implement Radix sort algorithm

Before solving this problem or implementing a Radix Sort Algorithm, let's first get the problem statement right:

Problem Statement:

Given a disordered list of integers, rearrange them in the natural order.

Sample Input: {18,5,100,3,1,19,6,0,7,4,2}

Sample Output: {0,1,2,3,4,5,6,7,18,19,100}

Here is a sample program to implement the Radix sort algorithm in Java

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

/*
 * Java Program sort an integer array using radix sort algorithm.
 * input: [180, 50, 10, 30, 10, 29, 60, 0, 17, 24, 12]
 * output: [0, 10, 10, 12, 17, 24, 29, 30, 50, 60, 180]
 *
 * Time Complexity of Solution:
 *   Best Case O(k*n); Average Case O(k*n); Worst Case O(k*n),
 *   where k is the length of the longest number and n is the
```

```
 *   size of the input array.
 *
 *   Note: if k is greater than log(n) then an n*log(n) algorithm would be
a
 *         better fit. In reality we can always change the radix to make k
 *         less than log(n).
 *
 */

public class Main {

  public static void main(String[] args) {

    System.out.println("Radix sort in Java");
    int[] input = { 181, 51, 11, 33, 11, 39, 60, 2, 27, 24, 12 };

    System.out.println("An Integer array before sorting");
    System.out.println(Arrays.toString(input));

    // sorting array using radix Sort Algorithm
    radixSort(input);

    System.out.println("Sorting an int array using radix sort algorithm");
    System.out.println(Arrays.toString(input));

  }

  /**
   * Java method to sort a given array using radix sort algorithm
   *
   * @param input
   */
  public static void radixSort(int[] input) {
    final int RADIX = 10;

    // declare and initialize bucket[]
    List<Integer>[] bucket = new ArrayList[RADIX];
```

```
    for (int i = 0; i < bucket.length; i++) {
      bucket[i] = new ArrayList<Integer>();
    }

    // sort
    boolean maxLength = false;
    int tmp = -1, placement = 1;
    while (!maxLength) {
      maxLength = true;

      // split input between lists
      for (Integer i : input) {
        tmp = i / placement;
        bucket[tmp % RADIX].add(i);
        if (maxLength && tmp > 0) {
          maxLength = false;
        }
      }

      // empty lists into input array
      int a = 0;
      for (int b = 0; b < RADIX; b++) {
        for (Integer i : bucket[b]) {
          input[a++] = i;
        }
        bucket[b].clear();
      }

      // move to next digit
      placement *= RADIX;
    }
  }
}

Output
Radix sort in Java
An Integer array before sorting
[181, 51, 11, 33, 11, 39, 60, 2, 27, 24, 12]
```

```
Sorting an int array using radix sort algorithm
[2, 11, 11, 12, 24, 27, 33, 39, 51, 60, 181]
```

12. How do you swap two numbers without using the third variable? (solution)

One of the oldest trick question from programming interview is, How do you swap two integers without using temp variable? This was first asked to me on a C, C++ interview and then several times on various Java interviews. Beauty of this question lies both on trick to think about how you can swap two numbers with out third variable, but also problems associated with each approach. If a programmer can think about integer overflow and consider that in its solution then it creates a very good impression in the eye of interviewers. Ok, so let's come to the point, suppose you have tow integers i = 10 and j = 20, how will you swap them without using any variable so that j = 10 and i = 20? Though this is journal problem and solution work more or less in every other programming language, you need to do this using Java programming constructs. You can swap numbers by performing some mathematical operations e.g. addition and subtraction and multiplication and division, but they face problem of integer overflow.

Solution 1 - Using Addition and Subtraction

You can use + and - operator in Java to swap two integers as shown below :

```
a = a + b;

b = a - b;   // actually (a + b) - (b), so now b is equal to a

a = a - b;   // (a + b) -(a), now a is equal to b
```

You can see that its really nice trick and first time it took sometime to think about this approach. I was really happy to even think about this solution because its neat, but happiness was short lived because interviewer said that it will not work in all conditions. He said that integer will overflow if addition is more than maximum value of int primitive as defined by Integer.MAX_VALUE and if subtraction is less than minimum value i.e. Integer.MIN_VALUE.

It confused me and I conceded only to find that the code is absolutely fine and work perfectly in Java, because overflows are clearly defined. Ofcourse same solution will not work in C or C++ but it will work absolutely fine in Java. Don't believe me? here is the proof :

```
a = Integer.MAX_VALUE;

b = 10;

System.out.printf("Before swap 'a': %d, 'b': %d %n", a, b);
```

```
a = (a + b) - (b = a);


System.out.printf("After swapping, 'a': %d, 'b': %d %n", a, b);
```

Output :

Before swap 'a': 2147483647, 'b': 10

After swapping, 'a': 10, 'b': 2147483647

Here addition of two numbers will overflow, Integer.MAX_VALUE + 10 = -2147483639, but we are also doing subtraction, which will compensate this value, because b is now Integer.MAX_VALUE and -2147483639 - 2147483647 will again overflow to give you 10 as output.

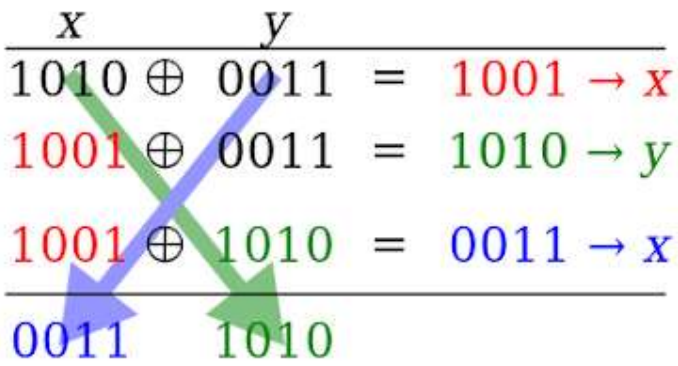
Solution 2 - Using XOR trick

Second solution to swap two integer numbers in Java without using temp variable is widely recognized as the best solution, as it will also work in language which doesn't handle integer overflow like Java e.g. C and C++. Java provides several bitwise and bitshift operator, one of them is XOR which is denoted by ^.

If you remember XOR truth table then you know that A XOR B will return 1 if A and B are different and 0 if A and B are same. This property gives birth to following code, popularly know as XOR trick:

```
x = x ^ y;

y = x ^ y;   // now y = x

x = x ^ y;  // now x = y
```

Let's see these examples in action by writing a simple Java program, as shown below.



Progarm to Swap Two Integers without tempoarry variable in Java

In this Java program, I will show you couple of ways to swap two integers without using any temporary or third variable, and what are problems comes with each approach and which one will work in all conditions. Actually the two popular solution works perfectly fine in Java but candidates, especially those coming from C and C++ background often think that first solution is broken and will fail on integer boundaries, but that's not true. Integer overflows are clearly define in Java e.g. Integer.MAX_VALUE + 1 will result in Integer.MIN_VALUE, which means if you do both addition and subtraction in with same set of numbers, your result will be fine, as seen in above example.

```
/**
* Java program to swap two integers without using temp variable
*
* @author java67
*/
public class Problem {

   public static void main(String args[]) {
      int a = 10;
      int b = 20;

      // one way using arithmetic operator e.g. + or -
      // won't work if sum overflows
      System.out.println("One way to swap two numbers without temp variable");
      System.out.printf("Before swap 'a': %d, 'b': %d %n", a, b);
      a = a + b;
      b = a - b; // actually (a + b) - (b), so now b is equal to a
      a = a - b; // (a + b) -(a), now a is equal to b

      System.out.printf("After swapping, 'a': %d, 'b': %d %n", a, b);

      // another example
      a = Integer.MIN_VALUE;
      b = Integer.MAX_VALUE;
```

```
        System.out.printf("Before swap 'a': %d, 'b': %d %n", a, b);

        a = (a + b) - (b = a);

        System.out.printf("After swapping, 'a': %d, 'b': %d %n", a, b);

        // Another way to swap integers without using temp variable is
        // by using XOR bitwise operator
        // Known as XOR trick
        System.out.println("Swap two integers without third variable
                        using XOR bitwise Operator");

        int x = 30;
        int y = 60;

        System.out.printf("Before swap 'x': %d, 'y': %d %n", x, y);
        x = x ^ y;
        y = x ^ y;
        x = x ^ y;

        System.out.printf("After swapping, 'x': %d, 'y': %d %n", x, y);
    }

}

Output
One way to swap two numbers without temp variable
Before swap 'a': 10, 'b': 20
After swapping, 'a': 20, 'b': 10
Before swap 'a': -2147483648, 'b': 2147483647
After swapping, 'a': 2147483647, 'b': -2147483648
Swap two integers without third variable using XOR bitwise Operator
Before swap 'x': 30, 'y': 60
After swapping, 'x': 60, 'y': 30
```

That's all about how to swap two integers without using temporary variable in Java. Unlike popular belief that first solution will not work on integer boundaries i.e. around maximum and minimum values, which is also true for languages like C and C++, it work perfectly fine in Java. Why? because overflow is clearly define and addition and subtraction together negate the effect of overflow. There is no doubt about second solution as it is the best solution and not subject to any sign or overflow

problem. It is also believed to be fasted way to swap two numbers without using temp variable in Java.

13. How do you check if two rectangles overlap with each other? (solution)

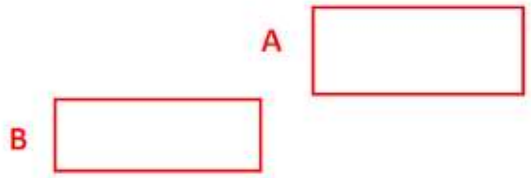
How to check if two Rectangle Overlaps

The algorithm to check if two rectangles are overlapping or not is very straight forward but before that you need to know how you can represent a rectangle in Java programs? Well, a rectangle can be represented by two coordinates, top left, and bottom right. As part of the problem you will be given four coordinates L1, R1 and L2, R2, top left and bottom right coordinate of two rectangles and you need to write a function isOverlapping() which should return true if rectangles are overlapping or false if they are not.

Algorithm to check if rectangles are overlapping

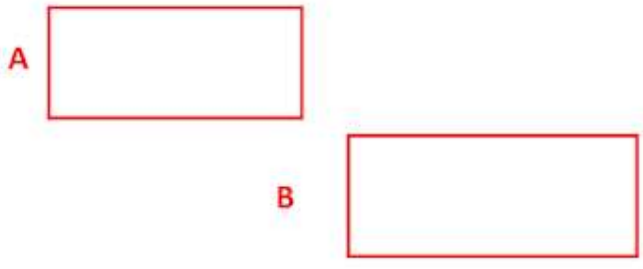
Two rectangles A and B will not overlap or intersect with each other if one of the following four conditions is true.

1. Left edge of A is to the right of right edge of B. In this case first rectangle A is completely on right side of second rectangle B as shown in the following diagram



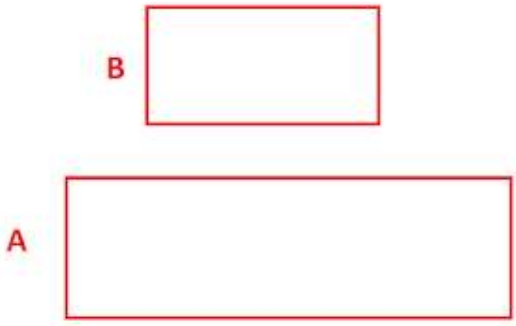
i) Rectangle A is to the right of B

2. right edge of A is to the left of left edge of B. In this case first rectangle A is completely on the left side of second rectangle B, as shown below



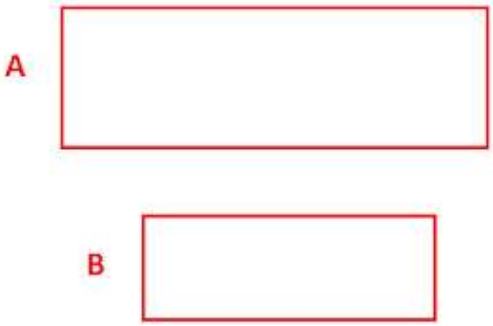
ii) Rectnagle A is left of Rectangle B

3. Top edge of A is below bottom edge of B. In this case, first rectangle A is completely under second rectangle B as shown in the following diagram



iii) Rectangle A is below Rectangle B

4. Bottom edge of A is above top edge of B. In this case, first rectangle A is completely above second rectangle B as shown in the following diagram



iv) Rectangle A is over Rectangle B

If any of above four conditions is not true then two rectangles are overlapping with each other, as in following diagram first condition is violated, hence rectangle A intersects rectangle B.

Java Program to check if two rectangles are intersecting

In this program, I have followed standard Java coding practices to solve this problem. I have encapsulated two co-ordinates in a Point class and have Rectangle has two Point instance variable and an instance method like equals() to check if another rectangle is overlapping with it or not. The logic to check if two rectangles are overlapping or not is coded in the isOverlapping() method which is as per the approach discuss in the solution section.

```
/*
 * Java Program to check if two rectangles is intersecting with each
 * other. This algorithm is also used as collision detection
 * algorithm in sprite-based arcade games e.g. Supre Mario Bros
 */

public class Main {

  public static void main(String[] args) {
  Point l1 = new Point(0, 10);
  Point r1 = new Point(10, 0);
  Point l2 = new Point(5, 5);
  Point r2 = new Point(15, 0);

  Rectangle first = new Rectangle(l1, r1);
  Rectangle second = new Rectangle(l2, r2);

  if (first.isOverLapping(second)) {
  System.out
  .println("Yes, two rectangles are intersecting with each other");
  } else {
  System.out
  .println("No, two rectangles are not overlapping with each other");
  }
  }

}

class Point {
  int x;
  int y;
```

```
  public Point(int x, int y) {
  super();
  this.x = x;
  this.y = y;
  }
}

class Rectangle {

  private final Point topLeft;
  private final Point bottomRight;

  public Rectangle(Point topLeft, Point bottomRight) {
  this.topLeft = topLeft;
  this.bottomRight = bottomRight;
  }

  /**
  * Java method to check if two rectangle are intersecting to each other.
  *
  * @param other
  * @return true if two rectangle overlap with each other
  */
  public boolean isOverLapping(Rectangle other) {
  if (this.topLeft.x > other.bottomRight.x // R1 is right to R2
  || this.bottomRight.x < other.topLeft.x // R1 is left to R2
  || this.topLeft.y < other.bottomRight.y // R1 is above R2
  || this.bottomRight.y > other.topLeft.y) { // R1 is below R1
  return false;
  }
  return true;
  }

}

Output
Yes, two rectangles are intersecting with each other
```

That's all about **how to check whether two rectangles are overlapping with each other or not**. It's one of the interesting coding questions and trying to develop the algorithm to solve the problem is also a good choice. Try testing this algorithm with a different kind of rectangle to see if it works in all scenario or not

14. How do you design a vending machine? (solution)

Problem Statement

You need to design a Vending Machine which

1. Accepts coins of 1,5,10,25 Cents i.e. penny, nickel, dime, and quarter.
2. Allow user to select products Coke(25), Pepsi(35), Soda(45)
3. Allow user to take refund by canceling the request.
4. Return selected product and remaining change if any
5. Allow reset operation for vending machine supplier.

The requirement statement is the most important part of the problem. You need to read problem statement multiple times to get a high-level understanding of the problem and what are you trying to solve. Usually, requirements are not very clear and you need to make a list of your own by reading through problem statement.

I like point based requirement because it's easy to track. Some of the requirement are also implicit but it's better to make it explicit in your list e.g. In this problem, vending machine should not accept a request if it doesn't have sufficient change to return.

The requirement statement is the most important part of the problem. You need to read problem statement multiple times to get a high-level understanding of the problem and what are you trying to solve. Usually, requirements are not very clear and you need to make a list of your own by reading through problem statement.

I like point based requirement because it's easy to track. Some of the requirement are also implicit but it's better to make it explicit in your list e.g. In this problem, vending machine should not accept a request if it doesn't have sufficient change to return.

Solution and Coding

My implementation of Java Vending Machine has following classes and interfaces :

VendingMachine
It defines the public API of vending machine, usually all high-level functionality should go in this class

VendingMachineImpl
Sample implementation of Vending Machine

VendingMachineFactory
A Factory class to create different kinds of Vending Machine

Item
Java Enum to represent Item served by Vending Machine

Inventory
Java class to represent an Inventory, used for creating case and item inventory inside Vending Machine

Coin
Another Java Enum to represent Coins supported by Vending Machine

Bucket
A parameterized class to hold two objects. It's kind of Pair class.

NotFullPaidException
An Exception thrown by Vending Machine when a user tries to collect an item, without paying the full amount.

NotSufficientChangeException
Vending Machine throws this exception to indicate that it doesn't have sufficient change to complete this request.

SoldOutExcepiton
Vending Machine throws this exception if the user request for a product which is sold out.

How to design Vending Machine in Java
Here is the complete code of Vending Machine in Java, make sure to test this code, and let me know if you face any issue.

VendingMachine.java
The public API of vending machine, usually all high-level functionality should go in this class

```
package vending;

import java.util.List;

/**
  * Decleare public API for Vending Machine
  * @author ocj4u
  */
public interface VendingMachine {
    public long selectItemAndGetPrice(Item item);
    public void insertCoin(Coin coin);
    public List<Coin> refund();
    public Bucket<Item, List<Coin>> collectItemAndChange();
    public void reset();
}
```

VendingMachineImpl.java
A sample implementation of VendingMachine interface represents a real world Vending Machine , which you see in your office, bus stand, railway station and public places.

```
package vending;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

/**
  * Sample implementation of Vending Machine in Java
  * @author ocj4u
  */
public class VendingMachineImpl implements VendingMachine {
    private Inventory<Coin> cashInventory = new Inventory<Coin>();
    private Inventory<Item> itemInventory = new Inventory<Item>();
    private long totalSales;
    private Item currentItem;
    private long currentBalance;
```

```
 public VendingMachineImpl(){
     initialize();
 }

 private void initialize(){
     //initialize machine with 5 coins of each denomination
     //and 5 cans of each Item
     for(Coin c : Coin.values()){
         cashInventory.put(c, 5);
     }

     for(Item i : Item.values()){
         itemInventory.put(i, 5);
     }

 }

@Override
 public long selectItemAndGetPrice(Item item) {
     if(itemInventory.hasItem(item)){
         currentItem = item;
         return currentItem.getPrice();
     }
     throw new SoldOutException("Sold Out, Please buy another item");
 }

 @Override
 public void insertCoin(Coin coin) {
     currentBalance = currentBalance + coin.getDenomination();
     cashInventory.add(coin);
 }

 @Override
 public Bucket<Item, List<Coin>> collectItemAndChange() {
     Item item = collectItem();
     totalSales = totalSales + currentItem.getPrice();
```

```
        List<Coin> change = collectChange();

        return new Bucket<Item, List<Coin>>(item, change);
    }

    private Item collectItem() throws NotSufficientChangeException,
            NotFullPaidException{
        if(isFullPaid()){
            if(hasSufficientChange()){
                itemInventory.deduct(currentItem);
                return currentItem;
            }
            throw new NotSufficientChangeException("Not Sufficient change
in
                                                    Inventory");

        }
        long remainingBalance = currentItem.getPrice() - currentBalance;
        throw new NotFullPaidException("Price not full paid, remaining :
",
                                        remainingBalance);
    }

    private List<Coin> collectChange() {
        long changeAmount = currentBalance - currentItem.getPrice();
        List<Coin> change = getChange(changeAmount);
        updateCashInventory(change);
        currentBalance = 0;
        currentItem = null;
        return change;
    }

    @Override
    public List<Coin> refund(){
        List<Coin> refund = getChange(currentBalance);
        updateCashInventory(refund);
        currentBalance = 0;
        currentItem = null;
```

```
        return refund;
    }


    private boolean isFullPaid() {
        if(currentBalance >= currentItem.getPrice()){
            return true;
        }
        return false;
    }


    private List<Coin> getChange(long amount) throws
NotSufficientChangeException{
        List<Coin> changes = Collections.EMPTY_LIST;

        if(amount > 0){
            changes = new ArrayList<Coin>();
            long balance = amount;
            while(balance > 0){
                if(balance >= Coin.QUARTER.getDenomination()
                            && cashInventory.hasItem(Coin.QUARTER)){
                    changes.add(Coin.QUARTER);
                    balance = balance - Coin.QUARTER.getDenomination();
                    continue;

                }else if(balance >= Coin.DIME.getDenomination()
                                && cashInventory.hasItem(Coin.DIME)) {
                    changes.add(Coin.DIME);
                    balance = balance - Coin.DIME.getDenomination();
                    continue;

                }else if(balance >= Coin.NICKLE.getDenomination()
                                && cashInventory.hasItem(Coin.NICKLE)) {
                    changes.add(Coin.NICKLE);
                    balance = balance - Coin.NICKLE.getDenomination();
                    continue;
```

```
                }else if(balance >= Coin.PENNY.getDenomination()
                                && cashInventory.hasItem(Coin.PENNY)) {
                    changes.add(Coin.PENNY);
                    balance = balance - Coin.PENNY.getDenomination();
                    continue;

                }else{
                    throw new
NotSufficientChangeException("NotSufficientChange,
                                        Please try another product");
                }
            }
        }

        return changes;
    }

    @Override
    public void reset(){
        cashInventory.clear();
        itemInventory.clear();
        totalSales = 0;
        currentItem = null;
        currentBalance = 0;
    }

    public void printStats(){
        System.out.println("Total Sales : " + totalSales);
        System.out.println("Current Item Inventory : " + itemInventory);
        System.out.println("Current Cash Inventory : " + cashInventory);
    }


    private boolean hasSufficientChange(){
        return hasSufficientChangeForAmount(currentBalance -
currentItem.getPrice());
    }
```

```
    private boolean hasSufficientChangeForAmount(long amount){
        boolean hasChange = true;
        try{
            getChange(amount);
        }catch(NotSufficientChangeException nsce){
            return hasChange = false;
        }

        return hasChange;
    }

    private void updateCashInventory(List change) {
        for(Coin c : change){
            cashInventory.deduct(c);
        }
    }

    public long getTotalSales(){
        return totalSales;
    }

}
```

VendingMachineFactory.java
A Factory class to create different kinds of Vending Machine

```
package vending;

/**
  * Factory class to create instance of Vending Machine, this can be
extended to create instance of
  * different types of vending machines.
  * @author ocj4u
  */
public class VendingMachineFactory {
    public static VendingMachine createVendingMachine() {
        return new VendingMachineImpl();
    }
```

```
}
```

Item.java
Java Enum to represent Item served by Vending Machine

```
package vending;
/**
  * Items or products supported by Vending Machine.
  * @author ocj4u
  */
public enum Item{
    COKE("Coke", 25), PEPSI("Pepsi", 35), SODA("Soda", 45);

    private String name;
    private int price;

    private Item(String name, int price){
        this.name = name;
        this.price = price;
    }

    public String getName(){
        return name;
    }

    public long getPrice(){
        return price;
    }
}
```

Coin.java
Another Java Enum to represent Coins supported by Vending Machine

```
package vending;

/**
  * Coins supported by Vending Machine.
```

```
  * @author ocj4u
  */
public enum Coin {
    PENNY(1), NICKLE(5), DIME(10), QUARTER(25);

    private int denomination;

    private Coin(int denomination){
        this.denomination = denomination;
    }

    public int getDenomination(){
        return denomination;
    }
}
```

Inventory.java
A Java class to represent an Inventory, used for creating case and item inventory inside Vending Machine.

```
package vending;
import java.util.HashMap;
import java.util.Map;

/**
  * An Adapter over Map to create Inventory to hold cash and
  * Items inside Vending Machine
  * @author ocj4u
  */
public class Inventory<T> {
    private Map<T, Integer> inventory = new HashMap<T, Integer>();

    public int getQuantity(T item){
        Integer value = inventory.get(item);
        return value == null? 0 : value ;
    }

    public void add(T item){
        int count = inventory.get(item);
```

```
        inventory.put(item, count+1);
    }

    public void deduct(T item) {
        if (hasItem(item)) {
            int count = inventory.get(item);
            inventory.put(item, count - 1);
        }
    }

    public boolean hasItem(T item){
        return getQuantity(item) > 0;
    }

    public void clear(){
        inventory.clear();
    }

    public void put(T item, int quantity) {
        inventory.put(item, quantity);
    }
}
```

Bucket.java
A parameterized utility class to hold two objects.

```
package vending;
/**
  * A parameterized utility class to hold two different object.
  * @author ocj4u
  */
public class Bucket<E1, E2> {
    private E1 first;
    private E2 second;

    public Bucket(E1 first, E2 second){
        this.first = first;
        this.second = second;
```

```
    }

    public E1 getFirst(){
        return first;
    }

    public E2 getSecond(){
        return second;
    }
}
```

NotFullPaidException.java
An Exception, thrown by Vending Machine when a user tries to collect an item, without paying the full amount.

```
package vending;
public class NotFullPaidException extends RuntimeException {
    private String message;
    private long remaining;

    public NotFullPaidException(String message, long remaining) {
        this.message = message;
        this.remaining = remaining;
    }

    public long getRemaining(){
        return remaining;
    }

    @Override
    public String getMessage(){
        return message + remaining;
    }

}
```

NotSufficientChangeException.java
Vending Machine throws this exception to indicate that it doesn't have sufficient

change to complete this request.

```
package vending;
public class NotSufficientChangeException extends RuntimeException {
    private String message;

    public NotSufficientChangeException(String string) {
        this.message = string;
    }

    @Override
    public String getMessage(){
        return message;
    }

}
```

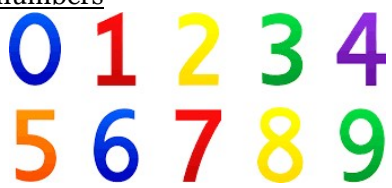
SoldOutException.java
The Vending Machine throws this exception if the user request for a product which is sold out

```
package vending;
public class SoldOutException extends RuntimeException {
    private String message;

    public SoldOutException(String string) {
        this.message = string;
    }

    @Override
    public String getMessage(){
        return message;
    }

}
```

That's all in this first part of **how to design a vending machine in Java**

15. How do you find the missing number in a given integer array of 1 to 100? (solution)

Java Program to find missing numbers



Let's understand the problem statement, we have numbers from 1 to 100 that are put into an integer array, what's the best way to find out which number is missing? If Interviewer especially mentions 1 to 100 then you can apply the above trick about the sum of the series as shown below as well. If it has more than one missing element that you can use BitSet class, of course only if your interviewer allows it.

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 elements.

I have provided a BitSet solution with another purpose, to introduce with this nice utility class. In many interviews, I have asked about this class to Java developers, but many of them not even aware of this. I think this problem is a nice way to learn how to use BitSet in Java as well.

By the way, if you are going for interview, then apart from this question, its also good to know how to find duplicate number in array and program to find second highest number in an integer array. More often than not, those are asked as follow-up question after this.

```
import java.util.Arrays;
import java.util.BitSet;

/**
 * Java program to find missing elements in a Integer array containing
 * numbers from 1 to 100.
 *
 * @author ocj4u
 */
public class MissingNumberInArray {

    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);
        }

    }
   /**
    * Java method to find missing number in array of size n containing
    * numbers from 1 to n only.
    * can be used to find missing elements on integer array of
    * numbers from 1 to 100 or 1 - 1000
    */
    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
```

That's all on this program to find missing element in an array of 100 elements. As I said, it's good to know the trick, which just require you to calculate sum of numbers and then subtract that from actual sum, but you can not use that if array has more than one missing numbers. On the other hand, BitSet solution is more general, as you can use it to find more than one missing values on integer array.

16. How do you find the duplicate number on a given integer array? (solution)

Java Program to remove duplicates from integer array without Collection

In this program, we have not used any collection class to remove duplicates, earlier, I had shown you a way to remove duplicates from ArrayList, which was using LinkedHashSet. You can still use that solution if the interviewer doesn't mention without Collection specifically.

All you need to do is to convert your array into ArrayList first then subsequently create a LinkedHashSet from that ArrayList. In this example, we are removing duplicates from the array by not copying them into result array, this solution is not actually deleting duplicates instead it replacing it with default value i.e. zero.

```
import java.util.Arrays;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * Java program to remove duplicates from this array. You don't
 * need to physically delete duplicate elements, replacing with
null, or
 * empty or default value is ok.
 *
 * @author http://javarevisited.blogspot.com
 */
public class TechnicalInterviewTest {

    private static final Logger logger =
LoggerFactory.getLogger(TechnicalInterviewTest.class);
```

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

        int[][] test = new int[][]{
            {1, 1, 2, 2, 3, 4, 5},
            {1, 1, 1, 1, 1, 1, 1},
            {1, 2, 3, 4, 5, 6, 7},
            {1, 2, 1, 1, 1, 1, 1},};

        for (int[] input : test) {
            System.out.println("Array with Duplicates       : " +
Arrays.toString(input));
            System.out.println("After removing duplicates   : " +
Arrays.toString(removeDuplicates(input)));
        }
    }

    /*
     * Method to remove duplicates from array in Java, without using
     * Collection classes e.g. Set or ArrayList. Algorithm for this
     * method is simple, it first sort the array and then compare
adjacent
     * objects, leaving out duplicates, which is already in the
result.
     */
    public static int[] removeDuplicates(int[]
numbersWithDuplicates) {

        // Sorting array to bring duplicates together
        Arrays.sort(numbersWithDuplicates);

        int[] result = new int[numbersWithDuplicates.length];
        int previous = numbersWithDuplicates[0];
        result[0] = previous;

        for (int i = 1; i < numbersWithDuplicates.length; i++) {
            int ch = numbersWithDuplicates[i];

            if (previous != ch) {
                result[i] = ch;
            }
            previous = ch;
        }
        return result;

    }
}
```

Output :

```
Array with Duplicates       : [1, 1, 2, 2, 3, 4, 5]
After removing duplicates   : [1, 0, 2, 0, 3, 4, 5]
Array with Duplicates       : [1, 1, 1, 1, 1, 1, 1]
After removing duplicates   : [1, 0, 0, 0, 0, 0, 0]
Array with Duplicates       : [1, 2, 3, 4, 5, 6, 7]
After removing duplicates   : [1, 2, 3, 4, 5, 6, 7]
Array with Duplicates       : [1, 2, 1, 1, 1, 1, 1]
```

```
After removing duplicates   : [1, 0, 0, 0, 0, 0, 2]
```

That's it about **how to remove duplicates from an array in Java without using Collection class**. As I said before, this solution is not perfect and has some serious limitation, which is an exercise for you to find out. One hint I can give is that array itself can contain default value as duplicates e.g. 0 for int, even if you use any Magic number e.g. Integer.MAX_VALUE, you can not be certain that they will not be part of the input.

Regarding removing duplicate permanently from result array, one approach could be to count a number of duplicates and then create an array of right size i.e. length - duplicates, and then copying content from intermediate result array to final array, leaving out elements which are marked duplicate.

17. How do you find duplicate numbers in an array if it contains multiple duplicates? (solution)

Here is the full code sample of finding first duplicate character in a given String. This program has three method to find first non-repeated character. Each uses their own algorithm to do this programming task. First algorithm is implemented in getFirstNonRepeatedChar(String str) method. It first gets character array from given String and loop through it to build a hash table with character as key and their count as value. In next step, It loop through LinkedHashMap to find an entry with value 1, that's your first non-repeated character, because LinkedHashMap maintains insertion order, and we iterate through character array from beginning to end. Bad part is it requires two iteration, first one is proportional to number of character in String, and second is proportional to number of duplicate characters in String. In worst case, where String contains non-repeated character at end, it will take 2*N time to solve this problem.

Second way to find first non-repeated or unique character is coded on firstNonRepeatingChar(String word) ,this solution finds first non repeated character in a String in just one pass. It applies classical space-time trade-off technique. It uses two storage to cut down one iteration, standard space vs time trade-off. Since we store repeated and non-repeated characters separately, at the end of iteration, first element from List is our first non repeated character from String. This one is slightly better than previous one, though it's your choice to return null or empty string if there is no non-repeated character in the String. Third way to solve this programming question is implemented in firstNonRepeatedCharacter(String word) method. It's very similar to first one except the fact that instead of LinkedHashMap, we have used HashMap. Since later doesn't guarantee any order, we have to rely on original String for finding first non repeated character. Here is the algorithm of this third solution. First step : Scan String and store count of each character in HashMap. Second Step : traverse String and get count for each character from Map. Since we are going through String from first to last character, when count for any character is 1, we break, it's the first non repeated character. Here order is achieved by going through String again.

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
```

```java
import java.util.Map.Entry;
import java.util.Set;

/**
 * Java Program to find first duplicate, non-repeated character in a
String.
 * It demonstrate three simple example to do this programming problem.
 *
 * @author ocj4u
 */
public class Programming {

    /*
     * Using LinkedHashMap to find first non repeated character of String
     * Algorithm :
     *            Step 1: get character array and loop through it to build
a
     *                    hash table with char and their count.
     *            Step 2: loop through LinkedHashMap to find an entry with
     *                    value 1, that's your first non-repeated
character,
     *                    as LinkedHashMap maintains insertion order.
     */
    public static char getFirstNonRepeatedChar(String str) {
        Map<Character,Integer> counts = new LinkedHashMap<>(str.length());

        for (char c : str.toCharArray()) {
            counts.put(c, counts.containsKey(c) ? counts.get(c) + 1 : 1);
        }

        for (Entry<Character,Integer> entry : counts.entrySet()) {
            if (entry.getValue() == 1) {
                return entry.getKey();
            }
        }
        throw new RuntimeException("didn't find any non repeated
Character");
    }


    /*
     * Finds first non repeated character in a String in just one pass.
     * It uses two storage to cut down one iteration, standard space vs
time
     * trade-off.Since we store repeated and non-repeated character
separately,
     * at the end of iteration, first element from List is our first non
     * repeated character from String.
     */
    public static char firstNonRepeatingChar(String word) {
        Set<Character> repeating = new HashSet<>();
        List<Character> nonRepeating = new ArrayList<>();
        for (int i = 0; i < word.length(); i++) {
            char letter = word.charAt(i);
            if (repeating.contains(letter)) {
                continue;
            }
```

```
            if (nonRepeating.contains(letter)) {
                nonRepeating.remove((Character) letter);
                repeating.add(letter);
            } else {
                nonRepeating.add(letter);
            }
        }
        return nonRepeating.get(0);
    }


    /*
     * Using HashMap to find first non-repeated character from String in
Java.
     * Algorithm :
     * Step 1 : Scan String and store count of each character in HashMap
     * Step 2 : traverse String and get count for each character from Map.
     *          Since we are going through String from first to last
character,
     *          when count for any character is 1, we break, it's the first
     *          non repeated character. Here order is achieved by going
     *          through String again.
     */
    public static char firstNonRepeatedCharacter(String word) {
        HashMap<Character,Integer> scoreboard = new HashMap<>();
        // build table [char -> count]
        for (int i = 0; i < word.length(); i++) {
            char c = word.charAt(i);
            if (scoreboard.containsKey(c)) {
                scoreboard.put(c, scoreboard.get(c) + 1);
            } else {
                scoreboard.put(c, 1);
            }
        }
        // since HashMap doesn't maintain order, going through string again
        for (int i = 0; i < word.length(); i++) {
            char c = word.charAt(i);
            if (scoreboard.get(c) == 1) {
                return c;
            }
        }
        throw new RuntimeException("Undefined behaviour");
    }

}
```

JUnit Test to find First Unique Character

Here are some JUnit test cases to test each of this method. We test different kind of inputs, one which contains duplicates, and other which doesn't contains duplicates. Since program has not defined what to do in case of empty String, null String and what to return if only contains duplicates, you are feel free to do in a way which make sense.

```
import static org.junit.Assert.*;
import org.junit.Test;

public class ProgrammingTest {

    @Test
    public void testFirstNonRepeatedCharacter() {
        assertEquals('b',
Programming.firstNonRepeatedCharacter("abcdefghija"));
        assertEquals('h', Programming.firstNonRepeatedCharacter("hello"));
        assertEquals('J', Programming.firstNonRepeatedCharacter("Java"));
        assertEquals('i',
Programming.firstNonRepeatedCharacter("simplest"));
    }

    @Test
    public void testFirstNonRepeatingChar() {
        assertEquals('b',
Programming.firstNonRepeatingChar("abcdefghija"));
        assertEquals('h', Programming.firstNonRepeatingChar("hello"));
        assertEquals('J', Programming.firstNonRepeatingChar("Java"));
        assertEquals('i', Programming.firstNonRepeatingChar("simplest"));
    }

    @Test
    public void testGetFirstNonRepeatedChar() {
        assertEquals('b',
Programming.getFirstNonRepeatedChar("abcdefghija"));
        assertEquals('h', Programming.getFirstNonRepeatedChar("hello"));
        assertEquals('J', Programming.getFirstNonRepeatedChar("Java"));
        assertEquals('i', Programming.getFirstNonRepeatedChar("simplest"));
    }
}
```

If you can enhance this test cases to check more scenario, just go for it. There is no better way to impress interviewer then writing detailed, creative test cases, which many programmer can't think of or just don't put effort to come up.

That's all on **How to find first non-repeated character of a String in Java**. We have seen three ways to solve this problem, although they use pretty much similar logic, they are different from each other. This program is also very good for beginners to master Java Collection framework. It gives you an opportunity to explore different Map implementations and understand difference between HashMap and LinkedHashMap to decide when to use them

18. Difference between a stable and unstable sorting algorithm?

A sorting algorithm is said to be stable if it maintains the relative order of numbers/records in the case of tie i.e. if you need to sort 1 1 2 3 then if you don't change order of those first two ones than your algorithm is stable, but if you swap them then it becomes unstable, despite the overall result or sorting order remain same.

This difference becomes more obvious when you sort objects e.g. sorting key-value pairs by keys. In the case of a stable algorithm, the original order of key-value pair is retained as shown in the following example.

Actually, Interviewer might ask that question as a follow-up of quicksort vs merge sort if you forget to mention those concepts.

One of the main difference between quicksort and mergesort is that the quicksort is unstable but merge sort is a stable sorting algorithm.

Stable vs Unstable Algorithm

Suppose you need to sort following key-value pairs in the increasing order of keys:

INPUT: (4,5), (3, 2) (4, 3) (5,4) (6,4)

Now, there is two possible solution for the two pairs where the key is the same i.e. (4,5) and (4,3) as shown below:

OUTPUT1: (3, 2), (4, 5), (4,3), (5,4), (6,4)

OUTPUT2: (3, 2), (4, 3), (4,5), (5,4), (6,4)

The sorting algorithm which will produce the first output will be known as stable sorting algorithm because the original order of equal keys are maintained, you can see that (4, 5) comes before (4,3) in the sorted order, which was the original order i.e. in the given input, (4, 5) comes before (4,3) .

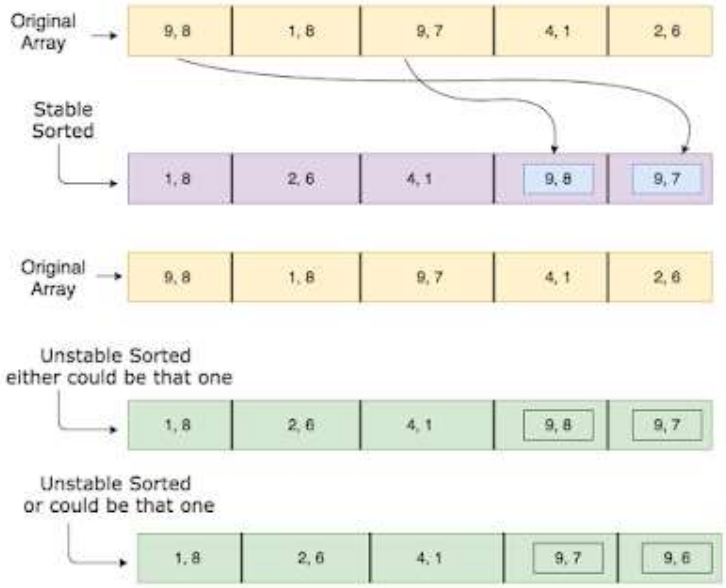
On the other hand, the algorithm which produces second output will know as an unstable sorting algorithm because the order of objects with the same key is not maintained in the sorted order. You can see that in the second output, the (4,3) comes before (4,5) which was not the case in the original input.

Now, the big question is what are some examples of stable and unstable sorting algorithms? Well, you can divide all well-known sorting algorithms into stable and unstable. Some examples of stable algorithms are Merge Sort, Insertion Sort, Bubble Sort, and Binary Tree Sort. While, QuickSort, Heap Sort, and Selection sort are the unstable sorting algorithm.

If you remember, Collections.sort() method from Java Collection framework uses iterative merge sort which is a stable algorithm. It also does far fewer comparison than NLog(N) in case input array is partially sorted.

Stable vs Unstable Algorithm Example

It's said that a picture is worth more than 1000 words, so let's see an image which explains how stable and unstable sort works:



That's all about the **difference between stable and unstable sorting algorithm**. Just remember, that if the original order of equal keys or number is maintained in the sorted output then the algorithm is known as sorting algorithm. Some popular examples of stable sorting algorithms are merge sort, insertion sort, and bubble sort.

19. How is an iterative quicksort algorithm implemented? (solution)

Iterative Quicksort Algorithm

I learned about quicksort in my engineering classes, one of the few algorithm which I managed to understand then. Since it's a divide-and-conquer algorithm, you choose a pivot and divide the array. Unlike merge sort, which is also a divide-and-conquer algorithm and where all important work happens on combine steps, In quicksort, the real work happens in divide step and the combining step does nothing important.

Btw, the working of the algorithm will remain same whether you implement an iterative solution or a recursion solution. In iterative solution, we'll use Stack instead of recursion. Here are the steps to implement iterative quicksort in Java:

- Push the range (0...n) into the Stack
- Partition the given array with a pivot
- Pop the top element.
- Push the partitions (index range) into a stack if the range has more than one element
- Do the above 3 steps, till the stack, is empty

You might know that even though writing recursive algorithms are easy they are always slower than their Iterative counterpart. So, when Interviewer will ask you to choose a method in terms of time complexity where memory is not a concern, which version will you choose?

Well, both recursive and iterative quicksorts are O(N log N) average case and O(n^2) in the worst case but the recursive version is shorter and clearer. Iterative is faster and makes you simulate the recursion call stack.

QuickSort example in Java without recursion.

Here is our sample Java program to implement Quicksort using for loop and stack, without using recursion. This is also known as iterative quicksort algorithm.

```
import java.util.Arrays;
import java.util.Scanner;
import java.util.Stack;

/**
 * Java Program to implement Iterative QuickSort Algorithm, without
recursion.
 *
 * @author WINDOWS 8
 */
public class Sorting {

    public static void main(String args[]) {

        int[] unsorted = {34, 32, 43, 12, 11, 32, 22, 21, 32};
        System.out.println("Unsorted array : " +
Arrays.toString(unsorted));

        iterativeQsort(unsorted);
        System.out.println("Sorted array : " + Arrays.toString(unsorted));
    }


    /*
     * iterative implementation of quicksort sorting algorithm.
     */
    public static void iterativeQsort(int[] numbers) {
        Stack stack = new Stack();
        stack.push(0);
        stack.push(numbers.length);
```

```
        while (!stack.isEmpty()) {
            int end = stack.pop();
            int start = stack.pop();
            if (end - start < 2) {
                continue;
            }
            int p = start + ((end - start) / 2);
            p = partition(numbers, p, start, end);

            stack.push(p + 1);
            stack.push(end);

            stack.push(start);
            stack.push(p);

        }
    }

    /*
     * Utility method to partition the array into smaller array, and
     * comparing numbers to rearrange them as per quicksort algorithm.
     */
    private static int partition(int[] input, int position, int start, int
end) {
        int l = start;
        int h = end - 2;
        int piv = input[position];
        swap(input, position, end - 1);

        while (l < h) {
            if (input[l] < piv) {
                l++;
            } else if (input[h] >= piv) {
                h--;
            } else {
                swap(input, l, h);
            }
```

```
        }
        int idx = h;
        if (input[h] < piv) {
            idx++;
        }
        swap(input, end - 1, idx);
        return idx;
    }

    /**
     * Utility method to swap two numbers in given array
     *
     * @param arr - array on which swap will happen
     * @param i
     * @param j
     */
    private static void swap(int[] arr, int i, int j) {
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }

}

Output:
Unsorted array : [34, 32, 43, 12, 11, 32, 22, 21, 32]
Sorted array : [11, 12, 21, 22, 32, 32, 32, 34, 43]
```

That's all about **how to implement quicksort in Java without recursion**. Just remember, when you use for loop and stack to implement quicksort, it's known as iterative implementation and when you call the method itself, it's known as recursive implementation. The recursive solution of quicksort is easier to write and understand but the iterative solution is much faster. Though average and worst case time complexity of both recursive and iterative quicksorts are O(N log N) average case and O(n^2).

20. How do you find the largest and smallest number in an unsorted integer array?

Java Program to find smallest and largest number in an integer array

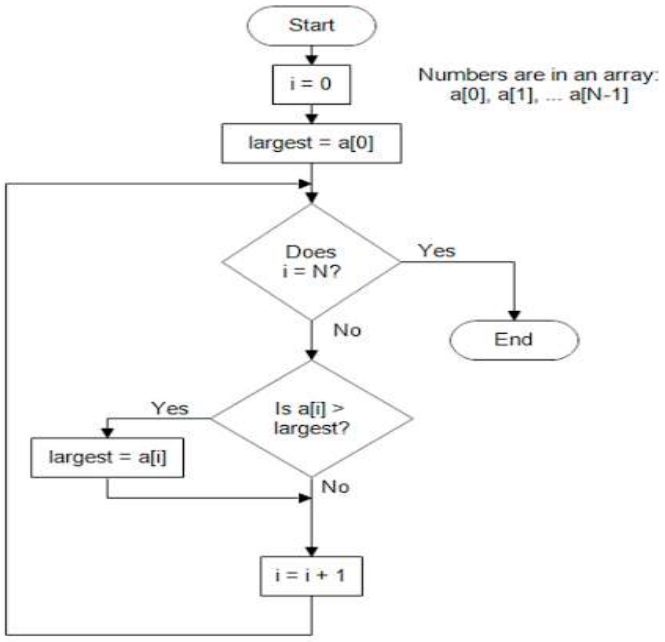
Here is full code example of Java program to find smallest and largest number from an integer array. You can create a Java source file with name MaximumMinimumArrayDemo.java and copy code there to compile and execute in your favorite IDE. If you don't have IDE setup, you can also compile and run this program by following steps I have shown on HelloWorld in Java.

If you look at the code here, we have created a method called largestAndSmallest(int[] numbers) to print largest and smallest number from int array passed to the program. We use two variables largest and smallest to store the maximum and minimum values from the array. Initially largest is initialized with Integer.MIN_VALUE and smallest is initialized with Integer.MAX_VALUE.

In each iteration of the loop, we compare current number with largest and smallest and update them accordingly. Since if a number is larger than largest, it can't be smaller than smallest, which means you don't need to check if the first condition is true, that's why we have used if-else code block, where else part will only execute if the first condition is not true.

Here is another logic to find the largest element from an array in Java, here instead of assigning the variable with Integer.MAX_VALUE, we have assigned the first element from the array.

Logic to Find the largest number from Array



Since array doesn't override the toString method in Java, we have used Arrays.toString() to print contents of an array. Remember this function is outside of core logic, so it's Ok to use it. Since this is a static method we can directly call this from the main method in Java, and so does our test code. We pass the random array to this method and see if largest and smallest number returned by the method is correct or not. For automated testing, a Unit test is better but for demonstration, you can use the main method.

Java Program to find the largest and smallest element in array:

```
import java.util.Arrays;
/**
 * Java program to find largest and smallest number from an array in Java.
 * You cannot use any library method both from Java and third-party
library.
 *
 * @author http://java67.blogspot.com
 */
public class MaximumMinimumArrayDemo{

    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
```

That's all about How to find largest and smallest number from integer array in Java. As I said this question can also be asked as to find the maximum and minimum numbers in an Array in Java, so don't get confused there. By the way, there are more ways to do the same task and you can practice it to code solution differently.

21. How do you reverse a linked list in place? (solution)

Reverse a linked list

Given pointer to the head node of a linked list, the task is to reverse the linked list. We need to reverse the list by changing links between nodes.

Examples:

Input: Head of following linked list

1->2->3->4->NULL

Output: Linked list should be changed to,

4->3->2->1->NULL

Input: Head of following linked list

1->2->3->4->5->NULL

Output: Linked list should be changed to,

5->4->3->2->1->NULL

Input: NULL

Output: NULL

Input: 1->NULL

Output: 1->NULL

Iterative Method

1. *Initialize three pointers prev as NULL, curr as head and next as NULL.*
2. *Iterate trough the linked list. In loop, do following.*

```
// Before changing next of current,
// store next node
next = curr->next
// Now change next of current
// This is where actual reversing happens
curr->next = prev

// Move prev and curr one step forward
prev = curr
curr = next
```

```
// Java program for reversing the linked list

class LinkedList {

    static Node head;

    static class Node {

        int data;
        Node next;

        Node(int d)
        {
            data = d;
            next = null;
        }
    }

    /* Function to reverse the linked list */
    Node reverse(Node node)
    {
        Node prev = null;
        Node current = node;
        Node next = null;
        while (current != null) {
            next = current.next;
            current.next = prev;
            prev = current;
            current = next;
        }
        node = prev;
```

```
        return node;
    }

    // prints content of double linked list
    void printList(Node node)
    {
        while (node != null) {
            System.out.print(node.data + " ");
            node = node.next;
        }
    }

    public static void main(String[] args)
    {
        LinkedList list = new LinkedList();
        list.head = new Node(85);
        list.head.next = new Node(15);
        list.head.next.next = new Node(4);
        list.head.next.next.next = new Node(20);

        System.out.println("Given Linked list");
        list.printList(head);
        head = list.reverse(head);
        System.out.println("");
        System.out.println("Reversed linked list ");
        list.printList(head);
    }
}
```

Output:

```
Given linked list

85 15 4 20

Reversed Linked list

20 4 15 85
```

Time Complexity: O(n)

Space Complexity: O(1)

A Simpler and Tail Recursive Method
Below is the implementation of this method.

```
// Java program for reversing the Linked list
class LinkedList {

    static Node head;

    static class Node {

        int data;
        Node next;
```

```
    Node(int d)
    {
        data = d;
        next = null;
    }
}

// A simple and tail recursive function to reverse
// a linked list.  prev is passed as NULL initially.
Node reverseUtil(Node curr, Node prev)
{

    /* If last node mark it head*/
    if (curr.next == null) {
        head = curr;

        /* Update next to prev node */
        curr.next = prev;

        return head;
    }

    /* Save curr->next node for recursive call */
    Node next1 = curr.next;

    /* and update next ..*/
    curr.next = prev;

    reverseUtil(next1, curr);
    return head;
}
// prints content of double linked list
void printList(Node node)
{
    while (node != null) {
        System.out.print(node.data + " ");
        node = node.next;
    }
}

public static void main(String[] args)
{
    LinkedList list = new LinkedList();
    list.head = new Node(1);
    list.head.next = new Node(2);
    list.head.next.next = new Node(3);
    list.head.next.next.next = new Node(4);
    list.head.next.next.next.next = new Node(5);
    list.head.next.next.next.next.next = new Node(6);
    list.head.next.next.next.next.next.next = new Node(7);
    list.head.next.next.next.next.next.next.next = new Node(8);

    System.out.println("Original Linked list ");
    list.printList(head);
    Node res = list.reverseUtil(head, null);
    System.out.println("");
    System.out.println("");
    System.out.println("Reversed linked list ");
    list.printList(res);     }    }
```

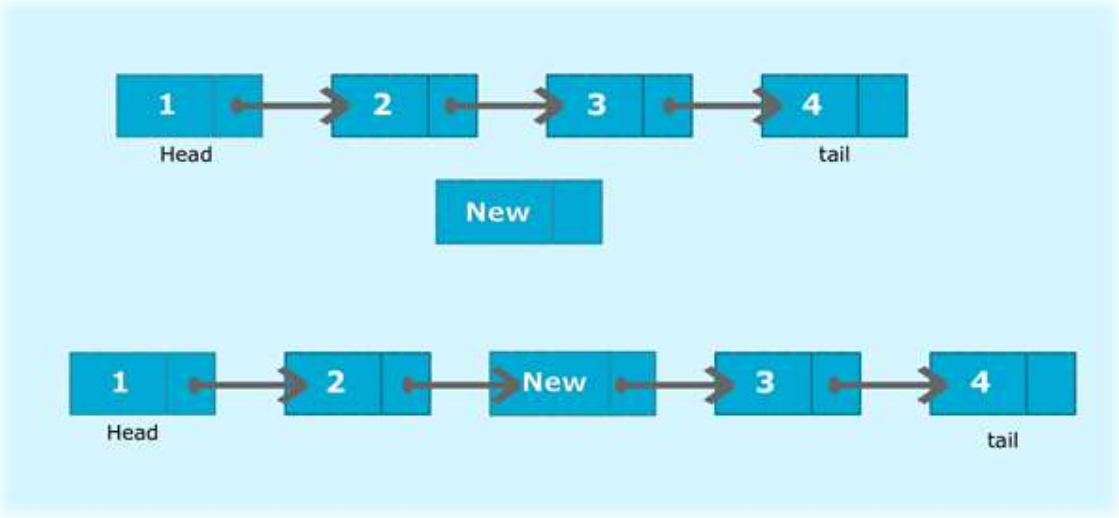
Output:

```
Given linked list

1 2 3 4 5 6 7 8

Reversed linked list

8 7 6 5 4 3 2 1
```

22. How to add an element at the middle of the linked list? (solution)

Java Program to insert a new node at the middle of the singly linked list

In this program, we will create a singly linked list and add a new node at the middle of the list. To accomplish this task, we will calculate the size of the list and divide it by 2 to get the mid-point of the list where the new node needs to be inserted.



Consider the above diagram; node 1 represents the head of the original list. Let node New is the new node which needs to be added at the middle of the list. First, we calculate size which in this case is 4. So, to get the mid-point, we divide it by 2 and store it in a variable count. Node current will point to head. First, we iterate through the list till current points to the mid position. Define another node temp which point to node next to current. Insert the New node between current and temp

Algorithm

- Create a class Node which has two attributes: data and next. Next is a pointer to the next node in the list.
- Create another class InsertMid which has three attributes: head, tail, and size that keep tracks of a number of nodes present in the list.

- addNode() will add a new node to the list:
  - Create a new node.
  - It first checks, whether the head is equal to null which means the list is empty.
  - If the list is empty, both head and tail will point to a newly added node.
  - If the list is not empty, the new node will be added to end of the list such that tail's next will point to a newly added node. This new node will become the new tail of the list.

a. addInMid() will add a new node at the middle of the list:

- It first checks, whether the head is equal to null which means the list is empty.
- If the list is empty, both head and tail will point to a newly added node.
- If the list is not empty, then calculate the size of the list and divide it by 2 to get mid-point of the list.
- Define node current that will iterate through the list until current will point to the mid node.
- Define another node temp which will point to node next to current.
- The new node will be inserted after current and before temp such that current will point to the new node and the new node will point to temp.

a. display() will display the nodes present in the list:

- Define a node current which will initially point to the head of the list.
- Traverse through the list till current points to null.
- Display each node by making current to point to node next to it in each iteration.

Program:

```
1.  public class InsertMid {
2.
3.     //Represent a node of the singly linked list
4.     class Node{
5.        int data;
6.        Node next;
7.
8.        public Node(int data) {
9.           this.data = data;
10.          this.next = null;
11.       }
12.    }
13.
```

```
14.    public int size;
15.    //Represent the head and tail of the singly linked list
16.    public Node head = null;
17.    public Node tail = null;
18.
19.    //addNode() will add a new node to the list
20.    public void addNode(int data) {
21.       //Create a new node
22.       Node newNode = new Node(data);
23.
24.       //Checks if the list is empty
25.       if(head == null) {
26.          //If list is empty, both head and tail will point to new node
27.          head = newNode;
28.          tail = newNode;
29.       }
30.       else {
31.          //newNode will be added after tail such that tail's next will point to newNode
32.          tail.next = newNode;
33.          //newNode will become new tail of the list
34.          tail = newNode;
35.       }
36.       //Size will count the number of nodes present in the list
37.       size++;
38.    }
39.
40.    //This function will add the new node at the middle of the list.
41.    public void addInMid(int data){
42.       //Create a new node
43.       Node newNode = new Node(data);
44.
45.       //Checks if the list is empty
46.       if(head == null) {
47.          //If list is empty, both head and tail would point to new node
48.          head = newNode;
49.          tail = newNode;
50.       }
51.       else {
52.          Node temp, current;
53.          //Store the mid position of the list
54.          int count = (size % 2 == 0) ? (size/2) : ((size+1)/2);
55.          //Node temp will point to head
56.          temp = head;
57.          current = null;
```

```
58.
59.         //Traverse through the list till the middle of the list is reached
60.         for(int i = 0; i < count; i++) {
61.            //Node current will point to temp
62.            current = temp;
63.            //Node temp will point to node next to it.
64.            temp = temp.next;
65.         }
66.
67.         //current will point to new node
68.         current.next = newNode;
69.         //new node will point to temp
70.         newNode.next = temp;
71.      }
72.      size++;
73.   }
74.
75.   //display() will display all the nodes present in the list
76.   public void display() {
77.      //Node current will point to head
78.      Node current = head;
79.      if(head == null) {
80.         System.out.println("List is empty");
81.         return;
82.      }
83.
84.      while(current != null) {
85.         //Prints each node by incrementing pointer
86.         System.out.print(current.data + " ");
87.         current = current.next;
88.      }
89.      System.out.println();
90.   }
91.
92.   public static void main(String[] args) {
93.
94.      InsertMid sList = new InsertMid();
95.
96.      //Adds data to the list
97.      sList.addNode(1);
98.      sList.addNode(2);
99.
100.            System.out.println("Original list: ");
101.            sList.display();
```

```
102.
103.            //Inserting node '3' in the middle
104.            sList.addInMid(3);
105.            System.out.println( "Updated List: ");
106.            sList.display();
107.
108.            //Inserting node '4' in the middle
109.            sList.addInMid(4);
110.            System.out.println("Updated List: ");
111.            sList.display();
112.        }
113.    }
```

Output:

```
Original list:
1 2
Updated List:
1 3 2
Updated List:
1 3 4 2
```

23. How do you sort a linked list in Java? (solution)

Java Program to sort the LinkedList in Java

Here is complete Java program to sort the LinkedList. We have used Collections.sort() method for sorting elements stored in LinkedList class.

```
import java.util.Collections;
import java.util.Comparator;
import java.util.LinkedList;

public class LinkedListSorting{

public static void main(String args[]) {

// Creating and initializing an LinkedList for sorting
LinkedList<String> singlyLinkedList = new LinkedList<>();
singlyLinkedList.add("Eclipse");
singlyLinkedList.add("NetBeans");
singlyLinkedList.add("IntelliJ");
```

```
singlyLinkedList.add("Resharper");
singlyLinkedList.add("Visual Studio");
singlyLinkedList.add("notepad");

System.out.println("LinkedList (before sorting): " + singlyLinkedList);

// Example 1 - Sorting LinkedList with Collecitons.sort() method in
natural order
Collections.sort(singlyLinkedList);

System.out.println("LinkedList (after sorting in natural): " +
singlyLinkedList);

// Example 2 - Sorting LinkedList using Collection.sort() and Comparator
in Java
Collections.sort(singlyLinkedList, new Comparator<String>() {
@Override
public int compare(String s1, String s2) {
return s1.length() - s2.length();
} } );

System.out.println("LinkedList (after sorting using Comparator): " +
singlyLinkedList);
}
}

Output
LinkedList (before sorting): [Eclipse, NetBeans, IntelliJ, Resharper,
Visual Studio, notepad]
LinkedList (after sorting in natural): [Eclipse, IntelliJ, NetBeans,
Resharper, Visual Studio, notepad]
LinkedList (after sorting using Comparator): [Eclipse, notepad, IntelliJ,
NetBeans, Resharper, Visual Studio]
```

That's all about how to sort a LinkedList in Java. In sorting a LinkedList is very inefficient because you need to traverse through elements, which is O(n) operation. There is no indexed access like an array, but using Collections.sort() method is as good as sorting ArrayList because it copies LinkedList elements into array, sorts it and then put it back into linked list.

24. How do you find all pairs of an integer array whose sum is equal to a given number? (solution)

Solution to Find Pair Of Integers in Array whose Sum is Given Number

```
import java.util.Arrays;
import java.util.HashSet;
import java.util.Set;

/**
 * Java Program to find all pairs on integer array whose sum is equal to k
 *
 * @author WINDOWS 7
 */
public class PrintArrayPairs {

    public static void main(String args[]) {
       prettyPrint( new int[]{ 12, 14, 17, 15, 19, 20, -11}, 9);
       prettyPrint( new int[]{ 2, 4, 7, 5, 9, 10, -1}, 9);
    }

    /**
     * Given a number finds two numbers from an array so that the sum is
equal to that number k.
     * @param numbers
     * @param k
     */
    public static void printPairsUsingTwoPointers(int[] numbers, int k){
        if(numbers.length < 2){
            return;
        }
        Arrays.sort(numbers);

        int left = 0; int right = numbers.length -1;
        while(left < right){
            int sum = numbers[left] + numbers[right];
            if(sum == k){
                System.out.printf("(%d, %d) %n", numbers[left],
numbers[right]);
```

```
                left = left + 1;
                right = right -1;

            }else if(sum < k){
                left = left +1;

            }else if (sum > k) {
                right = right -1;
            }
        }

    }

    /*
     * Utility method to print two elements in an array that sum to k.
     */
    public static void prettyPrint(int[] random, int k){
        System.out.println("input int array : " +
Arrays.toString(random));
        System.out.println("All pairs in an array of integers whose sum is
equal to a given value " + k);
        printPairsUsingTwoPointers(random, k);
    }

}

Output :
input int array : [12, 14, 17, 15, 19, 20, -11]
All pairs in an array of integers whose sum is equal to a given value 9
(-11, 20)
input int array : [2, 4, 7, 5, 9, 10, -1]
All pairs in an array of integers whose sum is equal to a given value 9
(-1, 10)
(2, 7)
(4, 5)
```

25. How do you implement an insertion sort algorithm? (solution)

How the Insertion Sort Algorithm works

If you know how to sort a hand of cards, you know how insertion sort works; but for many programmers, it's not easy to translate real-world knowledge into a working code example.

This is where natural programming ability comes into play. A good programmer has the ability to code any algorithm and convert a real-life example to an algorithm.

Now, how do you sort an array of an integer using this algorithm? You can say that we can treat this array as a deck of card, and we will use another array to pick and place an element from one place to another. Well, that will work, but it's a waste of space (memory) because what you are doing is comparing and shifting, which can also be done in place in the same array.

Here is the step by step guide to coding insertion sort algorithm in Java:

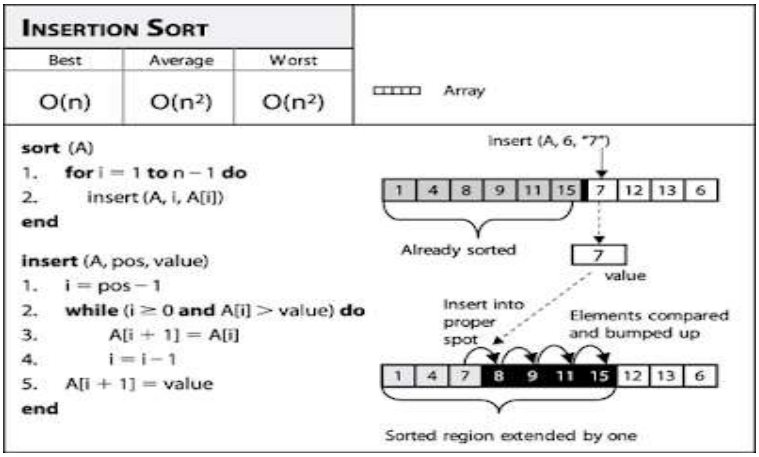
1) Consider the first element is sorted and it's on the proper place, that is index 0 for your array.

2) Now go to the second element (index 1 in the array), and compare it with what is in your hand (the part of the array, which is already sorted). Which means you compare this element going backward towards index zero.

3) If the current number is smaller than the previous number (which is in the proper place), we need to put our current number before that. How will we do that? Well for that we need to shift the existing number.

But what if there is another element which is greater than our current element? It means we need to continue comparing until we found a proper place for our current number, which again means current number> existing number or we are at the start of the list (index 0 in the array).

4) You need to repeat this process for all the numbers in the list. Once you finish that, you have a sorted list or array.



Insertion Sort in Java with Example

It's very easy to implement Insertion sort in Java. All you need to do is to iterate over the array and find proper position of each element, for that you need to shift element and you can do it by swapping. The logic of sorting integer array using insertion sort algorithm is inside method insertionSort(int[]).

In Java you can also sort any object e.g. String using this algorithm, all you need to do is to use Comparable interface because that will provide you mechanism to compare two objects. Now instead of using > (greater than) or < (less than) operator, we need to use compareTo() method.

For this, we have decided to overload our insertionSort() method, where overloaded version takes an Object array instead of an int array. Both methods sort element using insertion sort logic.

By the way, in the real world, you don't need to reinvent the wheel, java.util.Arrays class provides several utility methods to operate upon arrays and one of them is sort.

There is a couple of overloaded version of sort() method available to sort primitive and object arrays. This method uses double pivot QuickSort to sort the primitive array and MergeSort to sort object array.

Anyway, here is our complete code example to run Insertion sort in Java. If you are using Eclipse IDE then just copy paste the code in the src folder of your Java project and Eclipse will create packages and source file with the same name by itself. All you need to is that to run it as Java program.

```
import java.util.Arrays;


/**
 * Java program to sort an array using Insertion sort algorithm.
 * Insertion sort works great with already sorted, small arrays but
 * not suitable for large array with random order.
 *
 * @author ocj4u
 */
public class InsertionSort {

  public static void main(String args[]) {

  // getting unsorted integer array for sorting
```

```
int[] randomOrder = getRandomArray(9);
System.out.println("Random Integer array before Sorting : "
                    + Arrays.toString(randomOrder));

// sorting array using insertion sort in Java
insertionSort(randomOrder);
System.out.println("Sorted array uisng insretion sort : "
                     + Arrays.toString(randomOrder));

// one more example of sorting array using insertion sort
randomOrder = getRandomArray(7);
System.out.println("Before Sorting : " + Arrays.toString(randomOrder));
insertionSort(randomOrder);
System.out.println("After Sorting : " + Arrays.toString(randomOrder));

// Sorting String array using Insertion Sort in Java
String[] cities = {"London", "Paris", "Tokyo", "NewYork", "Chicago"};
System.out.println("String array before sorting : " + Arrays.toString(cities));
insertionSort(cities);
System.out.println("String array after sorting : " + Arrays.toString(cities));
}

public static int[] getRandomArray(int length) {
 int[] numbers = new int[length];
 for (int i = 0; i < length; i++) {
   numbers[i] = (int) (Math.random() * 100);
 }
 return numbers;
}

/*
```

```
 * Java implementation of insertion sort algorithm to sort
 * an integer array.
 */
 public static void insertionSort(int[] array) {
 // insertion sort starts from second element
 for (int i = 1; i < array.length; i++) {
  int numberToInsert = array[i];

  int compareIndex = i;
  while (compareIndex > 0 && array[compareIndex - 1] > numberToInsert) {
    array[compareIndex] = array[compareIndex - 1]; // shifting element
    compareIndex--; // moving backwards, towards index 0
  }

  // compareIndex now denotes proper place for number to be sorted
   array[compareIndex] = numberToInsert;
 }
}

 /*
 * Method to Sort String array using insertion sort in Java.
 * This can also sort any object array which implements
 * Comparable interface.
 */
 public static void insertionSort(Comparable[] objArray) {
 // insertion sort starts from second element
 for (int i = 1; i < objArray.length; i++) {
    Comparable objectToSort = objArray[i];

    int j = i;
    while (j > 0 && objArray[j - 1].compareTo(objectToSort) > 1) {
```

```
        objArray[j] = objArray[j - 1];
        j--;
      }
     objArray[j] = objectToSort;
   }
 }

}
```

Output:

Random Integer array before **Sorting** : [74, 87, 27, 6, 25, 94, 53, 91, 15]

Sorted array uisng insretion sort : [6, 15, 25, 27, 53, 74, 87, 91, 94]

Before Sorting : [71, 5, 60, 19, 4, 78, 42]

After Sorting : [4, 5, 19, 42, 60, 71, 78]

String array before sorting : [**London**, **Paris**, **Tokyo**, **NewYork**, **Chicago**]

String array after sorting : [**Chicago**, **London**, **NewYork**, **Paris**, **Tokyo**]

26. How are duplicates removed from a given array in Java? (solution)

Java Program to remove duplicates from integer array without Collection

In this program, we have not used any collection class to remove duplicates, earlier, I had shown you a way to remove duplicates from ArrayList, which was using LinkedHashSet. You can still use that solution if the interviewer doesn't mention without Collection specifically.

All you need to do is to convert your array into ArrayList first then subsequently create a LinkedHashSet from that ArrayList. In this example, we are removing duplicates from the array by not copying them into result array, this solution is not actually deleting duplicates instead it replacing it with default value i.e. zero.

```
import java.util.Arrays;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

/**

 * Java program to remove duplicates from this array. You don't

 * need to physically delete duplicate elements, replacing with null, or
```

```
 * empty or default value is ok.
 *
 * @author http://javarevisited.blogspot.com
 */
public class TechnicalInterviewTest {

    private static final Logger logger =
LoggerFactory.getLogger(TechnicalInterviewTest.class);

    public static void main(String args[]) {

        int[][] test = new int[][]{
            {1, 1, 2, 2, 3, 4, 5},
            {1, 1, 1, 1, 1, 1, 1},
            {1, 2, 3, 4, 5, 6, 7},
            {1, 2, 1, 1, 1, 1, 1},};

        for (int[] input : test) {
            System.out.println("Array with Duplicates       : " + Arrays.toString(input));
            System.out.println("After removing duplicates   : " +
Arrays.toString(removeDuplicates(input)));
        }
    }

    /*
     * Method to remove duplicates from array in Java, without using
     * Collection classes e.g. Set or ArrayList. Algorithm for this
     * method is simple, it first sort the array and then compare adjacent
```

```
     * objects, leaving out duplicates, which is already in the result.
     */
    public static int[] removeDuplicates(int[] numbersWithDuplicates) {

        // Sorting array to bring duplicates together
        Arrays.sort(numbersWithDuplicates);

        int[] result = new int[numbersWithDuplicates.length];
        int previous = numbersWithDuplicates[0];
        result[0] = previous;

        for (int i = 1; i < numbersWithDuplicates.length; i++) {
            int ch = numbersWithDuplicates[i];

            if (previous != ch) {
                result[i] = ch;
            }
            previous = ch;
        }
        return result;

    }
}
```

Output :

```
Array with Duplicates       : [1, 1, 2, 2, 3, 4, 5]
After removing duplicates   : [1, 0, 2, 0, 3, 4, 5]
Array with Duplicates       : [1, 1, 1, 1, 1, 1, 1]
```

After removing duplicates : [1, 0, 0, 0, 0, 0, 0]

Array with Duplicates : [1, 2, 3, 4, 5, 6, 7]

After removing duplicates : [1, 2, 3, 4, 5, 6, 7]

Array with Duplicates : [1, 2, 1, 1, 1, 1, 1]

After removing duplicates : [1, 0, 0, 0, 0, 0, 2]

That's it about **how to remove duplicates from an array in Java without using Collection class**. As I said before, this solution is not perfect and has some serious limitation, which is an exercise for you to find out. One hint I can give is that array itself can contain default value as duplicates e.g. 0 for int, even if you use any Magic number e.g. Integer.MAX_VALUE, you can not be certain that they will not be part of the input.

Regarding removing duplicate permanently from result array, one approach could be to count a number of duplicates and then create an array of right size i.e. length - duplicates, and then copying content from intermediate result array to final array, leaving out elements which are marked duplicate.

27. how to remove the duplicate character from String? (solution)

Here is my solution for the problem of removing repeated or duplicate characters from given String in Java programming language. If you understand the logic you can write this solution in any programming language e.g. C, C++, C#, Python or JavaScript.

```
/**
 * Java Program to remove duplicate characters from String.
 *
 * @author ocj4u
 */
public class RemoveDuplicateCharacters{

    public static void main(String args[]) {

        System.out.println("Call removeDuplicates(String word) method
....");
        String[] testdata = {"aabscs", "abcd", "aaaa", null, "",
                             "aaabbb", "abababa"};

        for (String input : testdata) {
            System.out.printf("Input : %s  Output: %s %n",
                              input, removeDuplicates(input));
```

```
        }

        System.out.println("Calling removeDuplicatesFromString(String
str).");
        for (String input : testdata) {
            System.out.printf("Input : %s  Output: %s %n",
                  input, removeDuplicatesFromString(input));
        }
    }

    /*
     * This algorithm goes through each character of String to check if
its
     * a duplicate of already found character. It skip the duplicate
     * character by inserting 0, which is later used to filter those
     * characters and update the non-duplicate character.
     * Time Complexity of this solution is O(n^2), excluded to
     * UniqueString() method, which creates String from character array.
     * This method will work even if String contains more than one
duplicate
     * character.
     */
    public static String removeDuplicates(String word) {
        if (word == null || word.length() < 2) {
            return word;
        }

        int pos = 1; // possible position of duplicate character
        char[] characters = word.toCharArray();

        for (int i = 1; i < word.length(); i++) {
            int j;
            for (j = 0; j < pos; ++j) {
                if (characters[i] == characters[j]) {
                    break;
                }
            }
            if (j == pos) {
```

```
                characters[pos] = characters[i];
                ++pos;
            } else {
                characters[pos] = 0;
                ++pos;
            }
        }

        return toUniqueString(characters);
    }


    /*
     * This solution assumes that given input String only contains
     * ASCII characters. You should ask this question to your Interviewer,
     * if he says ASCII then this solution is Ok. This Algorithm
     * uses additional memory of constant size.
     */
    public static String removeDuplicatesFromString(String input) {
        if (input == null || input.length() &lt; 2) {
            return input;
        }

        boolean[] ASCII = new boolean[256];
        char[] characters = input.toCharArray();
        ASCII[input.charAt(0)] = true;

        int dupIndex = 1;
        for (int i = 1; i &lt; input.length(); i++) {
            if (!ASCII[input.charAt(i)]) {
                characters[dupIndex] = characters[i];
                ++dupIndex;
                ASCII[characters[i]] = true;

            } else {
                characters[dupIndex] = 0;
                ++dupIndex;
            }
```

```
        }

        return toUniqueString(characters);
    }

    /*
     * Utility method to convert Character array to String, omitting
     * NUL character, ASCII value 0.
     */
    public static String toUniqueString(char[] letters) {
        StringBuilder sb = new StringBuilder(letters.length);
        for (char c : letters) {
            if (c != 0) {
                sb.append(c);
            }
        }
        return sb.toString();
    }
}


Output
Call removeDuplicates(String word) method ....
Input : aabscs  Output: absc
Input : abcd  Output: abcd
Input : aaaa  Output: a
Input : null  Output: null
Input :   Output:
Input : aaabbb  Output: ab
Input : abababa  Output: ab
Calling removeDuplicatesFromString(String str) method ....
Input : aabscs  Output: absc
Input : abcd  Output: abcd
Input : aaaa  Output: a
Input : null  Output: null
Input :   Output:
Input : aaabbb  Output: ab
Input : abababa  Output: ab
```

That's all about **how to remove duplicate characters from given String in Java**. You can use this logic to remove duplicate values from the array in Java as well.

28. How to find the maximum occurring character in given String? (solution)

Java program to count number of occurrence of any character on String:

```
import org.springframework.util.StringUtils;
/**
 * Java program to count the number of occurrence of any
character on String.
 * @author ocj4u
 */
public class CountCharacters {

    public static void main(String args[]) {

        String input = "Today is Monday"; //count number of
"a" on this String.

        //Using Spring framework StringUtils class for finding
occurrence of another String
        int count =
StringUtils.countOccurrencesOf(input, "a");

        System.out.println("count of occurrence of character
'a' on String: " +
                " Today is Monday' using Spring StringUtils
" + count);


        //Using Apache commons lang StringUtils class
        int number =
org.apache.commons.lang.StringUtils.countMatches(input, "a");
        System.out.println("count of character 'a' on String:
'Today is Monday' using commons StringUtils " + number);

        //counting occurrence of character with loop
        int charCount = 0;
        for(int i =0 ; i<input.length(); i++){
            if(input.charAt(i) == 'a'){
                charCount++;
            }
        }
        System.out.println("count of character 'a' on String:
'Today is Monday' using for loop  " + charCount);
```

```
        //a more elegant way of counting occurrence of
character in String using foreach loop

        charCount = 0; //resetting character count
        for(char ch: input.toCharArray()){
            if(ch == 'a'){
                charCount++;
            }
        }
        System.out.println("count of character 'a' on String:
'Today is Monday' using for each loop  " + charCount);
    }


}
```

Output

```
count of occurrence of character 'a' on String: 'Today is
Monday' using Spring StringUtils 2
count of character 'a' on String: 'Today is Monday' using
commons StringUtils 2
count of character 'a' on String: 'Today is
Monday' using for loop  2
count of character 'a' on String: 'Today is
Monday' using for each loop  2
```

Well, the beauty of this questions is that Interviewer can twist it on many ways, they can ask you to write a recursive function to count occurrences of a particular character or they can even ask to count how many times each character has appeared.

So if a String contains multiple characters and you need to store count of each character, consider using HashMap for storing character as key and number of occurrence as value. Though there are other ways of doing it as well but I like the HashMap way of counting character for simplicity.

29. How is an integer array sorted in place using the quicksort algorithm? (solution)

How QuickSort Algorithm works

Quicksort is a divide and conquer algorithm, which means original list is divided into multiple list, each of them is sorted individually and then sorted output is merged to produce the sorted list. Here is step by step explanation of how quicksort algorithm works.

Steps to implement Quick sort algorithm in place:

1) Choose an element, called pivot, from the list or array. Generally pivot is the middle element of array.

2) Reorder the list so that all elements with values less than the pivot come before the pivot,

and all elements with values greater than the pivot come after it (equal values can go either way). This is also known as *partitioning*. After partitioning the pivot is in its final position.

3) Recursively apply the above steps to the sub-list of elements with smaller values and separately the sub-list of elements with greater values. If the array contains only one element or zero elements then the array is sorted.

Sorting an array of integer using QuickSort sorting algorithm

Java Program to implement QuickSort Algorithm

Here is a Java program to sort an array of integers using QuickSort algorithm. It is an in-place, recursive implementation of QuickSort. Logic is encapsulated in QuickSort class, and method quickSort(int low, int high). This method is called recursively to sort the array. This algorithm work exactly as explained in above GIF image, so if you understand the logic there, its very easy to write by your own.

```
import java.util.Arrays;


/**
 * Test class to sort array of integers using Quicksort algorithm in Java.
 * @author ocj4u
 */
public class QuickSortDemo{

    public static void main(String args[]) {

        // unsorted integer array
        int[] unsorted = {6, 5, 3, 1, 8, 7, 2, 4};
        System.out.println("Unsorted array :" + Arrays.toString(unsorted));

        QuickSort algorithm = new QuickSort();

        // sorting integer array using quicksort algorithm
        algorithm.sort(unsorted);

        // printing sorted array
        System.out.println("Sorted array :" + Arrays.toString(unsorted));

    }

}
```

```java
/**
 * Java Program sort numbers using QuickSort Algorithm. QuickSort is a divide
 * and conquer algorithm, which divides the original list, sort it and then
 * merge it to create sorted output.
 *
 * @author ocj4u
 */
class QuickSort {

    private int input[];
    private int length;

    public void sort(int[] numbers) {

        if (numbers == null || numbers.length == 0) {
            return;
        }
        this.input = numbers;
        length = numbers.length;
        quickSort(0, length - 1);
    }

    /*
     * This method implements in-place quicksort algorithm recursively.
     */
    private void quickSort(int low, int high) {
        int i = low;
        int j = high;

        // pivot is middle index
        int pivot = input[low + (high - low) / 2];

        // Divide into two arrays
        while (i <= j) {
            /**
             * As shown in above image, In each iteration, we will identify a
             * number from left side which is greater then the pivot value, and
```

```
             * a number from right side which is less then the pivot value. Once
             * search is complete, we can swap both numbers.
             */
            while (input[i] < pivot) {
                i++;
            }
            while (input[j] > pivot) {
                j--;
            }
            if (i <= j) {
                swap(i, j);
                // move index to next position on both sides
                i++;
                j--;
            }
        }

        // calls quickSort() method recursively
        if (low < j) {
            quickSort(low, j);
        }

        if (i < high) {
            quickSort(i, high);
        }
    }

    private void swap(int i, int j) {
        int temp = input[i];
        input[i] = input[j];
        input[j] = temp;
    }
}

Output :
Unsorted array :[6, 5, 3, 1, 8, 7, 2, 4]
Sorted array :[1, 2, 3, 4, 5, 6, 7, 8]
```

Import points about Quicksort algorithm

Now we know how quick sort works and how to implement quicksort in Java, its time to revise some of the important points about this popular sorting algorithm.

1) QuickSort is a divide and conquer algorithm. Large list is divided into two and sorted separately (conquered), sorted list is merge later.

2) On "in-place" implementation of quick sort, list is sorted using same array, no additional array is required. Numbers are re-arranged pivot, also known as partitioning.

3) Partitioning happen around pivot, which is usually middle element of array.

4) Average case time complexity of Quicksort is O(n log n) and worst case time complexity is O(n ^2), which makes it one of the fasted sorting algorithm. Interesting thing is it's worst case performance is equal to Bubble Sort :)

5) Quicksort can be implemented with an in-place partitioning algorithm, so the entire sort can be done with only O(log n) additional space used by the stack during the recursion.

6) Quicksort is also a good example of algorithm which makes best use of CPU caches, because of it's divide and conquer nature.

7) In Java, Arrays.sort() method uses quick sort algorithm to sort array of primitives. It's different than our algorithm, and uses two pivots. Good thing is that it perform much better than most of the quicksort algorithm available on internet for different data sets, where traditional quick sort perform poorly. One more reason, not to reinvent the wheel but to use the library method, when it comes to write production code.

That's all about **Quicksort sorting algorithm in Java**. It is one of the must know algorithm for all level of Java programmers, not that you need it often to implement it but to do well on interviews and use the lesson learned while implementing quick sort in Java. In our example, we have implemented quicksort "in-place", which is what you should do if asked to write quicksort in Java. Remember as Java programmer, you don't need to write your own implementation as library implementation are much better implemented and tested. You should use Arrays.sort() method to sort your array instead of writing your own sort method. One more reason of using library method is that they are usually improved over

different version, and can take advantage of new machine instructions or native improvement.

30. How do you reverse a given string in place? (solution)

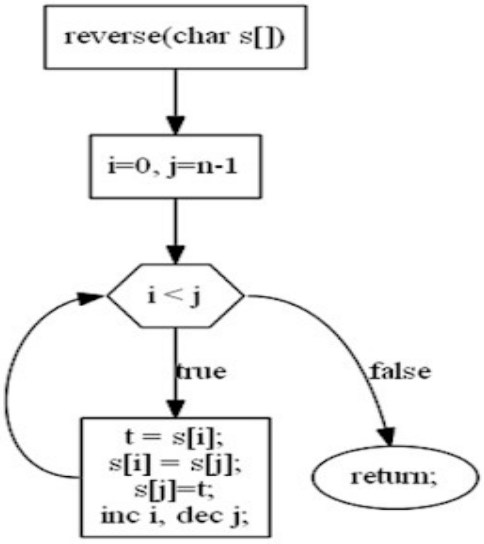
Java Program to Reverse a String in place

Here is the simple example to reverse characters in String by using two pointer technique. This is an in-place algorithm because it doesn't allocate any extra array, it just uses the two int variables to hold positions from start and end.

If you look closely this algorithm is similar to the algorithm we have earlier used to reverse an array in place. That's obvious because String is backed by character array in Java.

If you know how to reverse an array in place then reversing a String is not different for you. What is more important is checking for null and empty String because this is where many programmers become lazy and started writing code without validating input.

Here is the iterative algorithm to reverse String in place:



Java Program to reverse a String in Place

```
import org.junit.Assert;

import org.junit.Test;

/**

 * Java Program to reverse a String in place,

 * without any additional buffer in Java.

 *
```

```
 * @author WINDOWS 8
 *
 */
public class StringReversal {


   /**
    * Java method to reverse a String in place
    * @param str
    * @return  reverse of String
    */
   public static String reverse(String str) {
      if(str == null || str.isEmpty()){
         return str;
      }
      char[] characters = str.toCharArray();
      int i = 0;
      int j = characters.length - 1;
      while (i < j) {
         swap(characters, i, j);
         i++;
         j--;
      }
      return new String(characters);
   }
   /**
    * Java method to swap two numbers in given array
    * @param str
    * @param i
```

```
     * @param j
     */
    private static void swap(char[] str, int i, int j) {
        char temp = str[i];
        str[i] = str[j];
        str[j] = temp;
    }
    @Test
    public void reverseEmptyString(){
        Assert.assertEquals("", reverse(""));
    }
    @Test
    public void reverseString(){
        Assert.assertEquals("cba", reverse("abc"));
    }
   @Test
    public void reverseNullString(){
        Assert.assertEquals(null, reverse(null));
    }
  @Test
    public void reversePalindromeString(){
        Assert.assertEquals("aba", reverse("aba"));
    }
    @Test
 public void reverseSameCharacterString(){
        Assert.assertEquals("aaa", reverse("aaa"));
    }
    @Test
```

```
public void reverseAnagramString(){

      Assert.assertEquals("mary", reverse("yram"));

   }

}
```

You might have also noticed that this time, I have not to use the main() method to test the code, instead I have written couple of JUnit test cases. It's actually better to write unit test cases all the time to test your code instead of using main() method as it put unit testing in your habit.

31. How do you print duplicate characters from a string? (solution)

Java Program to find Repeated Characters of String

The standard way to solve this problem is to get the character array from String, iterate through that and build a Map with character and their count. Then iterate through that Map and print characters which have appeared more than once. So you actually need two loops to do the job, the first loop to build the map and second loop to print characters and counts.

If you look at below example, there is only one static method called printDuplicateCharacters(), which does both this job. We first got the character array from String by calling toCharArray().

Next we are using HashMap to store characters and their count. We use containsKey() method to check if key, which is a character already exists or not, if already exists we get the old count from HashMap by calling get() method and store it back after incrementing it by 1.

Once we build our Map with each character and count, next task is to loop through Map and check each entry, if count, which is the value of Entry is greater than 1, then that character has occurred more than once. You can now print duplicate characters or do whatever you want with them.

```
import java.util.HashMap;

import java.util.Map;

import java.util.Scanner;

import java.util.Set;


/**

* Java Program to find duplicate characters in String.

*/

public class FindDuplicateCharacters{

   public static void main(String args[]) {

      printDuplicateCharacters("Programming");
```

```
        printDuplicateCharacters("Combination");
        printDuplicateCharacters("Java");
    }

    /*
     * Find all duplicate characters in a String and print each of them.
     */
    public static void printDuplicateCharacters(String word) {
        char[] characters = word.toCharArray();

        // build HashMap with character and number of times they appear in String
        Map<Character, Integer> charMap = new HashMap<Character, Integer>();
        for (Character ch : characters) {
            if (charMap.containsKey(ch)) {
                charMap.put(ch, charMap.get(ch) + 1);
            } else {
                charMap.put(ch, 1);
            }
        }

        // Iterate through HashMap to print all duplicate characters of String
        Set<Map.Entry<Character, Integer>> entrySet = charMap.entrySet();
        System.out.printf("List of duplicate characters in String '%s' %n", word);
        for (Map.Entry<Character, Integer> entry : entrySet) {
            if (entry.getValue() > 1) {
                System.out.printf("%s : %d %n", entry.getKey(), entry.getValue());
            }
        }
    }
}
```

Output

List of duplicate characters in String 'Programming'

g : 2

r : 2

m : 2

List of duplicate characters in String 'Combination'

n : 2

o : 2

i : 2

List of duplicate characters in String 'Java'

That's all on how to find duplicate characters in a String.

32. How do you check if two strings are anagrams of each other? (solution)

There are multiple ways to find if two string are anagram or not. Classical way is getting character array of each String, and then comparing them, if both char array is equal then Strings are anagram. But before comparing, make sure that both String are in same case e.g. lowercase or uppercase and character arrays are sorted, because equals method of Arrays, return true, only if array contains same length, and each index has same character.
For simplicity, I have left checking if String is null or empty and converting them into uppercase or lowercase, you can do that if you want. If Interviewer ask you to write production quality code, then I would suggest definitely put those checks and throw `IllegalArgumentException` for null String or you can simply return false. I would personally prefer to return false rather than throwing Exception, similar to equals() method. Anyway, here are *three ways to check if two String are Anagram* or not. I have also included a JUnit Test to verify various String which contains both anagram and not.

```
import java.util.Arrays;

/**
 * Java program - String Anagram Example.
 * This program checks if two Strings are anagrams or not
 *
 * @author ocj4u
 */
public class AnagramCheck {

    /*
     * One way to find if two Strings are anagram in Java. This method
     * assumes both arguments are not null and in lowercase.
     *
     * @return true, if both String are anagram
     */
    public static boolean isAnagram(String word, String anagram){
        if(word.length() != anagram.length()){
            return false;
        }

        char[] chars = word.toCharArray();

        for(char c : chars){
```

```
            int index = anagram.indexOf(c);
            if(index != -1){
                anagram = anagram.substring(0,index) +
anagram.substring(index +1, anagram.length());
            }else{
                return false;
            }
        }

        return anagram.isEmpty();
    }

    /*
     * Another way to check if two Strings are anagram or not in Java
     * This method assumes that both word and anagram are not null and
lowercase
     * @return true, if both Strings are anagram.
     */
    public static boolean iAnagram(String word, String anagram){
        char[] charFromWord = word.toCharArray();
        char[] charFromAnagram = anagram.toCharArray();
        Arrays.sort(charFromWord);
        Arrays.sort(charFromAnagram);

        return Arrays.equals(charFromWord, charFromAnagram);
    }


    public static boolean checkAnagram(String first, String second){
        char[] characters = first.toCharArray();
        StringBuilder sbSecond = new StringBuilder(second);

        for(char ch : characters){
            int index = sbSecond.indexOf("" + ch);
            if(index != -1){
                sbSecond.deleteCharAt(index);
            }else{
                return false;
            }
        }

        return sbSecond.length()==0 ? true : false;
    }
}
```

33. How do you find all the permutations of a string? (solution)

Java Program to Print All Permutation of a String

Here is our sample Java program to print all permutations of given String using recursive algorithm. It uses both loop and recursive call to solve this problem. It also demonstrate a technique of hiding your implementation detail using a private method and exposing a much cleaner public method as API. In our solution, we have two permutation method, one is public and other is private.

First method is clean and exposed to client but second method require you to pass an

empty String as initial value of perm parameter which is used to store intermediate permutation of String.

If you expose this method to client then it will wonder about this empty String, since it's part of implementation, its better to hide and get rid of it as soon as you have a better algorithm to solve this problem, how about taking it as an exercise?

```
/**
  * Java program to find all permutations of a given String using
recursion.
  * For example, given a String "XYZ", this program will print all 6
possible permutations of
  * input e.g. XYZ, XZY, YXZ, YZX, ZXY, XYX
  *
  * @author ocj4u
  */
public class StringPermutations {

    public static void main(String args[]) {
        permutation("123");
    }


 /*
  * A method exposed to client to calculate permutation of String in Java.
  */
   public static void permutation(String input){
          permutation("", input);
   }

   /*
    * Recursive method which actually prints all permutations
    * of given String, but since we are passing an empty String
    * as current permutation to start with,
    * I have made this method private and didn't exposed it to client.
    */
   private static void permutation(String perm, String word) {
        if (word.isEmpty()) {
            System.err.println(perm + word);
```

```
        } else {
            for (int i = 0; i &lt; word.length(); i++) {
                permutation(perm + word.charAt(i), word.substring(0, i)
                                        + word.substring(i + 1,
word.length()));
            }
        }

    }
}

Output:
123
132
213
231
312
321
```

Explanation of Code :

All code for calculating permutation of String is inside **`permutation(String perm, String word)`** method, I have purposefully made this method private because of additional parameter I am passing as an initial value of permutation.

This demonstrates a technique of hiding implementation detail from a client and exposing much cleaner API to client e.g. just **`permutation(String input)`** method, passing empty String is an implementation detail and ugly for a client to pass whenever it needs to calculate permutation. It is also an exercise for you to see if you can improve the code by getting rid of that empty String.

Algorithm is nothing but keeping one character fix and then calculating permutations of others. Crux of program is in following code segment :

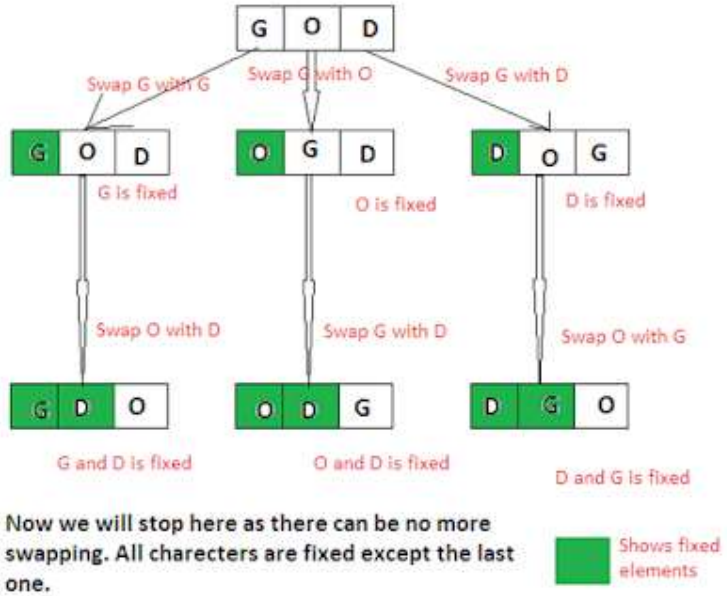
```
 for (int i = 0; i &lt; word.length(); i++) {
   permutation(perm + word.charAt(i), word.substring(0, i)
                    + word.substring(i + 1, word.length()));
}
```

Here we have a for loop to go through each character of String e.g. for input "123" this

loop will run three times. In each iteration, we are making a recursive call to function itself i.e. `permutation(String perm, String word)` method, where the first parameter is used to store the result.

After 1st iteration **perm** (first parameter of permutation() method) will be `"" + 1` as we are doing `word.charAt(i)` and i is zero. Next, we take out that character and pass the remaining characters to permutation method again e.g. `"23"` in the first iteration. Recursive call ends when it reaches to base case i.e. when remaining word becomes empty, at that point "perm" parameter contains a valid permutation to be printed. You can also store it into a List if you want to.

Here is a nice diagram which visually shows what this algorithm does :



That's all on **how to find all permutations of a String in Java using recursion**. It's a very good exercise for preparing Java coding interviews. Why not you give it a try and come up with another solution? also could you calculate complexity of this algorithm, to me it looks **`n*!n`** because loop will run for n times and for each n, we will call permutation method. Also, how about writing some JUnit test cases to see if this solution works for various input e.g. empty String, one letter String, many letters String, String with duplicate characters etc? It's a good practice to become hands-on writing JUnit tests.

34. How can a given string be reversed using recursion? (solution)

if you are from C background and new to Java then you get surprise that java **Strings are not character array** instead they are object and Strings in Java are **not null**

terminated but still you can use your C skill to write *iterative reverse function for string* by getting character array by calling String.toCharArray() and getting length of String by calling String.length(). Now if you succeed in writing **String reverse function**, an iterative version without using StringBuffer reverse than they finally ask you to write a recursive one. Since recursion is a tricky concept and not many Java developer think recursive as compared to C++ dudes, you may see many of Java programmer stuck here by doing doodling around subString, indexOf etc. So its better to prepare in advance. here I have given **example of reversing string using StringBuffer**, *iterative version of String reversal* and **reversing string in Java using recursion**. As I said recursive solution is always tricky to come up you need to found a suitable base case and than repetitive call recursive method with each call reducing length of argument. anyway mine way is just one way of reversing string using recursion, there are many solution out there. so don't forget to try out yourself.

String Reverse Example using Iteration and Recursion in Java

Here is **code example String reverse using iterative and recursive function** written in Java. Recursive solution is just for demonstrative and education purpose, don't use recursive solution in production code as it may result in StackOverFlowError if String to be reversed is very long String or if you have any bug in your reverse function, anyway its good test to make yourself comfortable with recursive functions in java.

```
import java.io.FileNotFoundException;

import java.io.IOException;

/**

*

*@author Ocj4u

/

public class StringReverseExample {


   public static void main(String args[]) throws FileNotFoundException, IOException {


      //original string

      String str = "Sony is going to introduce Internet TV soon";

      System.out.println("Original String: " + str);


      //reversed string using Stringbuffer

      String reverseStr = new StringBuffer(str).reverse().toString();
```

```
    System.out.println("Reverse String in Java using StringBuffer: " + reverseStr);

    //iterative method to reverse String in Java
    reverseStr = reverse(str);
    System.out.println("Reverse String in Java using Iteration: " + reverseStr);

    //recursive method to reverse String in Java
    reverseStr = reverseRecursively(str);
    System.out.println("Reverse String in Java using Recursion: " + reverseStr);

}

public static String reverse(String str) {
    StringBuilder strBuilder = new StringBuilder();
    char[] strChars = str.toCharArray();

    for (int i = strChars.length - 1; i >= 0; i--) {
        strBuilder.append(strChars[i]);
    }

    return strBuilder.toString();
}

public static String reverseRecursively(String str) {

    //base case to handle one char string and empty string
    if (str.length() < 2) {
        return str;
    }

    return reverseRecursively(str.substring(1)) + str.charAt(0);
```

```
    }
}
```

That's all on how to **reverse String in Java using Recursion**, **Iteration** and **without using StringBuffer**. This is one of my favorite interview question and I mostly ask to write recursive solution just to know whether programmer has ability to comprehend recursive problem or not.

35. How do you check if a given string is a palindrome? (solution)

Here is our Java program, which checks if a given String is palindrome or not. Program is simple and here are steps to find palindrome String :

1) Reverse the given String
2) Check if reverse of String is equal to itself, if yes then given String is palindrome.

In our solution, we have a static method isPalindromeString(String text), which accepts a String. It then call reverse(String text) method to reverse this String. This method uses recursion to reverse String. This function first check if given String is null or empty, if yes then it return the same String because they don't require to be reversed.

After this validation, it extract last character of String and pass rest or String using substring() method to this method itself, hence recursive solution. The validation also servers as base case because after every step, String keeps getting reduced and eventually it will become empty, there your function will stop recursion and will use String concatenation to concatenate all character in reverse order. Finally this method returns the reverse of String.

Once call to reverse() returns back, isPalindromeString(String text) uses equals() method to check if reverse of String is equal to original String or not, if yes then it returns true, which also means String is palindrome.

How to check if String is Palindrome in Java using Recursion

```
package test;


/**
 * Java program to show you how to check if a String is palindrome or not.
 * An String is said to be palindrome if it is equal to itself after
reversing.
 * In this program, you will learn how to check if a string is a
palindrome in java using recursion
 * and for loop both.
 *
 * @author Ocj4u
 */
```

```java
public class PalindromeTest {


    public static void main(String args[]) {
        System.out.println("Is aaa palindrom?: " +
isPalindromString("aaa"));
        System.out.println("Is abc palindrom?: " +
isPalindromString("abc"));

        System.out.println("Is bbbb palindrom?: " +
isPalindromString("bbbb"));
        System.out.println("Is defg palindrom?: " +
isPalindromString("defg"));



    }

    /**
     * Java method to check if given String is Palindrome
     * @param text
     * @return true if text is palindrome, otherwise false
     */
    public static boolean isPalindromString(String text){
       String reverse = reverse(text);
       if(text.equals(reverse)){
           return true;
       }

       return false;
    }

    /**
     * Java method to reverse String using recursion
     * @param input
     * @return reversed String of input
     */
    public static String reverse(String input){
        if(input == null || input.isEmpty()){
```

```
            return input;
        }

        return input.charAt(input.length()- 1) +
reverse(input.substring(0, input.length() - 1));
    }

}

Output
Is aaa palindrom?: true
Is abc palindrom?: false
Is bbbb palindrom?: true
Is defg palindrom?: false
```

You can also solve this problem by retrieving character array from String using toCharArray() and using a for loop and StringBuffer. All you need to do is iterate through character array from end to start i.e. from last index to first index and append those character into StringBuffer object.

36. How do you find the length of the longest substring without repeating characters? (solution)

```
// Java program to find the length of the longest substring
// without repeating characters
public class GFG {

    static final int NO_OF_CHARS = 256;

    static int longestUniqueSubsttr(String str)
    {
        int n = str.length();
        int cur_len = 1; // length of current substring
        int max_len = 1; // result
        int prev_index; // previous index
        int i;
        int visited[] = new int[NO_OF_CHARS];

        /* Initialize the visited array as -1, -1 is
         used to indicate that character has not been
         visited yet. */
        for (i = 0; i < NO_OF_CHARS; i++) {
            visited[i] = -1;
        }

        /* Mark first character as visited by storing the
             index of first   character in visited array. */
        visited[str.charAt(0)] = 0;
```

```
        /* Start from the second character. First character is
           already processed (cur_len and max_len are initialized
         as 1, and visited[str[0]] is set */
        for (i = 1; i < n; i++) {
            prev_index = visited[str.charAt(i)];

            /* If the current character is not present in
           the already processed substring or it is not
              part of the current NRCS, then do cur_len++ */
            if (prev_index == -1 || i - cur_len > prev_index)
                cur_len++;

            /* If the current character is present in currently
               considered NRCS, then update NRCS to start from
               the next character of the previous instance. */
            else {
                /* Also, when we are changing the NRCS, we
                   should also check whether length of the
                   previous NRCS was greater than max_len or
                   not.*/
                if (cur_len > max_len)
                    max_len = cur_len;

                cur_len = i - prev_index;
            }

            // update the index of current character
            visited[str.charAt(i)] = i;
        }

        // Compare the length of last NRCS with max_len and
        // update max_len if needed
        if (cur_len > max_len)
            max_len = cur_len;

        return max_len;
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {
        String str = "ABDEFGABEF";
        System.out.println("The input string is " + str);
        int len = longestUniqueSubsttr(str);
        System.out.println("The length of "
                           + "the longest non repeating character is " +
len);
    }
}
```

Output

The input string is ABDEFGABEF

The length of the longest non-repeating character substring is 6

Time Complexity: O(n + d) where n is length of the input string and d is number of characters in input string alphabet. Eg- if string consists of lowercase English characters then value of d is 26

37. Given string str, How do you find the longest palindromic substring in str? (solution)

```
// Java Solution
public class LongestPalinSubstring
{
    // A utility function to print a substring str[low..high]
    static void printSubStr(String str, int low, int high) {
        System.out.println(str.substring(low, high + 1));
    }

    // This function prints the longest palindrome substring
    // of str[].
    // It also returns the length of the longest palindrome
    static int longestPalSubstr(String str) {
        int n = str.length();   // get length of input string

        // table[i][j] will be false if substring str[i..j]
        // is not palindrome.
        // Else table[i][j] will be true
        boolean table[][] = new boolean[n][n];

        // All substrings of length 1 are palindromes
        int maxLength = 1;
        for (int i = 0; i < n; ++i)
            table[i][i] = true;

        // check for sub-string of length 2.
        int start = 0;
        for (int i = 0; i < n - 1; ++i) {
            if (str.charAt(i) == str.charAt(i + 1)) {
                table[i][i + 1] = true;
                start = i;
                maxLength = 2;
            }
        }

        // Check for lengths greater than 2. k is length
        // of substring
        for (int k = 3; k <= n; ++k) {

              // Fix the starting index
            for (int i = 0; i < n - k + 1; ++i)
            {
                // Get the ending index of substring from
                // starting index i and length k
                int j = i + k - 1;

                // checking for sub-string from ith index to
                // jth index iff str.charAt(i+1) to
                // str.charAt(j-1) is a palindrome
                if (table[i + 1][j - 1] && str.charAt(i) ==
                                           str.charAt(j)) {
                    table[i][j] = true;

                    if (k > maxLength) {
```

```
                        start = i;
                        maxLength = k;
                    }
                }
            }
        }
        System.out.print("Longest palindrome substring is; ");
        printSubStr(str, start, start + maxLength - 1);

        return maxLength; // return length of LPS
    }

    // Driver program to test above functions
    public static void main(String[] args) {

        String str = "forgeeksskeegfor";
        System.out.println("Length is: " +
                                longestPalSubstr(str));
    }
}
```

Output:
Longest palindrome substring is: geeksskeeg
Length is: 10
Time complexity: O (n^2)
Auxiliary Space: O (n^2)

38. How do you check if a string contains only digits? (solution)

Here is complete code example in Java programming language to check if a String is an integer number or not. In this Java program we are using regular expression to check if String contains only digits i.e. 0 to 9 or not. If String only contains digit than its number otherwise its not a numeric String. One interesting point to note is that this regular expression only checks for integer number as it not looking for `dot(.)` characters, which means floating point or decimal numbers will fail this test.

```
import java.util.regex.Pattern;
/**
 * Java program to demonstrate use of Regular Expression to check
 * if a String is a 6 digit number or not.
 */
public class RegularExpressionExample {

    public static void main(String args[]) {

        // Regular expression in Java to check if String is number or not
        Pattern pattern = Pattern.compile(".*[^0-9].*");
       //Pattern pattern = Pattern.compile(".*\\D.*");
       String [] inputs = {"123", "-123" , "123.12", "abcd123"};

       for(String input: inputs){
           System.out.println( "does " + input + " is number : "
                               + !pattern.matcher(input).matches());
       }

       // Regular expression in java to check if String is 6 digit number or not
       String [] numbers = {"123", "1234" , "123.12", "abcd123", "123456"};
       Pattern digitPattern = Pattern.compile("\\d{6}");
```

```
        //Pattern digitPattern = Pattern.compile("\\d\\d\\d\\d\\d\\d");


        for(String number: numbers){
            System.out.println( "does " + number + " is 6 digit number : "
                                + digitPattern.matcher(number).matches());
        }
    }

}
```

```
Output:
does 123 is number : true
does -123 is number : false
does 123.12 is number : false
does abcd123 is number : false

does 123 is 6 digit number : false
does 1234 is 6 digit number : false
does 123.12 is 6 digit number : false
does abcd123 is 6 digit number : false
does 123456 is 6 digit number : true
```

That's all on using **Java regular expression to check numbers in String**. As you have seen in this Java Regular Expression example that its pretty easy and fun to do validation using regular expression.

39. How to remove Nth Node from the end of a linked list? (solution)

Approach:

- Take two pointers, **first** will point to the **head** of the linked list and **second** will point to the **Nth** node from the beginning.
- Now keep increment both the pointers by one at the same time until second is pointing to the last node of the linked list.
- After the operations from the previous step, first pointer should be pointing to the **Nth** node from the end by now. So, delete the node first pointer is pointing to.

```
class LinkedList {

    // Head of list
    Node head;

    // Linked list Node
    class Node {
        int data;
        Node next;
        Node(int d)
        {
            data = d;
            next = null;
        }
    }

    // Function to delete the nth node from
    // the end of the given linked list
    void deleteNode(int key)
    {

        // First pointer will point to
        // the head of the linked list
```

```
    Node first = head;

    // Second pointer will poin to the
    // Nth node from the beginning
    Node second = head;
    for (int i = 0; i < key; i++) {

        // If count of nodes in the given
        // linked list is <= N
        if (second.next == null) {

            // If count = N i.e. delete the head node
            if (i == key - 1)
                head = head.next;
            return;
        }
        second = second.next;
    }

    // Increment both the pointers by one until
    // second pointer reaches the end
    while (second.next != null) {
        first = first.next;
        second = second.next;
    }

    // First must be pointing to the
    // Nth node from the end by now
    // So, delete the node first is pointing to
    first.next = first.next.next;
}

// Function to insert a new Node at front of the list
public void push(int new_data)
{
    Node new_node = new Node(new_data);
    new_node.next = head;
    head = new_node;
}

// Function to print the linked list
public void printList()
{
    Node tnode = head;
    while (tnode != null) {
        System.out.print(tnode.data + " ");
        tnode = tnode.next;
    }
}

// Driver code
public static void main(String[] args)
{
    LinkedList llist = new LinkedList();

    llist.push(7);
    llist.push(1);
    llist.push(3);
    llist.push(2);
```

```
        System.out.println("\nCreated Linked list is:");
        llist.printList();

        int N = 1;
        llist.deleteNode(N);

        System.out.println("\nLinked List after Deletion is:");
        llist.printList();
    }
}
```

Output:

```
Created Linked list is:

2 3 1 7

Linked List after Deletion is:

2 3 1
```

40. How to merge two sorted linked list? (solution)

The strategy here uses a temporary dummy node as the start of the result list. The pointer Tail always points to the last node in the result list, so appending new nodes is easy. The dummy node gives tail something to point to initially when the result list is empty. This dummy node is efficient, since it is only temporary, and it is allocated in the stack. The loop proceeds, removing one node from either ‘a’ or ‘b’, and adding it to tail. When we are done, the result is in dummy.next.

```
/* Java program to merge two
   sorted linked lists */
import java.util.*;

/* Link list node */
class Node
{
    int data;
    Node next;
    Node(int d) {data = d;
                 next = null;}
}

class MergeLists
{
Node head;

/* Method to insert a node at
   the end of the linked list */
public void addToTheLast(Node node)
{
    if (head == null)
    {
        head = node;
    }
```

```
    else
    {
        Node temp = head;
        while (temp.next != null)
            temp = temp.next;
        temp.next = node;
    }
}

/* Method to print linked list */
void printList()
{
    Node temp = head;
    while (temp != null)
    {
        System.out.print(temp.data + " ");
        temp = temp.next;
    }
    System.out.println();
}


// Driver Code
public static void main(String args[])
{
    /* Let us create two sorted linked
       lists to test the methods
       Created lists:
           llist1: 5->10->15,
           llist2: 2->3->20
    */
    MergeLists llist1 = new MergeLists();
    MergeLists llist2 = new MergeLists();

    // Node head1 = new Node(5);
    llist1.addToTheLast(new Node(5));
    llist1.addToTheLast(new Node(10));
    llist1.addToTheLast(new Node(15));

    // Node head2 = new Node(2);
    llist2.addToTheLast(new Node(2));
    llist2.addToTheLast(new Node(3));
    llist2.addToTheLast(new Node(20));


    llist1.head = new Gfg().sortedMerge(llist1.head,
                                        llist2.head);
    llist1.printList();

}
}

class Gfg
{
/* Takes two lists sorted in
increasing order, and splices
their nodes together to make
one big sorted list which is
returned. */
```

```
Node sortedMerge(Node headA, Node headB)
{

    /* a dummy first node to
       hang the result on */
    Node dummyNode = new Node(0);

    /* tail points to the
    last result node */
    Node tail = dummyNode;
    while(true)
    {

        /* if either list runs out,
        use the other list */
        if(headA == null)
        {
            tail.next = headB;
            break;
        }
        if(headB == null)
        {
            tail.next = headA;
            break;
        }

        /* Compare the data of the two
        lists whichever lists' data is
        smaller, append it into tail and
        advance the head to the next Node
        */
        if(headA.data <= headB.data)
        {
            tail.next = headA;
            headA = headA.next;
        }
        else
        {
            tail.next = headB;
            headB = headB.next;
        }

        /* Advance the tail */
        tail = tail.next;
    }
    return dummyNode.next;
}
}

// This code is contributed
```

Output :

```
Merged Linked List is:

2 3 5 10 15 20
```

41. How to convert a sorted list to a binary search tree? (solution)

Given a sorted array. Write a function that creates a Balanced Binary Search Tree using array elements.

Examples:

```
Input:  Array {1, 2, 3}

Output: A Balanced BST

     2

   /  \

  1    3


Input: Array {1, 2, 3, 4}

Output: A Balanced BST

      3

    /  \

   2    4

 /

1
```

Following is a simple algorithm where we first find the middle node of list and make it root of the tree to be constructed.

```
1) Get the Middle of the array and make it root.

2) Recursively do same for left half and right half.

      a) Get the middle of left half and make it left child of the root

          created in step 1.

      b) Get the middle of right half and make it right child of the

          root created in step 1.
```

Following is the implementation of the above algorithm. The main code which creates Balanced BST is highlighted.

```
// Java program to print BST in given range

// A binary tree node
class Node {

    int data;
    Node left, right;

    Node(int d) {
        data = d;
```

```
        left = right = null;
    }
}

class BinaryTree {

    static Node root;

    /* A function that constructs Balanced Binary Search Tree
     from a sorted array */
    Node sortedArrayToBST(int arr[], int start, int end) {

        /* Base Case */
        if (start > end) {
            return null;
        }

        /* Get the middle element and make it root */
        int mid = (start + end) / 2;
        Node node = new Node(arr[mid]);

        /* Recursively construct the left subtree and make it
         left child of root */
        node.left = sortedArrayToBST(arr, start, mid - 1);

        /* Recursively construct the right subtree and make it
         right child of root */
        node.right = sortedArrayToBST(arr, mid + 1, end);

        return node;
    }

    /* A utility function to print preorder traversal of BST */
    void preOrder(Node node) {
        if (node == null) {
            return;
        }
        System.out.print(node.data + " ");
        preOrder(node.left);
        preOrder(node.right);
    }

    public static void main(String[] args) {
        BinaryTree tree = new BinaryTree();
        int arr[] = new int[]{1, 2, 3, 4, 5, 6, 7};
        int n = arr.length;
        root = tree.sortedArrayToBST(arr, 0, n - 1);
        System.out.println("Preorder traversal of constructed BST");
        tree.preOrder(root);
    }
}
```

Output:

```
Preorder traversal of constructed BST

4 2 1 3 6 5 7
```

Time Complexity: O(n)
Following is the recurrance relation for sortedArrayToBST().

```
T(n) = 2T(n/2) + C
T(n) -->  Time taken for an array of size n
 C   -->  Constant (Finding middle of array and linking root to left
                    and right subtrees take constant time)
```

42. How do you find duplicate characters in a given string? (solution)

Java Program to find Repeated Characters of String

The standard way to solve this problem is to get the character array from String, iterate through that and build a Map with character and their count. Then iterate through that Map and print characters which have appeared more than once. So you actually need two loops to do the job, the first loop to build the map and second loop to print characters and counts.

If you look at below example, there is only one static method called printDuplicateCharacters(), which does both this job. We first got the character array from String by calling toCharArray().

Next we are using HashMap to store characters and their count. We use containsKey() method to check if key, which is a character already exists or not, if already exists we get the old count from HashMap by calling get() method and store it back after incrementing it by 1.

Once we build our Map with each character and count, next task is to loop through Map and check each entry, if count, which is the value of Entry is greater than 1, then that character has occurred more than once. You can now print duplicate characters or do whatever you want with them.

```
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;
import java.util.Set;

/**
* Java Program to find duplicate characters in String.
*
*
* @author http://java67.blogspot.com
*/
public class FindDuplicateCharacters{

    public static void main(String args[]) {
        printDuplicateCharacters("Programming");
        printDuplicateCharacters("Combination");
        printDuplicateCharacters("Java");
    }
```

```
    /*
     * Find all duplicate characters in a String and print
each of them.
     */
    public static void printDuplicateCharacters(String word) {
        char[] characters = word.toCharArray();

        // build HashMap with character and number of times
they appear in String
        Map<Character, Integer> charMap = new
HashMap<Character, Integer>();
        for (Character ch : characters) {
            if (charMap.containsKey(ch)) {
                charMap.put(ch, charMap.get(ch) + 1);
            } else {
                charMap.put(ch, 1);
            }
        }

        // Iterate through HashMap to print all duplicate
characters of String
        Set<Map.Entry<Character, Integer>> entrySet =
charMap.entrySet();
        System.out.printf("List of duplicate characters in
String '%s' %n", word);
        for (Map.Entry<Character, Integer> entry : entrySet) {
            if (entry.getValue() > 1) {
                System.out.printf("%s : %d %n",
entry.getKey(), entry.getValue());
            }
        }
    }

}

Output
List of duplicate characters in String 'Programming'
g : 2
r : 2
m : 2
List of duplicate characters in String 'Combination'
n : 2
o : 2
i : 2
List of duplicate characters in String 'Java'
```

That's all on how to find duplicate characters in a String. Next time if this question is asked to you in a programming job interview, you can confidently write a solution and can explain them. Remember this question is also asked as write a Java program to find repeated characters of a given String, so don't get confused yourself in wording, the algorithm will remain same.

43. How do you count a number of vowels and consonants in a given string? (solution)

Java Program to count vowels and consonants in String

```
import java.util.Scanner;

/**
  * Java Program to count vowels in a String. It accept a
String from command prompt
  * and count how many vowels it contains. To revise, 5
letters a, e, i, o and u are
  * known as vowels in English.
  */
public class VowelCounter {

     public static void main(String args[]) {
          System.out.println("Please enter some text");
          Scanner reader = new Scanner(System.in);

      String input = reader.nextLine();
          char[] letters = input.toCharArray();

      int count = 0;

      for (char c : letters) {
           switch (c) {
              case 'a':
                  case 'e':
                  case 'i':
                  case 'o':
                  case 'u':
                      count++;
          break;
          default:
                  // no count increment
          }

       }
            System.out.println("Number of vowels in String ["
+ input + "] is : " + count);
     }

}

Output:
Please enter some text
```

```
How many vowels in this String
Number of vowels in String [How many vowels in this String] is : 7
```

You can see that above String contains 7 vowel characters, highlighted by red font. This method is pretty quick as we are only accessing array and the using switch to compare it to another character. If you notice switch statement that you will see that we are using fall-through approach, means there is no break for all five cases, as you don't need to put break after every case. it will only increase count at one time, because increment is done at last case statement.

44. How do you reverse words in a given sentence without using any library method? (solution)

Reversing order of words in a Sentence in Java - Solution

Here is our Java solution of this problem. It's simple and straight forward. In this code example, I have shown two ways to reverse words in a String, first one is using, Java's regular expression support to split the string on spaces and then using reverse() method of Collections utility class. Once you split the String using regex "\\s", it will return you an array of words. It will also handle words separated by multiple spaces, so you don't need to worry.

Once you got the array, you can create an ArrayList from array and then you are eligible to use Collections.reverse() method. This will reverse your ArrayList and you will have all the words in reverse order, now all you need to do is concatenate multiple String by iterating over ArrayList.

I have used StringBuilder for concatenating String here. Also make sure to specify size, because resizing of StringBuilder is costly as it involves creation of new array and copying content from old to new array.

Second method is even more easier, instead of using Collections.reverse() method, I have just used traditional for loop and started looping over array from end and performing String concatenation. This way, you even don't need to convert your String array to ArrayList of String. This solution is more memory efficient and faster than previous one.

```
import java.util.Arrays;
import java.util.Collections;
import java.util.List;

/**
 * Java Program to reverse words in String. There are multiple way to
solve this
 * problem. you can either use any collection class e.g. List and reverse
the
 * List and later create reverse String by joining individual words.
```

```
 *
 * @author ocj4u
 */
public class Testing {

  public static void main(String args[]) {

  }

  /*
  * Method to reverse words in String in Java
  */
  public static String reverseWords(String sentence) {
  List< String> words = Arrays.asList(sentence.split("\\s"));
  Collections.reverse(words);
  StringBuilder sb = new StringBuilder(sentence.length());

  for (int i = words.size() - 1; i >= 0; i--) {
  sb.append(words.get(i));
  sb.append(' ');
  }

  return sb.toString().trim();
  }

  public static String reverseString(String line) {
  if (line.trim().isEmpty()) {
  return line;
  }

  StringBuilder reverse = new StringBuilder();
  String[] sa = line.trim().split("\\s");

  for (int i = sa.length - 1; i >= 0; i--) {
  reverse.append(sa[i]);
  reverse.append(' ');
  }
```

```
  return reverse.toString().trim();
  }
}
}
```

Sometime Interviewer may ask you to solve this problem without using Java Collection framework, because it obviously makes the task a lot easier. So its also good to prepare a solution based upon pure programming logic. If you know how to reverse array in Java, then you have an advantage because String is nothing but a character array, but tricky part is you don't need to reverse array but to reverse words.

45. How do you check if two strings are a rotation of each other? (solution)

Here is the exact algorithm to check if One String is a rotation of another:

1) check length of two strings, if length is not same then return false
2) concatenate given string to itself
3) check if rotated version of String exists in this concatenated string
4) if yes, then second String is rotated version of first string

```
import java.util.Scanner;

/*
 * Java Program to check if one String is rotation of
 * another.
 */
public class Main {

  public static void main(String[] args) throws Exception {

    Scanner scnr = new Scanner(System.in);
    System.out.println("Please enter original String");
    String input = scnr.nextLine();

    System.out.println("Please enter rotation of String");
    String rotation = scnr.nextLine();

    if (checkRotatation(input, rotation)) {
      System.out.println(input + " and " + rotation
          + " are rotation of each other");
    } else {
```

```
      System.out.println("Sorry, they are not rotation of another");
    }

    scnr.close();
  }

  /**
   * This method check is given strings are rotation of each other
   * @param original
   * @param rotation
   * @return true or false
   */
  public static boolean checkRotatation(String original, String rotation)
{
    if (original.length() != rotation.length()) {
      return false;
    }

    String concatenated = original + original;

    if (concatenated.indexOf(rotation) != -1) {
      return true;
    }

    return false;
  }
}
```

```
Output
Please enter original String
IndiaVsAustralia
Please enter rotation of String
AustraliaVsIndia
Sorry, they are not rotation of another

Please enter original String
IndiaVsEngland
```

```
Please enter rotation of String
EnglandIndiaVs
IndiaVsEngland and EnglandIndiaVs are rotation of each other
```

46. How to convert a byte array to String? (solution)

Let's look at a simple program showing how to convert byte array to String in Java.

```
package com.abc.util;

public class ByteArrayToString {


	public static void main(String[] args) {

		byte[] byteArray = { 'P', 'A', 'N', 'K', 'A', 'J' };

		byte[] byteArray1 = { 80, 65, 78, 75, 65, 74 };


		String str = new String(byteArray);

		String str1 = new String(byteArray1);


		System.out.println(str);

		System.out.println(str1);

	}

}
```

47. How do you remove a given character from String? (solution)

String Replace Example in Java

In this Java tutorial, we will see How to replace characters and substring from String in Java. First example in this program replaces a character, i.e. it replaces "J" with "K", which creates "Kava" from "Java". Second String replace example, replaces words from String, it replaces Scala with Java.

Third and fourth example shows how to use regular expression to replace String in Java. Both examples uses \\s to match spaces or any white space character and replace with #.

Third one uses replaceFirst() which only replace first match, while fourth one uses replaceAll() which replaces all matches.

```
package test;

/**
 * Java program to replace String in Java using regular
expression.
 * This examples examples how to replace character and
substring from String in Java.
 *
 * @author ocj4u
 */
public class StringReplace {

    public static void main(String args[]) {

     String word = "Java";

     //replacing character in this String
     String replaced = word.replace("J", "K");
     System.out.println("Replacing character in String");
     System.out.println("Original String before replace : " +
word);
     System.out.println("Replaced String : " + replaced);

     //replacing substring on String in Java
     String str = "Scala is good programming language";
     replaced = str.replaceAll("Scala", "Java");
     System.out.println("String before replace : " + str);
     System.out.println("String after replace : " + replaced);

     //replacing all space in String with # using regular
expression
     replaced = str.replaceFirst("\\s", "#");
     System.out.println("Replacing first match of regex using
replaceFirst()");
     System.out.println("Original String before replacement :
" + str);
     System.out.println("Final String : " + replaced);

     System.out.println("Replacing all occurrence of substring
which match regex");
     replaced = str.replaceAll("\\s", "#");
     System.out.println("ReplaceAll Example : " + replaced);
    }
```

```
}

Output:
Replacing character in String
Original String before replace : Java
Replaced String : Kava
String before replace : Scala is good programming language
String after replace : Java is good programming language
Replacing first match of regex using replaceFirst()
Original String before replacement : Scala is good programming
language
Final String : Scala#is good programming language
Replacing all occurrence of substring which match regex
ReplaceAll Example : Scala#is#good#programming#language
```

That's it on How to replace String in Java. As explained java.lang.String class provides multiple overloaded methods, which can replace single character or substring in Java. replaceAll() in particular is very powerful, and can replace all occurrence of any matching character or regular expression in Java. It also expect a regular expression pattern, which provides it more power. You can use this method to say replace all comma with pipe to convert a comma separated file to a pile delimited String. If you just want to replace one character, just use replace() method, which takes two character, the old and new characters

48. How do you find the middle element of a singly linked list in one pass? (solution)

Java Program to Find the Middle Node of a Linked list in a Single-pass

```
import test.LinkedList.Node;

/**
 * Java program to find middle element of linked list in one pass.
 * In order to find middle element of a linked list
 * we need to find the length first but since we can only
 * traverse linked list one time, we will have to use two pointers
 * one which we will increment on each iteration while
 * other which will be incremented every second iteration.
 * So when the first pointer will point to the end of a
 * linked list, second will be pointing to the middle
 * element of a linked list
 *
 * @author ocj4u
 */
public class LinkedListTest {


    public static void main(String args[]) {
        //creating LinkedList with 5 elements including head
      LinkedList linkedList = new LinkedList();
```

```
        LinkedList.Node head = linkedList.head();
        linkedList.add( new LinkedList.Node("1"));
        linkedList.add( new LinkedList.Node("2"));
        linkedList.add( new LinkedList.Node("3"));
        linkedList.add( new LinkedList.Node("4"));

        //finding middle element of LinkedList in single pass
        LinkedList.Node current = head;
        int length = 0;
        LinkedList.Node middle = head;

        while(current.next() != null){
            length++;
            if(length%2 ==0){
                middle = middle.next();
            }
            current = current.next();
        }

        if(length%2 == 1){
            middle = middle.next();
        }

        System.out.println("length of LinkedList: " + length);
        System.out.println("middle element of LinkedList :
"                                   + middle);

    }

}

class LinkedList{
    private Node head;
    private Node tail;

    public LinkedList(){
        this.head = new Node("head");
        tail = head;
    }

    public Node head(){
        return head;
    }

    public void add(Node node){
        tail.next = node;
        tail = node;
    }

    public static class Node{
        private Node next;
```

```
        private String data;

        public Node(String data){
            this.data = data;
        }

        public String data() {
            return data;
        }

        public void setData(String data) {
            this.data = data;
        }

        public Node next() {
            return next;
        }

        public void setNext(Node next) {
            this.next = next;
        }

        public String toString(){
            return this.data;
        }
    }
}

Output:
length of LinkedList: 4
middle element of LinkedList: 2
```

That's all on **How to find middle element of LinkedList in one pass.** As I said this is a good interview question to separate programmers from non-programmers. Also, the technique mentioned here to find middle node of LinkedList can be used to find the 3rd element from Last or nth element from last in a LinkedList as well.

49. How do you check if a given linked list contains a cycle? How do you find the starting node of the cycle? (solution)

Java program to check if linked list is circular or not.

This Java program uses `LinkedList`(not `java.util.LinkedList`) and `Node` class from previous example of Linked List, with modification of adding toString() method and `appendToTail()` method.
Also, `isCyclic()` method of linked list is used to implement logic to find if linked list contains cycle or not. Subsequently `isCyclic()` returns `true` if linked list is cyclic otherwise it return `false`.

```
/*
 * Java class to represent linked list data structure.
 */
```

```
public class LinkedList {
    private Node head;
    public LinkedList() { this.head = new Node("head"); }
    public Node head() { return head;}

    public void appendIntoTail(Node node) {
        Node current = head;

        //find last element of LinkedList i.e. tail
        while(current.next() != null){
            current = current.next();
        }
        //appending new node to tail in LinkedList
        current.setNext(node);
    }

    /*
     * If singly LinkedList contains Cycle then following would be true
     * 1) slow and fast will point to same node i.e. they meet
     * On the other hand if fast will point to null or next node of
     * fast will point to null then LinkedList does not contains cycle.
     */
    public boolean isCyclic(){
        Node fast = head;
        Node slow = head;

        while(fast!= null && fast.next != null){
            fast = fast.next.next;
            slow = slow.next;

            //if fast and slow pointers are meeting then LinkedList is
cyclic
            if(fast == slow ){
                return true;
            }
        }
        return false;
    }

    @Override
    public String toString(){
        StringBuilder sb = new StringBuilder();
        Node current = head.next();
        while(current != null){
           sb.append(current).append("-->");
           current = current.next();
        }
        sb.delete(sb.length() - 3, sb.length()); // to remove --> from last
node
       return sb.toString();
    }

    public static class Node {
        private Node next;
        private String data;

        public Node(String data) {
            this.data = data;
        }

        public String data() { return data; }
```

```
        public void setData(String data) { this.data = data;}

        public Node next() { return next; }
        public void setNext(Node next) { this.next = next; }

        @Override
        public String toString() {
            return this.data;
        }
    }
}
```

50. How do you reverse a linked list? (solution)

Here is our sample program to demonstrate how to reverse a linked list in Java. In order to reverse, I have first created a class called SinglyLinkedList, which represents a linked list data structure. I have further implemented add() and print() method to add elements to the linked list and print them in forwarding order.

The logic of reversing the linked list is encapsulated inside the reverse() method. It traverses through the linked list from head to tail and reverses the link in each step like each node instead of pointing to next element started pointing to the previous node, this way the whole linked list is reversed when you reach the last element, which then becomes the new head of a linked list.

```
package test;


/**
 * Java Program to reverse a singly list without using recursion.
 */
public class LinkedListProblem {

  public static void main(String[] args) {

    // creating a singly linked list
    SinglyLinkedList.Node head = new SinglyLinkedList.Node(1);
    SinglyLinkedList linkedlist = new SinglyLinkedList(head);

    // adding node into singly linked list
    linkedlist.add(new SinglyLinkedList.Node(2));
    linkedlist.add(new SinglyLinkedList.Node(3));
    // printing a singly linked list
    linkedlist.print();

    // reversing the singly linked list
    linkedlist.reverse();
```

```
    // printing the singly linked list again
    linkedlist.print();

  }

}
/**
 * A class to represent singly list in Java
 *
 * @author WINDOWS 8
 *
 */
class SinglyLinkedList {

  static class Node {

    private int data;
    private Node next;

    public Node(int data) {
      this.data = data;
    }

    public int data() {
      return data;
    }

    public Node next() {
      return next;
    }
  }

  private Node head;

  public SinglyLinkedList(Node head) {
    this.head = head;
  }
```

```
/**
 * Java method to add an element to linked list
 * @param node
 */
public void add(Node node) {
  Node current = head;
  while (current != null) {
    if (current.next == null) {
      current.next = node;
      break;
    }
    current = current.next;
  }
}

/**
 * Java method to print a singly linked list
 */
public void print() {
  Node node = head;
  while (node != null) {
    System.out.print(node.data() + " ");
    node = node.next();
  }
  System.out.println("");
}

/**
 * Java method to reverse a linked list without recursion
 */
public void reverse() {
  Node pointer = head;
  Node previous = null, current = null;

  while (pointer != null) {
    current = pointer;
    pointer = pointer.next;
```

```
      // reverse the link
      current.next = previous;
      previous = current;
      head = current;
    }

  }
}
```

Output

```
1 2 3
3 2 1
```

51. How do you reverse a singly linked list without recursion? (solution)

```
/**
  * Java Class to represent singly linked list for demonstration purpose.
  * In order to understand How to reverse linked list, focus on two
methods
  * reverseIteratively() and reverseRecursively().

  * @author Ocj4u Paul
  */
public class SinglyLinkedList {
    private Node head;  // Head is the first node in linked list

    public void append(T data){
        if(head == null){
            head = new Node(data);
            return;
        }
        tail().next = new Node(data);
    }

    private Node tail() {
        Node tail = head;

        // Find last element of linked list known as tail
        while(tail.next != null){
            tail = tail.next;
```

```
        }
        return tail;

    }


    @Override
    public String toString(){
        StringBuilder sb = new StringBuilder();
        Node current = head;
        while(current != null){
           sb.append(current).append("-->");
           current = current.next;
        }
        if(sb.length() &gt;=3){
            sb.delete(sb.length() - 3, sb.length());
            // to remove --> from last node
        }

        return sb.toString();
    }

    /**
      * Reverse linked list using 3 pointers approach in O(n) time
      * It basically creates a new list by reversing direction, and
      * subsequently insert the element at the start of the list.
      */
    public void reverseIteratively() {
        Node current = head;
        Node previous = null;
        Node forward = null;

        // traversing linked list until there is no more element
        while(current.next != null){

            // Saving reference of next node, since we are changing
current node
            forward = current.next;
```

```
            // Inserting node at start of new list
            current.next = previous;
            previous = current;

            // Advancing to next node
            current = forward;
        }

        head = current;
        head.next = previous;
    }

    /*
     * Reverse a singly linked list using recursion. In recursion Stack is
     * used to store data.
     * 1. Traverse linked list till we find the tail,
     * that would be new head for reversed linked list.
     */
    private Node reverseRecursively(Node node){
       Node newHead;

       //base case - tail of original linked list
       if((node.next == null)){
           return node;
       }
       newHead = reverseRecursively(node.next);

       //reverse the link e.g. C->D->null will be null
       node.next.next = node;
       node.next = null;
       return newHead;
    }

    public void reverseRecursively(){
        head = reverseRecursively(head);
    }
```

```
   private static class Node {
        private Node next;
        private T data;

        public Node(T data) {
            this.data = data;
        }

        @Override
        public String toString() {
            return data.toString();
        }
    }

}
```

Test Class

Here is our test class, which will test both methods of reversing a linked list, reverseIteratively() and reverseRecursively(). You have first created a singly linked list with 6 nodes A-B-C-D-E-F, and first reversed them iteratively using 3 points approach and later reversed the same list recursively.

Since the same instance of the singly linked list is reversed two times, you can see in the output that the final list is the same as the original linked list.

```
/**
  * Java program to test code of reversing singly linked list in Java.
  * This test class test both iterative and recursive solution. Since
  * the same list is first reversed using loops, and then again using
recursion.
  * You can see that final output is same as original linked list.

  * @author Ocj4u Paul
  */
public class SinglyLinkedListTest {

    public static void main(String args[]) {
       SinglyLinkedList linkedlist = getDefaultList();
       System.out.println("linked list before reversing : " + linkedlist);
       linkedlist.reverseIteratively();
```

```
        System.out.println("linked list after reversing : " + linkedlist);
        linkedlist.reverseRecursively();
        System.out.println("linked list after reversing recursively: "
                                    + linkedlist);

    }

     private static SinglyLinkedList getDefaultList(){
        SinglyLinkedList linkedlist = new SinglyLinkedList();
        linkedlist.append("A"); linkedlist.append("B");
linkedlist.append("C");
        linkedlist.append("D"); linkedlist.append("E");
linkedlist.append("F");
        return linkedlist;
    }

}

Output:
linked list before reversing : A-->B-->C-->D-->E-->F
linked list after reversing : F-->E-->D-->C-->B-->A
linked list after reversing recursively: A-->B-->C-->D-->E-->F
```

52. How are duplicate nodes removed in an unsorted linked list? (solution)

This is the simple way where two loops are used. Outer loop is used to pick the elements one by one and inner loop compares the picked element with rest of the elements.

```
// Java program to remove duplicates from unsorted
// linked list

class LinkedList {

    static Node head;

    static class Node {

        int data;
        Node next;

        Node(int d) {
            data = d;
            next = null;
        }
```

```
    }

    /* Function to remove duplicates from an
       unsorted linked list */
    void remove_duplicates() {
        Node ptr1 = null, ptr2 = null, dup = null;
        ptr1 = head;

        /* Pick elements one by one */
        while (ptr1 != null && ptr1.next != null) {
            ptr2 = ptr1;

            /* Compare the picked element with rest
                of the elements */
            while (ptr2.next != null) {

                /* If duplicate then delete it */
                if (ptr1.data == ptr2.next.data) {

                    /* sequence of steps is important here */
                    dup = ptr2.next;
                    ptr2.next = ptr2.next.next;
                    System.gc();
                } else /* This is tricky */ {
                    ptr2 = ptr2.next;
                }
            }
            ptr1 = ptr1.next;
        }
    }

    void printList(Node node) {
        while (node != null) {
            System.out.print(node.data + " ");
            node = node.next;
        }
    }

    public static void main(String[] args) {
        LinkedList list = new LinkedList();
        list.head = new Node(10);
        list.head.next = new Node(12);
        list.head.next.next = new Node(11);
        list.head.next.next.next = new Node(11);
        list.head.next.next.next.next = new Node(12);
        list.head.next.next.next.next.next = new Node(11);
        list.head.next.next.next.next.next.next = new Node(10);

        System.out.println("Linked List before removing duplicates : \n ");
        list.printList(head);
        list.remove_duplicates();
        System.out.println("");
        System.out.println("Linked List after removing duplicates : \n ");
        list.printList(head);
}
}
```

Output :

```
Linked list before removing duplicates:
```

```
 10 12 11 11 12 11 10
Linked list after removing duplicates:
 10 12 11
```

53. How do you find the length of a singly linked list? (solution)

Iterative Solution

```
1) Initialize count as 0
2) Initialize a node pointer, current = head.
3) Do following while current is not NULL
     a) current = current -> next
     b) count++;
4) Return count
```

```
// Java program to count number of nodes in a linked list

/* Linked list Node*/
class Node
{
    int data;
    Node next;
    Node(int d)  { data = d;  next = null; }
}

// Linked List class
class LinkedList
{
    Node head;  // head of list

    /* Inserts a new Node at front of the list. */
    public void push(int new_data)
    {
        /* 1 & 2: Allocate the Node &
                  Put in the data*/
        Node new_node = new Node(new_data);

        /* 3. Make next of new Node as head */
        new_node.next = head;

        /* 4. Move the head to point to new Node */
        head = new_node;
    }

    /* Returns count of nodes in linked list */
    public int getCount()
    {
        Node temp = head;
        int count = 0;
        while (temp != null)
```

```
        {
            count++;
            temp = temp.next;
        }
        return count;
    }

    /* Driver program to test above functions. Ideally
       this function should be in a separate user class.
       It is kept here to keep code compact */
    public static void main(String[] args)
    {
        /* Start with the empty list */
        LinkedList llist = new LinkedList();
        llist.push(1);
        llist.push(3);
        llist.push(1);
        llist.push(2);
        llist.push(1);

        System.out.println("Count of nodes is " +
                           llist.getCount());
    }
}
```

Output:

```
count of nodes is 5
```

Recursive Solution

```
int getCount(head)
1) If head is NULL, return 0.
2) Else return 1 + getCount(head->next)
```

54. How do you find the third node from the end in a singly linked list? (solution)

The Kth Node from the End in a Singly linked list

```
public class Practice {

  public static void main(String args[]) {
    SinglyLinkedList list = new SinglyLinkedList();
    list.append("1");
    list.append("2");
    list.append("3");
    list.append("4");
```

```
    System.out.println("linked list : " + list);

    System.out.println("The first node from last: " +
list.getLastNode(1));
    System.out.println("The second node from the end: " +
list.getLastNode(2));
    System.out.println("The third node from the tail: " +
list.getLastNode(3));
  }
}

/**
 * Java Program to implement linked list data structure
 *
 * @author Ocj4u
 *
 */
class SinglyLinkedList {
  static class Node {
    private Node next;
    private String data;

    public Node(String data) {
      this.data = data;
    }

    @Override
    public String toString() {
      return data.toString();
    }
  }

  private Node head; // Head is the first node in linked list

  /**
   * checks if linked list is empty
   *
```

```
 * @return true if linked list is empty i.e. no node
 */
public boolean isEmpty() {
  return length() == 0;
}

/**
 * appends a node at the tail of this linked list
 *
 * @param data
 */
public void append(String data) {
  if (head == null) {
    head = new Node(data);
    return;
  }
  tail().next = new Node(data);
}

/**
 * returns the last node or tail of this linked list
 *
 * @return last node
 */
private Node tail() {
  Node tail = head;
  // Find last element of linked list known as tail
  while (tail.next != null) {
    tail = tail.next;
  }
  return tail;
}

/**
 * method to get the length of linked list
 *
 * @return length i.e. number of nodes in linked list
 */
```

```java
public int length() {
  int length = 0;
  Node current = head;

  while (current != null) {
    length++;
    current = current.next;
  }
  return length;
}

/**
 * to get the nth node from end
 *
 * @param n
 * @return nth node from last
 */
public String getLastNode(int n) {
  Node fast = head;
  Node slow = head;
  int start = 1;

  while (fast.next != null) {
    fast = fast.next;
    start++;

    if (start > n) {
      slow = slow.next;
    }
  }

  return slow.data;
}

@Override
public String toString() {
  StringBuilder sb = new StringBuilder();
```

```
    Node current = head;
    while (current != null) {
      sb.append(current).append("-->");
      current = current.next;
    }

    if (sb.length() >= 3) {
      sb.delete(sb.length() - 3, sb.length());

    }
    return sb.toString();
  }

}
```

```
Output
linked list : 1-->2-->3-->4
the first node from last: 4
the second node from the end: 3
the third node from the tail: 2
```

55. How do you find the sum of two linked lists using Stack? (solution)

Following are the steps.
1) Calculate sizes of given two linked lists.
2) If sizes are same, then calculate sum using recursion. Hold all nodes in recursion call stack till the rightmost node, calculate the sum of rightmost nodes and forward carry to the left side.
3) If size is not same, then follow below steps:
....**a)** Calculate difference of sizes of two linked lists. Let the difference be *diff*
....**b)** Move *diff* nodes ahead in the bigger linked list. Now use step 2 to calculate the sum of the smaller list and right sub-list (of the same size) of a larger list. Also, store the carry of this sum.
....**c)** Calculate the sum of the carry (calculated in the previous step) with the remaining left sub-list of a larger list. Nodes of this sum are added at the beginning of the sum list obtained the previous step.

```
// Java program to add two linked lists

public class linkedlistATN
{
    class node
    {
        int val;
        node next;
```

```
    public node(int val)
    {
        this.val = val;
    }
}

// Function to print linked list
void printlist(node head)
{
    while (head != null)
    {
        System.out.print(head.val + " ");
        head = head.next;
    }
}

node head1, head2, result;
int carry;

/* A utility function to push a value to linked list */
void push(int val, int list)
{
    node newnode = new node(val);
    if (list == 1)
    {
        newnode.next = head1;
        head1 = newnode;
    }
    else if (list == 2)
    {
        newnode.next = head2;
        head2 = newnode;
    }
    else
    {
        newnode.next = result;
        result = newnode;
    }

}

// Adds two linked lists of same size represented by
// head1 and head2 and returns head of the resultant
// linked list. Carry is propagated while returning
// from the recursion
void addsamesize(node n, node m)
{
    // Since the function assumes linked lists are of
    // same size, check any of the two head pointers
    if (n == null)
        return;

    // Recursively add remaining nodes and get the carry
    addsamesize(n.next, m.next);

    // add digits of current nodes and propagated carry
    int sum = n.val + m.val + carry;
    carry = sum / 10;
```

```
    sum = sum % 10;

    // Push this to result list
    push(sum, 3);

}

node cur;

// This function is called after the smaller list is
// added to the bigger lists's sublist of same size.
// Once the right sublist is added, the carry must be
// added to the left side of larger list to get the
// final result.
void propogatecarry(node head1)
{
    // If diff. number of nodes are not traversed, add carry
    if (head1 != cur)
    {
        propogatecarry(head1.next);
        int sum = carry + head1.val;
        carry = sum / 10;
        sum %= 10;

        // add this node to the front of the result
        push(sum, 3);
    }
}

int getsize(node head)
{
    int count = 0;
    while (head != null)
    {
        count++;
        head = head.next;
    }
    return count;
}

// The main function that adds two linked lists
// represented by head1 and head2. The sum of two
// lists is stored in a list referred by result
void addlists()
{
    // first list is empty
    if (head1 == null)
    {
        result = head2;
        return;
    }

    // first list is empty
    if (head2 == null)
    {
        result = head1;
        return;
    }
```

```
    int size1 = getsize(head1);
    int size2 = getsize(head2);

    // Add same size lists
    if (size1 == size2)
    {
        addsamesize(head1, head2);
    }
    else
    {
        // First list should always be larger than second list.
        // If not, swap pointers
        if (size1 < size2)
        {
            node temp = head1;
            head1 = head2;
            head2 = temp;
        }
        int diff = Math.abs(size1 - size2);

        // move diff. number of nodes in first list
        node temp = head1;
        while (diff-- >= 0)
        {
            cur = temp;
            temp = temp.next;
        }

        // get addition of same size lists
        addsamesize(cur, head2);

        // get addition of remaining first list and carry
        propogatecarry(head1);
    }
        // if some carry is still there, add a new node to
        // the front of the result list. e.g. 999 and 87
        if (carry > 0)
            push(carry, 3);

}

// Driver program to test above functions
public static void main(String args[])
{
    linkedlistATN list = new linkedlistATN();
    list.head1 = null;
    list.head2 = null;
    list.result = null;
    list.carry = 0;
    int arr1[] = { 9, 9, 9 };
    int arr2[] = { 1, 8 };

    // Create first list as 9->9->9
    for (int i = arr1.length - 1; i >= 0; --i)
        list.push(arr1[i], 1);

    // Create second list as 1->8
    for (int i = arr2.length - 1; i >= 0; --i)
        list.push(arr2[i], 2);
```

```
        list.addlists();

        list.printlist(list.result);
    }
}
```

Output:

1 0 1 7

Time Complexity: O(m+n) where m and n are the sizes of given two linked lists.

56. What is the difference between array and linked list? (answer)

Array vs Linked List in Java

Here is my list of some key differences between an array and linked list in Java. Don't try to remember these differences, instead, try to understand that by learning how array and linked list are actually implemented in any programming langue e.g. Java or C++.

Once you understand how array and the linked list is implemented and work in any programming language e.g. Java, you can easily figure out these differences.

1) Flexibility

A linked list is more flexible than array data structure because you can change the size of the linked list once created which is not possible with an array.

A linked list can also grow unlimited but the array cannot grow beyond its size. This is one of the most fundamental differences between an array and a linked list is that the length of the array cannot be changed once created but you can add unlimited elements into linked list unless memory is not a constraint.

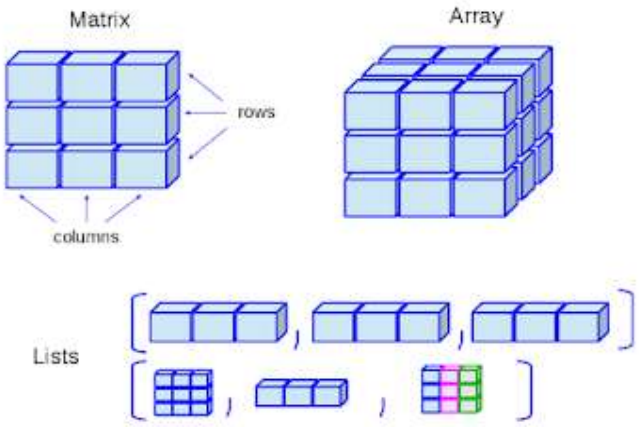
2) Memory utilization

One more significant difference between linked list and array data structure comes from a memory perspective. the array requires a contiguous chunk of memory, which means if you want to create a large array and even if memory is available you may fail because there is no single chunk of memory which is big enough for your array.

This is a huge restriction and that's why any large array should be created at the very start of an application when you have a big chunk of memory available.

A linked list is more flexible in terms of memory as well. Since linked list doesn't need a contiguous chunk of memory and nodes of a linked list can be scattered all around heap memory, it's possible to store more elements in the linked list than array if you have fragmented heap space.

In short, a linked list is a better data structure for memory utilization than an array.



3) Memory required

linked list data structure requires slightly more memory than an array because apart from data i.e. the element you store, linked list node also stores the address of next node.

In Java, the linked list also has object metadata overhead because each Node of the linked list is an object. In short, an array requires less memory than a linked list for storing the same number of elements.

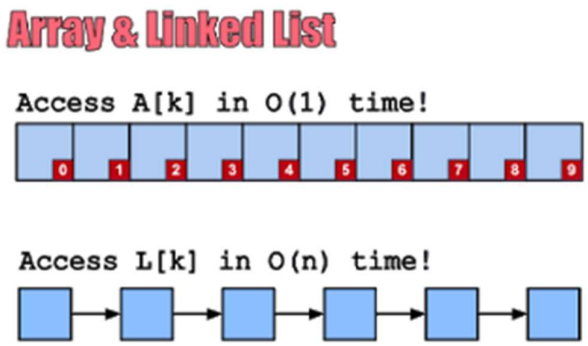
4) Performance

Another key difference between an array and linked list data structure comes from a performance perspective, which is also the main factor to decide when to use the array over the linked list in Java.

An array gives O(1) performance for the searching element when you know the index but linked list search is in order of O(n). So if you need fast retrieval and you know the index then you should use an array.

When it comes performance of adding and deleting element than linked list stores better than an array because adding into head or tail is O(1) operation if you have the necessary pointer but adding at a random position is O(n).

With an array, adding or removing is difficult because it requires rearranging of all other elements as well.



5) Dimension and types

One of the structural difference between linked list and array comes from their variety. The array can be multi-dimensional in Java which makes it ideal data structure for representing matrices, 2D plain, 2D game board, terrain etc.

On the other hand, a linked list has just one dimension but it also comes in two flavors, singly linked list and a doubly linked list.

The Singly linked list holds the address of next node only and thus allows you to move only in one direction i.e. forward but the doubly linked list contains two points, one for storing the address of next node and other for storing the address of the previous node. Which means it allows you to traverse in both forward and backward direction.

Here is a nice summary of some key differences between array and singly linked list data structure in Java:

Singly Linked Lists and Arrays

Singly linked list	Array
Elements are stored in linear order, accessible with links.	Elements are stored in linear order, accessible with an index.
Do not have a fixed size.	Have a fixed size.
Cannot access the previous element directly.	Can access the previous element easily.
No binary search.	Binary search.

That's all about the difference between array and linked list data structure in Java. As I told you,

most of the differences are at the data structure level so they are valid for other programming languages as well e.g. C and C++. The key takeaway is to remember these difference so that programmer can choose when to use an array over the linked list and vice-versa.

57. How to remove duplicates from a sorted linked list? (solution)

Algorithm

This is a simple problem that merely tests your ability to manipulate list node pointers. Because the input list is sorted, we can determine if a node is a duplicate by comparing its value to the node *after* it in the list. If it is a duplicate, we change the `next` pointer of the current node so that it skips the next node and points directly to the one after the next node.

```
public ListNode deleteDuplicates(ListNode head) {
  ListNode current = head;
  while (current != null && current.next != null) {
    if (current.next.val == current.val) {
      current.next = current.next.next;
    } else {
      current = current.next;
    }
  }
  return head;
}
```

58. How to find the node at which the intersection of two singly linked lists begins. (solution)

Here's my solution which is like approach 3 but a little different. I store the size of ListA and ListB as len1 and len2. Then I reset the pointers to headA and headB and find the difference between len1 and len2, and then let the pointer of the longer list proceed by the difference between len1 and len2. Finally, traverse through the lists again, the intersection node can be easily found.

```
class Solution {
public:
    ListNode *getIntersectionNode(ListNode *headA, ListNode *headB) {
        ListNode *p1 = headA;
        ListNode *p2 = headB;
        ListNode *ret = NULL;
```

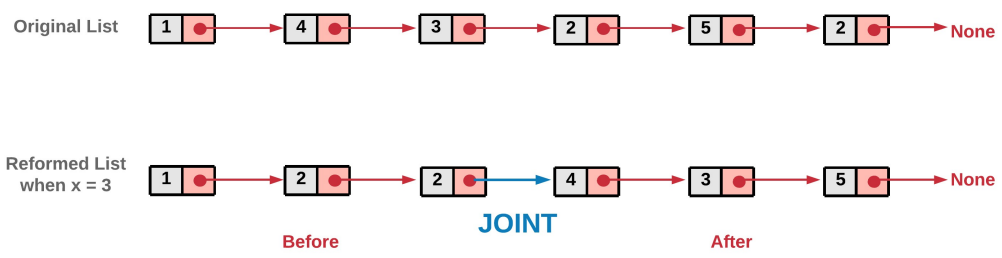
```
        int len1 = 0;
        int len2 = 0;
        while(p1 != NULL) {
            len1++;
            p1 = p1->next;
        }
        while(p2 != NULL) {
            len2++;
            p2 = p2->next;
        }
        p1 = headA;
        p2 = headB;
        if(len1 > len2) {
            int diff = len1 - len2;
            for(int i = 0;i < diff;i++) {
                p1 = p1->next;
            }
        }else {
            int diff = len2 - len1;
            for(int i = 0;i < diff;i++) {
                p2 = p2->next;
            }
        }
        while(p1 != p2) {
            p1 = p1->next;
            p2 = p2->next;
        }
        return p1;
```

```
    }
};
```

59. Given a linked list and a value *x*, partition it such that all nodes less than *x* come before nodes greater than or equal to *x*. (solution)

Solution

The problem wants us to reform the linked list structure, such that the elements lesser that a certain value `x`, come before the elements greater or equal to `x`. This essentially means in this reformed list, there would be a point in the linked list `before` which all the elements would be smaller than `x` and `after` which all the elements would be greater or equal to `x`. Let's call this point as the `JOINT`.



Reverse engineering the question tells us that if we break the reformed list at the `JOINT`, we will get two smaller linked lists, one with lesser elements and the other with elements greater or equal to `x`. In the solution, our main aim is to create these two linked lists and join them.

```
class Solution {

  public ListNode partition(ListNode head, int x) {


    // before and after are the two pointers used to create the two list

    // before_head and after_head are used to save the heads of the two lists.

    // All of these are initialized with the dummy nodes created.

    ListNode before_head = new ListNode(0);

    ListNode before = before_head;

    ListNode after_head = new ListNode(0);
```

```
ListNode after = after_head;

while (head != null) {

   // If the original list node is lesser than the given x,
   // assign it to the before list.
   if (head.val < x) {
      before.next = head;
      before = before.next;
   } else {
      // If the original list node is greater or equal to the given x,
      // assign it to the after list.
      after.next = head;
      after = after.next;
   }

   // move ahead in the original list
   head = head.next;
}

// Last node of "after" list would also be ending node of the reformed list
after.next = null;

// Once all the nodes are correctly assigned to the two lists,
// combine them to form a single list which would be returned.
before.next = after_head.next;

return before_head.next;
```

```
  }

}
```

60. How to check if a given linked list is a palindrome? (solution)

Solution-

- A simple solution is to use a stack of list nodes. This mainly involves three steps.
- Traverse the given list from head to tail and push every visited node to stack.
- Traverse the list again. For every visited node, pop a node from stack and compare data of popped node with currently visited node.
- If all nodes matched, then return true, else false.

```
/* Java program to check if linked list is palindrome recursively */
import java.util.*;

class linkeList {
    public static void main(String args[])
    {
        Node one = new Node(1);
        Node two = new Node(2);
        Node three = new Node(3);
        Node four = new Node(4);
        Node five = new Node(3);
        Node six = new Node(2);
        Node seven = new Node(1);
        one.ptr = two;
        two.ptr = three;
        three.ptr = four;
        four.ptr = five;
        five.ptr = six;
        six.ptr = seven;
        boolean condition = isPalindrome(one);
        System.out.println("isPalidrome :" + condition);
    }
    static boolean isPalindrome(Node head)
    {

        Node slow = head;
        boolean ispalin = true;
        Stack<Integer> stack = new Stack<Integer>();

        while (slow != null) {
            stack.push(slow.data);
            slow = slow.ptr;
        }

        while (head != null) {

            int i = stack.pop();
            if (head.data == i) {
                ispalin = true;
            }
            else {
```

```
                ispalin = false;
                break;
            }
            head = head.ptr;
        }
        return ispalin;
    }
}

class Node {
    int data;
    Node ptr;
    Node(int d)
    {
        ptr = null;
        data = d;
    }
}
```

Output

```
 isPalindrome: true
```

The time complexity of the above method is O(n).

61. How to remove all elements from a linked list of integers which matches with given value? (solution)

Given a singly linked list, delete all occurrences of a given key in it. For example, consider the following list.

```
Input: 2 -> 2 -> 1 -> 8 -> 2 ->  3 ->  2 -> 7

       Key to delete = 2

Output:  1 -> 8 -> 3 -> 7
```

This is mainly an extension of the question which deletes first occurrence of a given key.
We need to first check for all occurrences at head node and change the head node appropriately. Then we need to check for all occurrences inside a loop and delete them one by one. Following is the implementation for the same.

```
// Java Program to delete all occurrences
// of a given key in linked list
class LinkedList
{
    static Node head; // head of list

    /* Linked list Node*/
    class Node
    {
        int data;
        Node next;
        Node(int d)
        {
```

```
        data = d;
        next = null;
    }
}

/* Given a key, deletes all occurrence
of the given key in linked list */
void deleteKey(int key)
{
    // Store head node
    Node temp = head, prev = null;

    // If head node itself holds the key
    // or multiple occurrences of key
    while (temp != null && temp.data == key)
    {
        head = temp.next; // Changed head
        temp = head;         // Change Temp
    }

    // Delete occurrences other than head
    while (temp != null)
    {
        // Search for the key to be deleted,
        // keep track of the previous node
        // as we need to change 'prev->next'
        while (temp != null && temp.data != key)
        {
            prev = temp;
            temp = temp.next;
        }

        // If key was not present in linked list
        if (temp == null) return;

        // Unlink the node from linked list
        prev.next = temp.next;

        //Update Temp for next iteration of outer loop
        temp = prev.next;
    }
}

/* Inserts a new Node at front of the list. */
public void push(int new_data)
{
    Node new_node = new Node(new_data);
    new_node.next = head;
    head = new_node;
}

/* This function prints contents of linked list
starting from the given node */
public void printList()
{
    Node tnode = head;
    while (tnode != null)
    {
        System.out.print(tnode.data + " ");
```

```
            tnode = tnode.next;
        }
    }

    // Driver Code
    public static void main(String[] args)
    {
        LinkedList llist = new LinkedList();

        llist.push(7);
        llist.push(2);
        llist.push(3);
        llist.push(2);
        llist.push(8);
        llist.push(1);
        llist.push(2);
        llist.push(2);

        int key = 2; // key to delete

        System.out.println("Created Linked list is:");
        llist.printList();

        llist.deleteKey(key);

        System.out.println("\nLinked List after Deletion is:");
        llist.printList();
    }
}
```

Output:

Created Linked List:

2 2 1 8 2 3 2 7

Linked List after Deletion of 1:

1 8 3 7

62. How is a binary search tree implemented? (solution)

Binary Search tree in Java

Here, You will learn how to create a binary search tree with integer nodes. I am not using Generics just to keep the code simple but if you like you can extend the problem to use Generics, which will allow you to create a Binary tree of String, Integer, Float or Double. Remember, you make sure that node of BST must implement the Comparable interface. This is what many Java programmer forget when they try to implement binary search tree with Generics.

Here is an implementation of a binary search tree in Java. It's just a structure, we will subsequently add methods to add a node in a binary search tree, delete a node from

binary search tree and find a node from BST in the subsequent part of this binary search tree tutorial.

In this implementation, I have created a `Node` class, which is similar to our linked list node class, which we created when I have shown you how to implement linked list in Java. It has a data element, an integer and a Node reference to point to another node in the binary tree.

I have also created four basic functions, as shown below:

- **getRoot()**, returns the root of binary tree
- **isEmpty()**, to check if binary search tree is empty or not
- **size()**, to find the total number of nodes in a BST
- **clear()**, to clear the BST

```
import java.util.Stack;


/**
 * Java Program to implement a binary search tree. A binary search tree is
a
 * sorted binary tree, where value of a node is greater than or equal to
its
 * left the child and less than or equal to its right child.
 *
 * @author WINDOWS 8
 *
 */
public class BST {

    private static class Node {
        private int data;
        private Node left, right;

        public Node(int value) {
            data = value;
            left = right = null;
        }
    }

    private Node root;

    public BST() {
```

```
    root = null;
}

public Node getRoot() {
    return root;
}

/**
 * Java function to check if binary tree is empty or not
 * Time Complexity of this solution is constant O(1) for
 * best, average and worst case.
 *
 * @return true if binary search tree is empty
 */
public boolean isEmpty() {
    return null == root;
}


/**
 * Java function to return number of nodes in this binary search tree.
 * Time complexity of this method is O(n)
 * @return size of this binary search tree
 */
public int size() {
    Node current = root;
    int size = 0;
    Stack<Node> stack = new Stack<Node>();
    while (!stack.isEmpty() || current != null) {
        if (current != null) {
            stack.push(current);
            current = current.left;
        } else {
            size++;
            current = stack.pop();
            current = current.right;
        }
    }
```

```
        return size;
    }


    /**
     * Java function to clear the binary search tree.
     * Time complexity of this method is O(1)
     */
    public void clear() {
        root = null;
    }

}
```

That's all in this tutorial about **how to implement binary search tree in Java**. In this tutorial, you have learned to create the structure of BST using Node class and some basic function. In next couple of tutorials, you will learn some more interesting things with BST e.g. writing a method to add Nodes in BST, this method must make sure that property of binary search tree is not violated. I mean, it first needs to find a right place and then needs to add the element. Subsequently, you will also learn how to search a node in binary search tree.

63. How do you perform preorder traversal in a given binary tree? (solution)

How to traverse a Binary tree in PreOrder in Java using Recursion

Since binary tree is a recursive data structure (why? because after removing a node, rest of the structure is also a binary tree like left and right binary tree, similar to linked list, which is also a recursive data structure), it's naturally a good candidate for using recursive algorithm to solve tree based problem.

Steps on **PreOrder traversal** algorithm

1. visit the node
2. visit the left subtree
3. visit the right subtree

You can easily implement this **pre-order traversal algorithm** using recursion by printing the value of the node and then recursive calling the same preOrder() method with left subtree and right subtree, as shown in the following program:

```
private void preOrder(TreeNode node) {
    if (node == null) {
      return;
    }
```

```
    System.out.printf("%s ", node.data);
    preOrder(node.left);
    preOrder(node.right);
  }
```

You can see that it's just a couple of line of code. What is most important here is the base case, from where the recursive algorithm starts to unwind. Here node == null is our base case because you have now reached the leaf node and you cannot go deeper now, it's time to backtrack and go to another path.

Binary tree PreOrer traversal in Java without Recursion

One of the easier ways to convert a recursive algorithm to iterative one is by using the Stack data structure. If you remember, recursion implicitly uses a Stack which started to unwind when your algorithm reaches the base case.

You can use external Stack to replace that implicit stack and solve the problem without actually using recursion. This is also safer because now your code will not throw StackOverFlowError even for huge binary search trees but often they are not as concise and readable as their recursive counterpart.

Anyway, here is the **preOrder algorithm without using recursion** in Java.

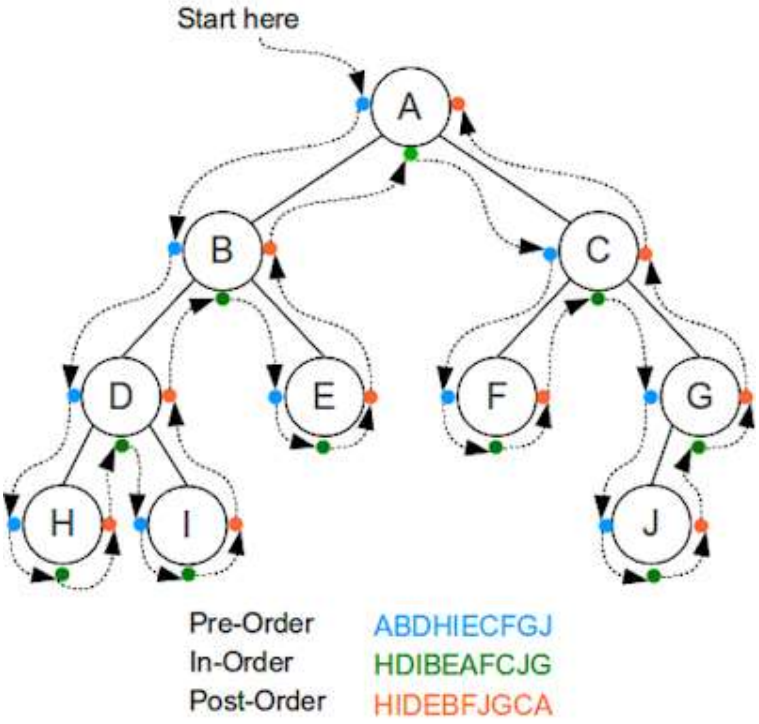
```
public void preOrderWithoutRecursion() {
  Stack<TreeNode> nodes = new Stack<>();
  nodes.push(root);

  while (!nodes.isEmpty()) {
   TreeNode current = nodes.pop();
   System.out.printf("%s ", current.data);

   if (current.right != null) {
    nodes.push(current.right);
   }
   if (current.left != null) {
    nodes.push(current.left);
   }
  }
 }
```

To be honest, this is code is also easy to understand but there is tricky part in the middle of the algorithm, where you have to **push right sub-tree before the left subtree**, which is different from the recursive algorithm. We initially push the root in the Stack to start the traversal and then use a while loop to go over Stack until its empty. In each iteration, we pop

element for visiting it.

If you remember, Stack is a LIFO data structure, since we want to visit the tree in order of node-left-right, we push right node first and left node afterward, so that in the next iteration when we pop() from Stack we get the left sub-tree.

This way a binary tree is traversed in the PreOrder traversal. If you change the order of insertion into the stack, the tree will be traversed in the post-order traversal.
Here is a nice diagram which shows pre-order traversal along with in-order and post-order traversal. Follow the blue line to traverse a binary tree in pre-order.



Java Program to traverse a Binary tree in PreOrder Algorithm

Here is our complete program to traverse a given binary tree in PreOrder. In this program, you will find an implementation of both recursive and iterative pre-order traversal algorithm. You can run this program from the command line or Eclipse IDE to test and get a feel of how tree traversal works.

This program has a class called BinaryTree which represents a BinaryTree, remember it's not

a binary search tree because TreeNode doesn't
implement Comparable or Comparator interface. The TreeNode class represents a node in the binary tree, it contains a data part and two references to left and right children.

I have created a preOrder() method in the BinaryTree class to traverse the tree in pre-order. This is a public method but actual work is done by another private method which is an overloaded version of this method.

The method accepts a TreeNode. Similarly, there is another method
called preOrderWithoutRecursion() to implement the **iterative pre-order traversal** of the binary tree.

```
import java.util.Stack;

/*
 * Java Program to traverse a binary tree using PreOrder traversal.
 * In PreOrder the node value is printed first, followed by visit
 * to left and right subtree.
 * input:
 *     1
 *    / \
 *   2   5
 *  / \   \
 * 3   4   6
 *
 * output: 1 2 3 4 5 6
 */
public class PreOrderTraversal {

  public static void main(String[] args) throws Exception {

    // construct the binary tree given in question
    BinaryTree bt = new BinaryTree();
    BinaryTree.TreeNode root = new BinaryTree.TreeNode("1");
    bt.root = root;
    bt.root.left = new BinaryTree.TreeNode("2");
    bt.root.left.left = new BinaryTree.TreeNode("3");

    bt.root.left.right = new BinaryTree.TreeNode("4");
    bt.root.right = new BinaryTree.TreeNode("5");
    bt.root.right.right = new BinaryTree.TreeNode("6");

    // printing nodes in recursive preOrder traversal algorithm
    bt.preOrder();

    System.out.println();

    // traversing binary tree in PreOrder without using recursion
    bt.preOrderWithoutRecursion();
```

```
  }

}

class BinaryTree {
  static class TreeNode {
    String data;
    TreeNode left, right;

    TreeNode(String value) {
      this.data = value;
      left = right = null;
    }

    boolean isLeaf() {
      return left == null ? right == null : false;
    }

  }

  // root of binary tree
  TreeNode root;

  /**
   * Java method to print tree nodes in PreOrder traversal
   */
  public void preOrder() {
    preOrder(root);
  }

  /**
   * traverse the binary tree in PreOrder
   *
   * @param node
   *          - starting node, root
   */
  private void preOrder(TreeNode node) {
    if (node == null) {
      return;
    }
    System.out.printf("%s ", node.data);
    preOrder(node.left);
    preOrder(node.right);
  }

  /**
   * Java method to visit tree nodes in PreOrder traversal without recursion.
   */
  public void preOrderWithoutRecursion() {
    Stack<TreeNode> nodes = new Stack<>();
    nodes.push(root);

    while (!nodes.isEmpty()) {
      TreeNode current = nodes.pop();
```

```
    System.out.printf("%s ", current.data);

    if (current.right != null) {
      nodes.push(current.right);
    }
    if (current.left != null) {
      nodes.push(current.left);
    }
   }
  }

}
```

Output
1 2 3 4 5 6
1 2 3 4 5 6

That's all about **how to traverse a binary tree in PreOrder in Java**. We'have seen how to implement a pre-order traversing algorithm using both recursion and iteration e.g. by using a Stack data structure.

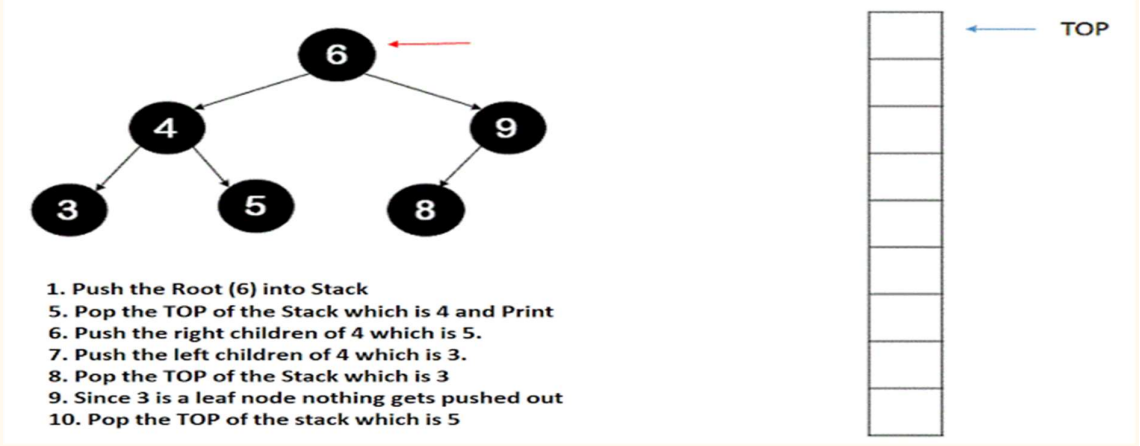
64. How do you traverse a given binary tree in preorder without recursion? (solution)

Java Program to traverse binary tree using preOrder traversal

Here is our complete Java program to print binary tree nodes in the pre-order traversal. You start traversing from the root node by pushing that into Stack. We have used the same class which is used in the earlier binary tree tutorial.

The BinaryTree class is your binary tree and TreeNode is your individual nodes in the tree. This time, though I have moved the logic to create a sample binary tree inside the BinaryTree class itself. This way, you don't need to create a new tree every time in the main() method.

Here is a diagram of the iterative pre-order traversal algorithm which will make the steps clearer:



Iterative Pre-Order Traversal of Binary Tree in Java

```
import java.util.Stack;

/*
 * Java Program to traverse a binary tree
 * using PreOrder traversal without recursion.
 * In PreOrder the node value is printed first,
 * followed by visit to left and right subtree.
 *
 * input:
 *     a
 *    / \
 *   b   e
 *  / \   \
 * c   d   f
 *
 * output: a b c d e f
 */

public class Main {

  public static void main(String[] args) throws Exception {

    // construct the binary tree given in question
    BinaryTree bt = BinaryTree.create();

    // traversing binary tree in PreOrder without using recursion
    System.out
        .println("printing nodes of a binary tree in preOrder using
recursion");

    bt.preOrderWithoutRecursion();

  }

}

class BinaryTree {
```

```
  static class TreeNode {
    String data;
    TreeNode left, right;

    TreeNode(String value) {
      this.data = value;
      left = right = null;
    }

    boolean isLeaf() {
      return left == null ? right == null : false;
    }

  }

  // root of binary tree
  TreeNode root;

  /**
   * Java method to visit tree nodes in PreOrder traversal without
recursion.
   */
  public void preOrderWithoutRecursion() {
    Stack<TreeNode> nodes = new Stack<>();
    nodes.push(root);

    while (!nodes.isEmpty()) {
      TreeNode current = nodes.pop();
      System.out.printf("%s ", current.data);

      if (current.right != null) {
        nodes.push(current.right);
      }
      if (current.left != null) {
        nodes.push(current.left);
      }
    }
  }
```

```
  /**
   * Java method to create binary tree with test data
   *
   * @return a sample binary tree for testing
   */
  public static BinaryTree create() {
    BinaryTree tree = new BinaryTree();
    TreeNode root = new TreeNode("a");
    tree.root = root;
    tree.root.left = new TreeNode("b");
    tree.root.left.left = new TreeNode("c");

    tree.root.left.right = new TreeNode("d");
    tree.root.right = new TreeNode("e");
    tree.root.right.right = new TreeNode("f");

    return tree;
  }

}
```

```
Output
printing nodes of a binary tree in preOrder using recursion
a b c d e f
```

That's all about how to traverse a binary tree using PreOrder traversal in Java. The order in which you visit the node left and right subtree is key because that order determines your traversal algorithm. If you visit the node first means it preOrder, if you visit the node second means its InOrder and when you visit the node last then its called postOrder traversal.

65. How do you perform an inorder traversal in a given binary tree? (solution)

The recursive algorithm to implement InOrder traversal of a Binary tree

The recursive algorithm of inorder traversal is very simple. You just need to call the inOrder() method of BinaryTree class in the order you want to visit the tree. What is most important is to include base case, which is key to any recursive algorithm.

For example, in this problem, the base case is you reach to the leaf node and there is no more node to explore, at that point of time recursion starts to wind down. Here are the exact steps

to traverse binary tree using InOrder traversal:

1. visit left node
2. print value of the root
3. visit right node

and here is the sample code to implement this algorithm using recursion in Java:

```
private void inOrder(TreeNode node) {
   if (node == null) {
    return;
   }

   inOrder(node.left);
   System.out.printf("%s ", node.data);
   inOrder(node.right);
}
```

Similar to preOrder() method in the last example, there is another inOrder() method which exposes inorder traversal to the public and calls this private method which actually performs the InOrder traversal.

This is the standard way to write a recursive method which takes input, it makes it easier for a client to call the method.

```
public void inOrder() {
   inOrder(root);
}
```

You can see that we start with root and then recursive call the inOrder() method with node.left, which means we are going down on left subtree until we hit node == null, which means the last node was a leaf node.

At this point in time, the inOrder() method will return and execute the next line, which prints the node.data. After that its again recursive inOrder() call with node.right, which will initiate the same process again.

Java Program to implement InOrder traversal of a Binary tree

Here is our complete solution of inorder traversal algorithm in Java. This program uses a recursive algorithm to print the value of all nodes of a binary tree using InOrder traversal.

As I have told you before, during in-order traversal value of left subtree is printed first, followed by root and right subtree. If you are interested in the iterative algorithm, you can further check this tutorial of implementing in order traversal without recursion.

```
import java.util.Stack;

/*
 * Java Program to traverse a binary tree
 * using inorder traversal without recursion.
 * In InOrder traversal first left node is visited, followed by root
 * and right node.
 *
 * input:
 *      40
 *     /  \
 *    20   50
 *   / \    \
 *  10  30   60
 * /   /  \
 * 5  67  78
 *
 * output: 5 10 20 30 40 50 60 67 78
 */

public class Main {

  public static void main(String[] args) throws Exception {

    // construct the binary tree given in question
    BinaryTree bt = BinaryTree.create();

    // traversing binary tree using InOrder traversal using recursion
    System.out
        .println("printing nodes of binary tree on InOrder using
recursion");
```

```
    bt.inOrder();
  }

}

class BinaryTree {
  static class TreeNode {
    String data;
    TreeNode left, right;

    TreeNode(String value) {
      this.data = value;
      left = right = null;
    }

  }

  // root of binary tree
  TreeNode root;

  /**
   * traverse the binary tree on InOrder traversal algorithm
   */
  public void inOrder() {
    inOrder(root);
  }

  private void inOrder(TreeNode node) {
    if (node == null) {
      return;
    }

    inOrder(node.left);
    System.out.printf("%s ", node.data);
    inOrder(node.right);
  }

  /**
```

```
    * Java method to create binary tree with test data
    *
    * @return a sample binary tree for testing
    */
   public static BinaryTree create() {
     BinaryTree tree = new BinaryTree();
     TreeNode root = new TreeNode("40");
     tree.root = root;
     tree.root.left = new TreeNode("20");
     tree.root.left.left = new TreeNode("10");
     tree.root.left.left.left = new TreeNode("5");

     tree.root.left.right = new TreeNode("30");
     tree.root.right = new TreeNode("50");
     tree.root.right.right = new TreeNode("60");
     tree.root.left.right.left = new TreeNode("67");
     tree.root.left.right.right = new TreeNode("78");

     return tree;
   }

}
```

```
Output
printing nodes of binary tree on InOrder using recursion
5 10 20 30 67 78 40 50 60
```

That's all about how to implement inOrder traversal of a binary tree in Java using recursion. You can see the code is pretty much similar to the preOrder traversal with the only difference in the order we recursive call the method. In this case, we call inOrder(node.left) first and then print the value of the node.

It's worth remembering that in order traversal is a depth-first algorithm and prints tree node in sorted order if given binary tree is a binary search tree.

66. How do you print all nodes of a given binary tree using inorder traversal without recursion? (solution)

The recursive algorithm to implement InOrder traversal of a Binary tree

The recursive algorithm of inorder traversal is very simple. You just need to call the inOrder() method of BinaryTree class in the order you want to visit the tree. What is most important is to include base case, which is key to any recursive algorithm.

For example, in this problem, the base case is you reach to the leaf node and there is no more node to explore, at that point of time recursion starts to wind down. Here are the exact steps to traverse binary tree using InOrder traversal:

visit left node

print value of the root

visit right node

and here is the sample code to implement this algorithm using recursion in Java:

```
private void inOrder(TreeNode node) {
   if (node == null) {
     return;
   }

   inOrder(node.left);
   System.out.printf("%s ", node.data);
   inOrder(node.right);
}
```

Similar to preOrder() method , there is another inOrder() method which exposes inorder traversal to the public and calls this private method which actually performs the InOrder traversal.

This is the standard way to write a recursive method which takes input, it makes it easier for a client to call the method.

```
public void inOrder() {
   inOrder(root);
}
```

You can see that we start with root and then recursive call the inOrder() method with node.left, which means we are going down on left subtree until we hit node == null, which means the last node was a leaf node.

At this point in time, the inOrder() method will return and execute the next line, which prints the node.data. After that its again recursive inOrder() call with node.right, which will initiate the same process again.

Java Program to implement InOrder traversal of a Binary tree

Here is our complete solution of inorder traversal algorithm in Java. This program uses a recursive algorithm to print the value of all nodes of a binary tree using InOrder traversal.

As I have told you before, during in-order traversal value of left subtree is printed first, followed by root and right subtree. If you are interested in the iterative algorithm, you can further check this tutorial of implementing in order traversal without recursion.

```
import java.util.Stack;

/*
 * Java Program to traverse a binary tree
 * using inorder traversal without recursion.
 * In InOrder traversal first left node is visited, followed by root
 * and right node.
 *
 * input:
 *      40
 *     /  \
 *    20   50
 *   / \    \
 *  10  30   60
 * /   /  \
 * 5  67  78
 *
 * output: 5 10 20 30 40 50 60 67 78
 */

public class Main {

  public static void main(String[] args) throws Exception {
```

```
    // construct the binary tree given in question
    BinaryTree bt = BinaryTree.create();

    // traversing binary tree using InOrder traversal using recursion
    System.out
        .println("printing nodes of binary tree on InOrder using
recursion");

    bt.inOrder();
  }

}

class BinaryTree {
  static class TreeNode {
    String data;
    TreeNode left, right;

    TreeNode(String value) {
      this.data = value;
      left = right = null;
    }

  }

  // root of binary tree
  TreeNode root;

  /**
   * traverse the binary tree on InOrder traversal algorithm
   */
  public void inOrder() {
    inOrder(root);
  }

  private void inOrder(TreeNode node) {
    if (node == null) {
```

```
        return;
      }

      inOrder(node.left);
      System.out.printf("%s ", node.data);
      inOrder(node.right);
    }

    /**
     * Java method to create binary tree with test data
     *
     * @return a sample binary tree for testing
     */
    public static BinaryTree create() {
      BinaryTree tree = new BinaryTree();
      TreeNode root = new TreeNode("40");
      tree.root = root;
      tree.root.left = new TreeNode("20");
      tree.root.left.left = new TreeNode("10");
      tree.root.left.left.left = new TreeNode("5");

      tree.root.left.right = new TreeNode("30");
      tree.root.right = new TreeNode("50");
      tree.root.right.right = new TreeNode("60");
      tree.root.left.right.left = new TreeNode("67");
      tree.root.left.right.right = new TreeNode("78");

      return tree;
    }

}
```

```
Output
printing nodes of binary tree on InOrder using recursion
5 10 20 30 67 78 40 50 60
```

That's all about how to implement inOrder traversal of a binary tree in Java using recursion. You can see the code is pretty much similar to the preOrder traversal with the only difference in the order

we recursive call the method. In this case, we call inOrder(node.left) first and then print the value of the node.

67. How do you implement a postorder traversal algorithm? (solution)

Java Program to print the binary tree in a post-order traversal

Here is the complete Java program to print all nodes of a binary tree in the post-order traversal. In this part of the tutorial, we are learning the recursive post-order traversal and next part, I'll show you how to implement post order algorithm without recursion, one of the toughest tree traversal algorithms for beginner programmers.

Similar to our earlier examples, I have created a class called BinaryTree to represent a binary tree in Java. This class has a static nested class to represent a tree node, called TreeNode. This is similar to the Map. Entry class which is used to represent an entry in the hash table. The class just keep the reference to root and TreeNode takes care of left and right children.

This class has two methods postOrder() and postOrder(TreeNode root), the first one is public and the second one is private. The actual traversing is done in the second method but since root is internal to the class and client don't have access to root, I have created a postOrder() method which calls the private method. This is a common trick to implement a recursive algorithm.

This also gives you the luxury to change your algorithm without affecting clients like tomorrow we can change the recursive algorithm to an iterative one and client will still be calling the post order method without knowing that now the iterative algorithm is in place

Printing nodes of a binary tree in post order

```
import java.util.Stack;


/*
 * Java Program to traverse a binary tree
 * using postOrder traversal without recursion.
 * In postOrder traversal first left subtree is visited, followed by right
subtree
 * and finally data of root or current node is printed.
 *
 * input:
 *     55
```

```
 *    /  \
 *   35   65
 *   / \    \
 *  25  45   75
 *  /  /  \
 * 15 87  98
 *
 * output: 15 25 45 35 87 98 75 65 55
 */

public class Main {

  public static void main(String[] args) throws Exception {

    // construct the binary tree given in question
    BinaryTree bt = BinaryTree.create();

    // traversing binary tree using post-order traversal using recursion
    System.out.println("printing nodes of a binary tree on post order in
Java");

    bt.postOrder();

  }

}

class BinaryTree {
  static class TreeNode {
    String data;
    TreeNode left, right;

    TreeNode(String value) {
      this.data = value;
      left = right = null;
    }

    boolean isLeaf() {
```

```
    return left == null ? right == null : false;
  }

}

// root of binary tree
TreeNode root;

/**
 * traverse the binary tree on post order traversal algorithm
 */
public void postOrder() {
  postOrder(root);
}

private void postOrder(TreeNode node) {
  if (node == null) {
    return;
  }

  postOrder(node.left);
  postOrder(node.right);
  System.out.printf("%s ", node.data);
}

/**
 * Java method to create binary tree with test data
 *
 * @return a sample binary tree for testing
 */
public static BinaryTree create() {
  BinaryTree tree = new BinaryTree();
  TreeNode root = new TreeNode("55");
  tree.root = root;
  tree.root.left = new TreeNode("35");
  tree.root.left.left = new TreeNode("25");
  tree.root.left.left.left = new TreeNode("15");
```

```
    tree.root.left.right = new TreeNode("45");
    tree.root.right = new TreeNode("65");
    tree.root.right.right = new TreeNode("75");
    tree.root.right.right.left = new TreeNode("87");
    tree.root.right.right.right = new TreeNode("98");

    return tree;
  }

}
```

```
Output
printing nodes of a binary tree on post order in Java
15 25 87 98 45 35 75 65 55
```

68. How do you traverse a binary tree in postorder traversal without recursion? (solution)

Java Program for Binary tree PostOrder traversal

Here is our complete Java program to implement post order traversal of a binary tree in Java without using recursion. The iterative algorithm is encapsulated inside the postOrder() method. We have used the same BinaryTree and TreeNode class to implement a binary tree and then added the postOrder() method to print all nodes of a binary tree into post order.

The algorithm we have used doesn't need recursion and it instead uses a while loop and a Stack, traditional tool to convert a recursive algorithm to an iterative one.

```
import java.util.Stack;
/*
 * Java Program to traverse a binary tree
 * using postOrder traversal without recursion.
 * In postOrder traversal first left subtree is visited,
   followed by right subtree
 * and finally data of root or current node is printed.
 *
 * input:
 * 55
 * / \
 * 35 65
 * / \ \
 * 25 45 75
```

```
 * / / \
 * 15 87 98
 *
 * output: 15 25 45 35 87 98 75 65 55
 */

public class Main {

  public static void main(String[] args) throws Exception {

    // construct the binary tree given in question
    BinaryTree bt = BinaryTree.create();

    // traversing binary tree on post order traversal without recursion
    System.out
        .println("printing nodes of binary tree on post order using
iteration");
    bt.postOrderWithoutRecursion();
  }

}

class BinaryTree {
  static class TreeNode {
    String data;
    TreeNode left, right;

    TreeNode(String value) {
      this.data = value;
      left = right = null;
    }

    boolean isLeaf() {
      return left == null ? right == null : false;
    }

  }
```

```
// root of binary tree
TreeNode root;

/**
 * Java method to print all nodes of tree in post-order traversal
 */
public void postOrderWithoutRecursion() {
  Stack<TreeNode> nodes = new Stack<>();
  nodes.push(root);

  while (!nodes.isEmpty()) {
    TreeNode current = nodes.peek();

    if (current.isLeaf()) {
      TreeNode node = nodes.pop();
      System.out.printf("%s ", node.data);
    } else {

      if (current.right != null) {
        nodes.push(current.right);
        current.right = null;
      }

      if (current.left != null) {
        nodes.push(current.left);
        current.left = null;
      }
    }

  }
}

/**
 * Java method to create binary tree with test data
 *
 * @return a sample binary tree for testing
 */
public static BinaryTree create() {
```

```
    BinaryTree tree = new BinaryTree();
    TreeNode root = new TreeNode("55");
    tree.root = root;
    tree.root.left = new TreeNode("35");
    tree.root.left.left = new TreeNode("25");
    tree.root.left.left.left = new TreeNode("15");

    tree.root.left.right = new TreeNode("45");
    tree.root.right = new TreeNode("65");
    tree.root.right.right = new TreeNode("75");
    tree.root.right.right.left = new TreeNode("87");
    tree.root.right.right.right = new TreeNode("98");

    return tree;
  }

}
```

When you will run this program in your favorite IDE e.g. Eclipse or IntelliJIDea, you will see the following output:

```
Output
printing nodes of a binary tree on post order using iteration
15 25 45 35 87 98 75 65 55
```

69. How are all leaves of a binary search tree printed? (solution)

Steps to find all leaf nodes in a binary tree

Here are the steps you can follow to print all leaf nodes of a binary tree:

1. If give tree node or root is null then return
2. print the node if both right and left tree is null, that's your leaf node
3. repeat the process with both left and right subtree

And, here is our Java method to implement this logic into code:

```
  public static void printLeaves(TreeNode node) {
    // base case
    if (node == null) {
```

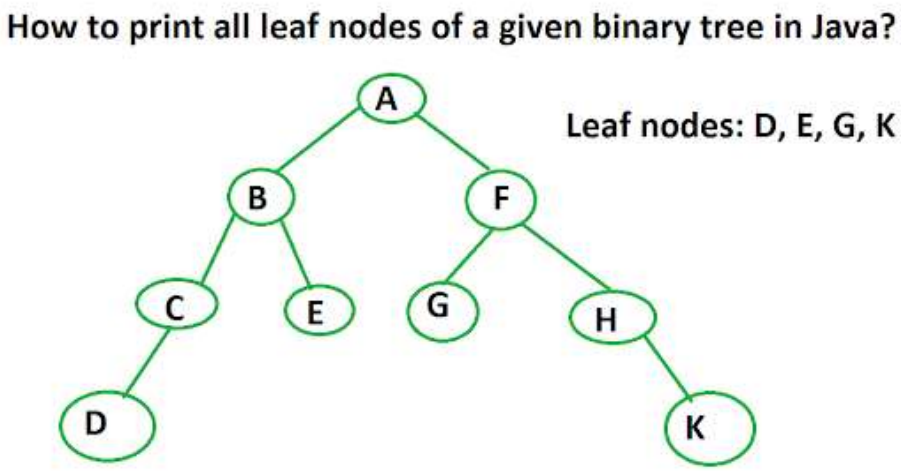
```
      return;
    }

    if (node.isLeaf()) {
      System.out.printf("%s ", node.value);
    }

    printLeaves(node.left);
    printLeaves(node.right);

  }
```

You can see that this method accept a TreeNode, which is nothing but our class to represent a binary tree node. It contains a value and reference to two other nodes, left and right.

In order to start processing, you pass the root node to this method. It then checks if its null or not, if not then it further checks if it's a leaf node or not, if yes, then its print the value of the node and repeat the process with left and right subtree.

This is where recursion is useful because you call the printLeaves() method again with left and right node. The logic to check if a node is a leaf or not is simple, if both left and right children of that node are null then it's a leaf node. This logic is encapsulated in the isLeaf() method of the TreeNode class.

Here is our binary tree with four leaf nodes D, E, G, and K



And, here is our program to print all leaf nodes of this binary tree in Java:

```
/*
 * Java Program to print all leaf nodes of binary tree
 * using recursion
 * input :   a
 *          / \
 *         b   f
 *        / \  / \
 *       c   e g  h
 *      /          \
 *     d            k
 *
 * output: d e g k
 */

public class Main {

  public static void main(String[] args) throws Exception {

    // let's create a binary tree
    TreeNode d = new TreeNode("d");
    TreeNode e = new TreeNode("e");
    TreeNode g = new TreeNode("g");
    TreeNode k = new TreeNode("k");

    TreeNode c = new TreeNode("c", d, null);
    TreeNode h = new TreeNode("h", k, null);

    TreeNode b = new TreeNode("b", c, e);
    TreeNode f = new TreeNode("f", g, h);

    TreeNode root = new TreeNode("a", b, f);

    // print all leaf nodes of binary tree using recursion
    System.out
        .println("Printing all leaf nodes of binary tree in Java
(recursively)");
    printLeaves(root);
```

```
}

/**
 * A class to represent a node in binary tree
 */
private static class TreeNode {
  String value;
  TreeNode left;
  TreeNode right;

  TreeNode(String value) {
    this.value = value;
  }

  TreeNode(String data, TreeNode left, TreeNode right) {
    this.value = data;
    this.left = left;
    this.right = right;
  }

  boolean isLeaf() {
    return left == null ? right == null : false;
  }
}

/**
 * Java method to print leaf nodes using recursion
 *
 * @param root
 *
 */
public static void printLeaves(TreeNode node) {
  // base case
  if (node == null) {
    return;
  }

  if (node.isLeaf()) {
```

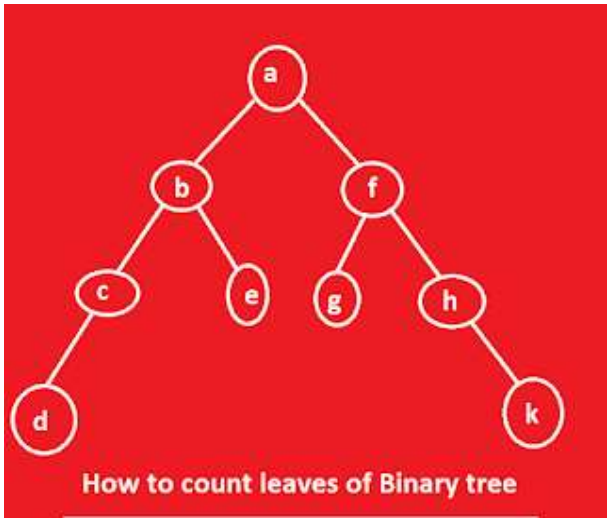
```
      System.out.printf("%s ", node.value);
    }

    printLeaves(node.left);
    printLeaves(node.right);

  }
}
```

```
Output
Printing all leaf nodes of binary tree in Java (recursively)
d e g k
```

70. How do you count a number of leaf nodes in a given binary tree? (solution)

Java Program to count the number of leaf nodes in a binary tree

Here is the complete **program to count a total number of leaf nodes in a given binary tree in Java**. This program demonstrates both recursive and iterative algorithm to solve this problem. In this program, we'll use the following binary tree for testing purpose.



Since there are four leaf nodes in this tree (d, e, g, and h), your program should print 4.
The countLeafNodesRecursively() method solves this problem
using recursion and countLeafNodes() solves this problem without recursion.

The working of methods is explained in the previous paragraph.

```
import java.util.Stack;

/*
 * Java Program to count all leaf nodes of binary tree
 * with and without recursion.
 * input : a
 *         / \
 *        b   f
 *       / \  / \
 *      c  e g  h
 *     /         \
 *    d           k
 *
 * output: 4
 */

public class Main {

  public static void main(String[] args) throws Exception {

    // let's create a binary tree
    BinaryTree bt = new BinaryTree();
    bt.root = new BinaryTree.TreeNode("a");
    bt.root.left = new BinaryTree.TreeNode("b");
    bt.root.right = new BinaryTree.TreeNode("f");
    bt.root.left.left = new BinaryTree.TreeNode("c");
    bt.root.left.right = new BinaryTree.TreeNode("e");
    bt.root.left.left.left = new BinaryTree.TreeNode("d");
    bt.root.right.left = new BinaryTree.TreeNode("g");
    bt.root.right.right = new BinaryTree.TreeNode("h");
    bt.root.right.right.right = new BinaryTree.TreeNode("k");

    // count all leaf nodes of binary tree using recursion
    System.out
        .println("total number of leaf nodes of binary tree in Java
(recursively)");
```

```
    System.out.println(bt.countLeafNodesRecursively());

    // count all leaf nodes of binary tree without recursion
    System.out
        .println("count of leaf nodes of binary tree in Java
(iteration)");
    System.out.println(bt.countLeafNodes());

  }

}

class BinaryTree {
  static class TreeNode {
    String value;
    TreeNode left, right;

    TreeNode(String value) {
      this.value = value;
      left = right = null;
    }

    boolean isLeaf() {
      return left == null ? right == null : false;
    }

  }

  // root of binary tree
  TreeNode root;

  /**
   * Java method to calculate number of leaf node in binary tree.
   *
   * @param node
   * @return count of leaf nodes.
   */
  public int countLeafNodesRecursively() {
```

```
  return countLeaves(root);
}

private int countLeaves(TreeNode node) {
  if (node == null)
    return 0;

  if (node.isLeaf()) {
    return 1;
  } else {
    return countLeaves(node.left) + countLeaves(node.right);
  }
}

/**
 * Java method to count leaf nodes using iteration
 *
 * @param root
 * @return number of leaf nodes
 *
 */
public int countLeafNodes() {
  if (root == null) {
    return 0;
  }

  Stack<TreeNode> stack = new Stack<>();
  stack.push(root);
  int count = 0;

  while (!stack.isEmpty()) {
    TreeNode node = stack.pop();
    if (node.left != null)
      stack.push(node.left);
    if (node.right != null)
      stack.push(node.right);
    if (node.isLeaf())
      count++;
```

```
    }

    return count;

  }
}
```

```
Output
total number of leaf nodes of a binary tree in Java (recursively)
4
count of leaf nodes of a binary tree in Java (iteration)
4
```

That's all about **how to count the number of leaf nodes in a binary tree in Java.** You have learned to solve this problem both by using recursion and iteration. As in most cases, the recursive algorithm is easier to write, read and understand that the iterative algorithm, but, even the iterative algorithm is not as tough as the iterative quicksort or iterative post-order traversal algorithm.

71. How do you perform a binary search in a given array? (solution)

Binary Search tree in Java

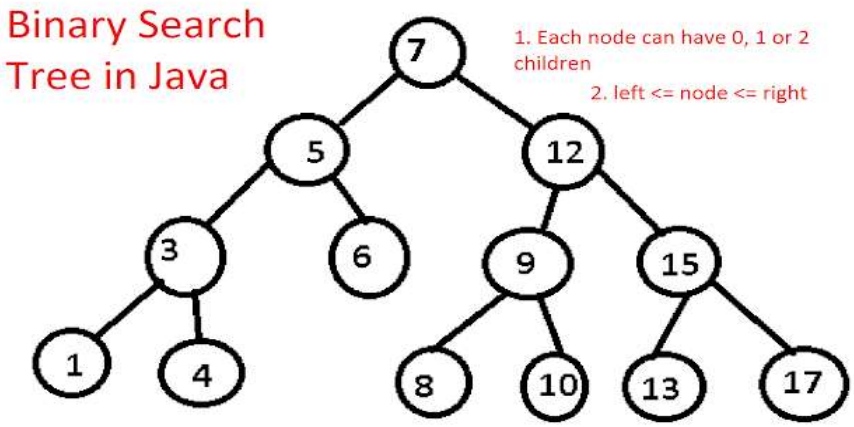
Here, You will learn how to create a binary search tree with integer nodes. I am not using Generics just to keep the code simple but if you like you can extend the problem to use Generics, which will allow you to create a Binary tree of String, Integer, Float or Double. Remember, you make sure that node of BST must implement the Comparable interface. This is what many Java programmer forget when they try to implement binary search tree with Generics.

Here is an implementation of a binary search tree in Java. It's just a structure, we will subsequently add methods to add a node in a binary search tree, delete a node from binary search tree and find a node from BST in the subsequent part of this binary search tree tutorial.

In this implementation, I have created a Node class, which is similar to our linked list node class, which we created when I have shown you how to implement linked list in Java. It has a data element, an integer and a Node reference to point to another node in the binary tree.

I have also created four basic functions, as shown below:

- **getRoot()**, returns the root of binary tree
- **isEmpty()**, to check if binary search tree is empty or not
- **size()**, to find the total number of nodes in a BST
- **clear()**, to clear the BST




```
import java.util.Stack;

/**
 * Java Program to implement a binary search tree. A binary search tree is a
 * sorted binary tree, where value of a node is greater than or equal to its
 * left the child and less than or equal to its right child.
 *
 * @author WINDOWS 8
 *
 */
public class BST {

    private static class Node {
        private int data;
        private Node left, right;

        public Node(int value) {
            data = value;
            left = right = null;
        }
    }

    private Node root;
```

```
public BST() {
    root = null;
}

public Node getRoot() {
    return root;
}

/**
 * Java function to check if binary tree is empty or not
 * Time Complexity of this solution is constant O(1) for
 * best, average and worst case.
 *
 * @return true if binary search tree is empty
 */
public boolean isEmpty() {
    return null == root;
}



/**
 * Java function to return number of nodes in this binary search tree.
 * Time complexity of this method is O(n)
 * @return size of this binary search tree
 */
public int size() {
    Node current = root;
    int size = 0;
    Stack<Node> stack = new Stack<Node>();
    while (!stack.isEmpty() || current != null) {
        if (current != null) {
            stack.push(current);
            current = current.left;
        } else {
            size++;
            current = stack.pop();
            current = current.right;
        }
```

```
        }
        return size;
    }


    /**
     * Java function to clear the binary search tree.
     * Time complexity of this method is O(1)
     */
    public void clear() {
        root = null;
    }

}
```

That's all in this tutorial about **how to implement binary search tree in Java**. In this tutorial, you have learned to create the structure of BST using Node class and some basic function. In next couple of tutorials, you will learn some more interesting things with BST e.g. writing a method to add Nodes in BST, this method must make sure that property of binary search tree is not violated. I mean, it first needs to find a right place and then needs to add the element. Subsequently, you will also learn how to search a node in binary search tree.

72. How to implement an LRU Cache in your favorite programming language? (solution)

We use two data structures to implement an LRU Cache.

1. **Queue** which is implemented using a doubly linked list. The maximum size of the queue will be equal to the total number of frames available (cache size). The most recently used pages will be near front end and least recently pages will be near the rear end.
2. **A Hash** with page number as key and address of the corresponding queue node as value.

When a page is referenced, the required page may be in the memory. If it is in the memory, we need to detach the node of the list and bring it to the front of the queue.
If the required page is not in memory, we bring that in memory. In simple words, we add a new node to the front of the queue and update the corresponding node address in the hash. If the queue is full, i.e. all the frames are full, we remove a node from the rear of the queue, and add the new node to the front of the queue.

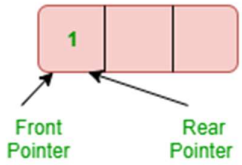
Example – Consider the following reference string :

```
1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
```

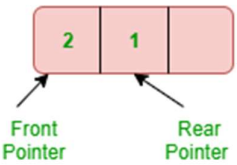
Find the number of page faults using least recently used (LRU) page replacement algorithm with 3 page frames.
Explanation –

Given 3 page frames, so we take size of Queue is 3. Initially, Queue is empty.

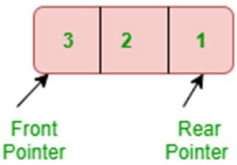
Input : 1



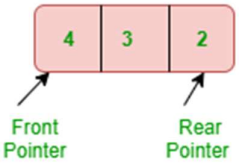
Input : 2 (every new input
will be front as defined LRU)



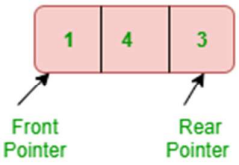
Input : 3



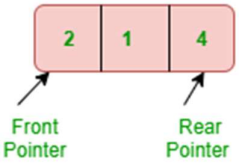
Input : 4 (since all the frames are full, we remove a node from the rear of queue, and add the new node to the front of queue.)



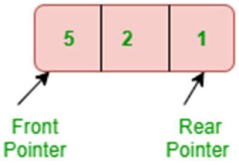
Input : 1 (since all the frames are full, we remove a node from the rear of queue, and add the new node to the front of queue.)



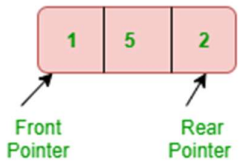
Input : 2 (since all the frames are full, we remove a node from the rear of queue, and add the new node to the front of queue.)



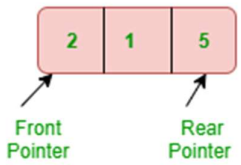
Input : 5 (since all the frames are full, we remove a node from the rear of queue, and add the new node to the front of queue.)



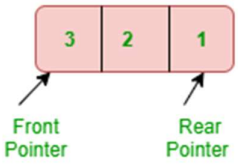
Input : 1 (since present in memory, so bring it to the front of the queue. This is called hit)



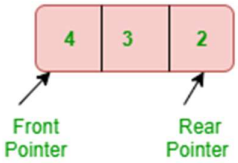
Input : 2 (since present in memory, so bring it to the front of the queue. This is called hit)



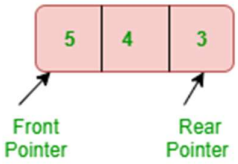
Input : 3 (since all the frames are full, we remove a node from the rear of queue, and add the new node to the front of queue.)



Input : 4 (since all the frames are full, we remove a node from the rear of queue, and add the new node to the front of queue.)



Input : 5 (since all the frames are full, we remove a node from the rear of queue, and add the new node to the front of queue.)



So, we have only 2 hits and 10 page faults using LRU page replacement algorithm.

Note: Initially no page is in the memory.

```
/* We can use Java inbuilt Deque as a double
   ended queue to store the cache keys, with
   the descending time of reference from front
   to back and a set container to check presence
   of a key. But remove a key from the Deque using
   remove(), it takes O(N) time. This can be
   optimized by storing a reference (iterator) to
   each key in a hash map. */
import java.util.Deque;
```

```
import java.util.HashSet;
import java.util.LinkedList;
import java.util.Iterator;
public class LRUCache {
    // store keys of cache
    static Deque<Integer> dq;
    // store references of key in cache
    static HashSet<Integer> map;
    // maximum capacity of cache
    static int csize;

    LRUCache(int n)
    {
        dq = new LinkedList<>();
        map = new HashSet<>();
        csize = n;
    }

    /* Refers key x with in the LRU cache */
    public void refer(int x)
    {
        if (!map.contains(x)) {
            if (dq.size() == csize) {
                int last = dq.removeLast();
                map.remove(last);
            }
        }
        else {
            /* The found page may not be always the last element, even if
it's an
               intermediate element that needs to be removed and added to
the start
               of the Queue */
            int index = 0, i = 0;
            Iterator<Integer> itr = dq.iterator();
            while (itr.hasNext()) {
                if (itr.next() == x) {
                    index = i;
                    break;
                }
                i++;
            }
            dq.remove(index);
        }
        dq.push(x);
        map.add(x);
    }

    // display contents of cache
    public void display()
    {
        Iterator<Integer> itr = dq.iterator();
        while (itr.hasNext()) {
            System.out.print(itr.next() + " ");
        }
    }

    public static void main(String[] args)
    {
        LRUCache ca = new LRUCache(4);
```

```
        ca.refer(1);
        ca.refer(2);
        ca.refer(3);
        ca.refer(1);
        ca.refer(4);
        ca.refer(5);
        ca.display();
    }
}
```

Output: 5 4 1 3

73. How to check if a given number is a Palindrome? (solution)

How to check if String is Palindrome in Java using Recursion

```
package test;

/**
 * Java program to show you how to check if a String is palindrome or not.
 * An String is said to be palindrome if it is equal to itself after
reversing.
 * In this program, you will learn how to check if a string is a
palindrome in java using recursion
 * and for loop both.
 *
 * @author Ocj4u
 */
public class PalindromeTest {
    public static void main(String args[]) {
        System.out.println("Is aaa palindrom?: " +
isPalindromString("aaa"));
        System.out.println("Is abc palindrom?: " +
isPalindromString("abc"));

        System.out.println("Is bbbb palindrom?: " +
isPalindromString("bbbb"));
        System.out.println("Is defg palindrom?: " +
isPalindromString("defg"));
```

```
    }

    /**
     * Java method to check if given String is Palindrome
     * @param text
     * @return true if text is palindrome, otherwise false
     */
    public static boolean isPalindromString(String text){
       String reverse = reverse(text);
       if(text.equals(reverse)){
           return true;
       }

       return false;
    }

    /**
     * Java method to reverse String using recursion
     * @param input
     * @return reversed String of input
     */
    public static String reverse(String input){
        if(input == null || input.isEmpty()){
            return input;
        }

        return input.charAt(input.length()- 1) +
reverse(input.substring(0, input.length() - 1));
    }

}
Output
Is aaa palindrom?: true
Is abc palindrom?: false
Is bbbb palindrom?: true
Is defg palindrom?: false
```

How to check if String is Palindrome using StringBuffer and For loop

```
import java.util.Scanner;

/**
 * How to check if String is palindrome in Java
 * using StringBuffer and for loop.
 *
 * @author ocj4u
 */

public class Palindrome{

    public static void main(String args[]) {

        Scanner reader = new Scanner(System.in);
        System.out.println("Please enter a String");
        String input = reader.nextLine();

        System.out.printf("Is %s a palindrome? : %b %n", input,
isPalindrome(input));



        System.out.println("Please enter another String");
        input = reader.nextLine();

        System.out.printf("Is %s a palindrome? : %b %n", input,
isPalindrome(input));

        reader.close();



    }

    public static boolean isPalindrome(String input) {
        if (input == null || input.isEmpty()) {
            return true;
        }
```

```
        char[] array = input.toCharArray();
        StringBuilder sb = new StringBuilder(input.length());
        for (int i = input.length() - 1; i >= 0; i--) {
            sb.append(array[i]);
        }

        String reverseOfString = sb.toString();

        return input.equals(reverseOfString);
    }

}
```

That's all about how to check for palindrome in Java. You have learned how to find if a given String is palindrome using recursion as well by using StringBuffer and for loop. More importantly you have done it by developing your own logic and writing your own code i.e. not taking help from third party library. If you want to do, you can write some unit test for our recursive and iterative palindrome functions and see if it works in all conditions including corner cases.

74. How to check if a given number is an Armstrong number? (solution)

Java Program to Find Armstrong Number

```
import java.util.Scanner;

/**
 * Simple Java Program to check or find if a number is
Armstrong number or not.
 * An Armstrong number of three digit is a number whose sum of
cubes of its digit is equal
 * to its number. For example 153 is an Armstrong number of 3
digit because 1^3+5^3+3^3 or   1+125+27=153
 * @author Ocj4u
 */
public class ArmstrongTest{


    public static void main(String args[]) {

        //input number to check if its Armstrong number
        System.out.println("Please enter a 3 digit number to
find if
                                    its an Armstrong number:");
        int number = new Scanner(System.in).nextInt();
```

```
        //printing result
        if(isArmStrong(number)){
            System.out.println("Number : " + number + " is an
Armstrong number");
        }else{
            System.out.println("Number : " + number + " is not
an Armstrong number");
        }


    }

    /*
     * @return true if number is Armstrong number or return
false
     */
    private static boolean isArmStrong(int number) {
        int result = 0;
        int orig = number;
        while(number != 0){
            int remainder = number%10;
            result = result + remainder*remainder*remainder;
            number = number/10;
        }
        //number is Armstrong return true
        if(orig == result){
            return true;
        }

        return false;
    }

}
```

Output:

```
Please enter a 3 digit number to find if its an Armstrong
number:
153
Number : 153 is an Armstrong number
Please enter a 3 digit number to find if its an Armstrong
number:
153
Number : 153 is an Armstrong number
Please enter a 3 digit number to find if its an Armstrong
number:
371
Number : 371 is an Armstrong number
```

75. How to find all prime factors of a given number? (solution)

```
// Program to print all prime factors
import java.io.*;
import java.lang.Math;

class GFG {
    // A function to print all prime factors
    // of a given number n
    public static void primeFactors(int n)
    {
        // Print the number of 2s that divide n
        while (n % 2 == 0) {
            System.out.print(2 + " ");
            n /= 2;
        }

        // n must be odd at this point.  So we can
        // skip one element (Note i = i +2)
        for (int i = 3; i <= Math.sqrt(n); i += 2) {
            // While i divides n, print i and divide n
            while (n % i == 0) {
                System.out.print(i + " ");
                n /= i;
            }
        }

        // This condition is to handle the case whien
        // n is a prime number greater than 2
        if (n > 2)
            System.out.print(n);
    }

    public static void main(String[] args)
    {
        int n = 315;
        primeFactors(n);
    }
}
```

Output:

```
3 3 5 7
```

How does this work?

- Steps 1 and 2 take care of composite number and step-3 takes care of prime numbers. To prove that the complete algorithm works, we need to prove that steps 1 and 2 actually take care of composite numbers.
  It's clear that step-1 takes care of even numbers. After step-1, all remaining prime factor must be odd (difference of two prime factors must be at least 2), this explains why i is incremented by 2.

- Now the main part is, the loop runs till square root of *n*. To prove that this optimization works, let us consider the following property of composite numbers.

Every composite number has at least one prime factor less than or equal to square root of itself.

This property can be proved using counter statement. Let a and b be two factors of n such that a*b = n. If both are greater than √n, then a.b > √n, * √n, which contradicts the expression "a * b = n".

- In step-2 of the above algorithm, we run a loop and do following-
    - Find the least prime factor i (must be less than √n,)
    - Remove all occurrences i from n by repeatedly dividing n by i.
    - Repeat steps a and b for divided n and i = i + 2. The steps a and b are repeated till n becomes either 1 or a prime number.

76. How to check if a given number is positive or negative in Java? (solution)

Program-1

```
public static String checkSignWithRelational(double number){
        if( number < 0){
            return "negative";
        }else {
            return "positive";
        }
}



Testing:

System.out.println("0.0 is " + checkSignWithRelational(0.0));
System.out.println("2.0 is " + checkSignWithRelational(2.0));
System.out.println("-2.0 is " + checkSignWithRelational(-2.0));
System.out.println("Double.POSITIVE_INFINITY is " +
checkSignWithRelational(Double.POSITIVE_INFINITY));
System.out.println("Double.NEGATIVE_INFINITY is " +
checkSignWithRelational(Double.NEGATIVE_INFINITY));

Output
0.0 is positive
2.0 is positive
-2.0 is negative
Double.POSITIVE_INFINITY is positive
Double.NEGATIVE_INFINITY is negative
```

Program-2

```
public static String checkSign(int number){
        if(number == 0) return "positive";
        if(number >> 31 != 0){
            return "negative";
        }else{
```

```
            return "positive";
        }
    }

    public static String checkSign(long number){
        if(number == 0) return "positive";
        if(number >> 63 != 0){
            return "negative";
        }else{
            return "positive";
        }
    }
```

Testing for checkSign(int) method

```
System.out.println("0 is " + checkSign(0));
System.out.println("2 is " + checkSign(2));
System.out.println("-2 is " + checkSign(-2));
System.out.println("Integer.MAX_VALUE is " + checkSign(Integer.MAX_VALUE));
System.out.println("Integer.MIN_VALUE is " + checkSign(Integer.MIN_VALUE));
```

Output

```
0 is positive
2 is positive
-2 is negative
Integer.MAX_VALUE is positive
Integer.MIN_VALUE is negative
```

77. How to find the largest prime factor of a given integral number? (solution)

```
import java.util.Random;
import java.util.Scanner;

/**
* Java program to find and print largest prime factor of an integer
number. for
* example number 6 has two prime factors 2 and 3, but 3 is the largest
prime
* factor of 6. input 15 output 5
*
* @author Javin Paul
*/
public class LargetPrimeFactor{

    public static void main(String args[]) {

        System.out.printf("Largest prime factor of number '%d' is %d %n",
                6, largestPrimeFactor(6));
        System.out.printf("highest prime factor of number '%d' is %d %n",
```

```
                15, largestPrimeFactor(15));
        System.out.printf("Biggest prime factor of number '%d' is %d %n",
                392832, largestPrimeFactor(392832));
        System.out.printf("Largest prime factor of number '%d' is %d %n",
                1787866, largestPrimeFactor(1787866));
    }

    /**
     * @return largest prime factor of a number
     */
    public static int largestPrimeFactor(long number) {
        int i;
        long copyOfInput = number;

        for (i = 2; i <= copyOfInput; i++) {
            if (copyOfInput % i == 0) {
                copyOfInput /= i;
                i--;
            }
        }

        return i;
    }

}

Output:
Largest prime factor of number '6' is 3
highest prime factor of number '15' is 5
Biggest prime factor of number '392832' is 31
Largest prime factor of number '1787866' is 893933
```

You can see from output that our program is working properly, for 6 the largest prime factor is 3 and for 15 its 5. Here is our unit test to check couple of more numbers :

```
import static org.junit.Assert.assertEquals;

import org.junit.Test;
```

```
public class LargestPrimeFactorTest {


    @Test
    public void testLargestPrimeFactors(){
        assertEquals(2, HelloWorld.largestPrimeFactor(2));
        assertEquals(3, HelloWorld.largestPrimeFactor(6));
        assertEquals(5, HelloWorld.largestPrimeFactor(15));
        assertEquals(7, HelloWorld.largestPrimeFactor(147));
        assertEquals(17, HelloWorld.largestPrimeFactor(17));
        assertEquals(31, HelloWorld.largestPrimeFactor(392832));
        assertEquals(893933, HelloWorld.largestPrimeFactor(1787866));
    }
}
```

78. How to print all prime numbers up to a given number? (solution)

```
import java.util.Scanner;

/**
 * Simple Java program to print prime numbers from 1 to 100 or any number.
 * A prime number is a number which is greater than 1 and divisible
 * by either 1 or itself.
 */
public class PrimeNumberExample {

    public static void main(String args[]) {

     //get input till which prime number to be printed
      System.out.println("Enter the number till which prime number to be printed:
");
      int limit = new Scanner(System.in).nextInt();

      //printing primer numbers till the limit ( 1 to 100)
      System.out.println("Printing prime number from 1 to " + limit);
      for(int number = 2; number<=limit; number++){
          //print prime numbers only
          if(isPrime(number)){
              System.out.println(number);
          }
      }

    }

    /*
```

```
     * Prime number is not divisible by any number other than 1 and itself
     * @return true if number is prime
     */
    public static boolean isPrime(int number){
        for(int i=2; i<number; i++){
           if(number%i == 0){
               return false; //number is divisible so its not prime
           }
        }
        return true; //number is prime now
    }
}
```

Output:

```
Enter the number till which prime number to be printed:
20
Printing prime number from 1 to 20
2
3
5
7
11
13
17
19
```

79. How to print Floyd's triangle? (solution)

```
import java.util.Scanner;
/**
* Java program to print Floyd's triangle up-to a given row
*
* @author Javin Paul
*/
public class FloydTriangle {

    public static void main(String args[]) {
        Scanner cmd = new Scanner(System.in);

        System.out.println("Enter the number of rows of Floyd's triangle,
you want to display");
        int rows = cmd.nextInt();
        printFloydTriangle(rows);

    }

    /**
     * Prints Floyd's triangle of a given row
     *
     * @param rows
     */
    public static void printFloydTriangle(int rows) {
        int number = 1;
        System.out.printf("Floyd's triangle of %d rows is : %n", rows);

        for (int i = 1; i <= rows; i++) {
            for (int j = 1; j <= i; j++) {
                System.out.print(number + " ");
                number++;
            }
```

```
            System.out.println();
        }
    }

}
```

Output

```
Enter the number of rows of Floyd's triangle, you want to display
5
Floyd's triangle of 5 rows is :
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15

Enter the number of rows of Floyd's triangle, you want to display
10
Floyd's triangle of 10 rows is :
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 31 32 33 34 35 36
37 38 39 40 41 42 43 44 45
46 47 48 49 50 51 52 53 54 55
```

That's all about **how do you print Floyd's triangle in Java**. It's a very good programming exercise to build your foundation on loops especially nested loops. After that you can also try couple of other pattern based programming task's e.g. printing Pascal's triangle and some other patterns.

80. How to print Pascal's triangle? (solution)

```
import java.util.Scanner;


/*
 * Java Program to print Pascal's triangle for given number of rows
 *
 */
public class PascalTriangleInJava {

    public static void main(String[] args) {

        System.out.println("Welcome to Java program to print Pascal's
triangle");
```

```
        System.out.println("Please enter number of rows of Pascal's
triangle");

        // Using try with resource statment to open Scanner
        // no need to close Scanner later
        try (Scanner scnr = new Scanner(System.in)) {
            int rows = scnr.nextInt();
            System.out.printf("Pascal's triangle with %d rows %n", rows);
            printPascalTriangle(rows);
        }
    }

    /**
     * Java method to print Pascal's triangle for given number of rows
     *
     * @param rows
     */
    public static void printPascalTriangle(int rows) {
        for (int i = 0; i < rows; i++) {
            int number = 1;
            System.out.printf("%" + (rows - i) * 2 + "s", "");
            for (int j = 0; j <= i; j++) {
                System.out.printf("%4d", number);
                number = number * (i - j) / (j + 1);

            }
            System.out.println();
        }
    }

}
```

Output

```
Welcome to Java program to print Pascal's triangle
Please enter number of rows of Pascal's triangle
4
Pascal's triangle with 4 rows
```

```
          1
        1   1
      1   2   1
    1   3   3   1
```

```
Welcome to Java program to print Pascal's triangle
Please enter number of rows of Pascal's triangle
7
Pascal's triangle with 7 rows
                1
              1   1
            1   2   1
          1   3   3   1
        1   4   6   4   1
      1   5  10  10   5   1
    1   6  15  20  15   6   1
```

81. How to calculate the square root of a given number? (solution)

```
// A Java program to find floor(sqrt(x))

class GFG {

    // Returns floor of square root of x
    static int floorSqrt(int x)
    {
        // Base cases
        if (x == 0 || x == 1)
            return x;

        // Staring from 1, try all numbers until
        // i*i is greater than or equal to x.
        int i = 1, result = 1;

        while (result <= x) {
            i++;
            result = i * i;
        }
        return i - 1;
    }

    // Driver program
    public static void main(String[] args)
    {
        int x = 11;
        System.out.print(floorSqrt(x));
```

```
    }
}
```

Program-2

```
// A Java program to find floor(sqrt(x)
public class Test
{
    public static int floorSqrt(int x)
    {
        // Base Cases
        if (x == 0 || x == 1)
            return x;

        // Do Binary Search for floor(sqrt(x))
        int start = 1, end = x, ans=0;
        while (start <= end)
        {
            int mid = (start + end) / 2;

            // If x is a perfect square
            if (mid*mid == x)
                return mid;

            // Since we need floor, we update answer when mid*mid is
            // smaller than x, and move closer to sqrt(x)
            if (mid*mid < x)
            {
                start = mid + 1;
                ans = mid;
            }
            else    // If mid*mid is greater than x
                end = mid-1;
        }
        return ans;
    }

    // Driver Method
    public static void main(String args[])
    {
        int x = 11;
        System.out.println(floorSqrt(x));
    }
}
```

82. How to check if the given number is a prime number? (solution)

```
import java.util.Scanner;
/**
 * Java Program to check if a number is Prime or Not. This
program accepts a
 * number from command prompt and check if it is prime or not.
 *
```

```
 * @author  ocj4u
 */
public class Testing {

    public static void main(String args[]) {
        Scanner scnr = new Scanner(System.in);
        int number = Integer.MAX_VALUE;
        System.out.println("Enter number to check if prime or
not ");
        while (number != 0) {
            number = scnr.nextInt();
            System.out.printf("Does %d is prime? %s %s  %s
%n", number,
                    isPrime(number), isPrimeOrNot(number),
isPrimeNumber(number));
        }
    }


    /*
     * Java method to check if an integer number is prime or
not.
     * @return true if number is prime, else false
     */
    public static boolean isPrime(int number) {
        int sqrt = (int) Math.sqrt(number) + 1;
        for (int i = 2; i < sqrt; i++) {
            if (number % i == 0) {
                // number is perfectly divisible - no prime
                return false;
            }
        }
        return true;
    }


    /*
     * Second version of isPrimeNumber method, with
improvement like not
     * checking for division by even number, if its not
divisible by 2.
     */
    public static boolean isPrimeNumber(int number) {
        if (number == 2 || number == 3) {
            return true;
        }
        if (number % 2 == 0) {
            return false;
        }
        int sqrt = (int) Math.sqrt(number) + 1;
        for (int i = 3; i < sqrt; i += 2) {
```

```
            if (number % i == 0) {
                return false;
            }
        }
        return true;
    }


    /*
     * Third way to check if a number is prime or not.
     */
    public static String isPrimeOrNot(int num) {
        if (num < 0) {
            return "not valid";
        }
        if (num == 0 || num == 1) {
            return "not prime";
        }
        if (num == 2 || num == 3) {
            return "prime number";
        }
        if ((num * num - 1) % 24 == 0) {
            return "prime";
        } else {
            return "not prime";
        }
    }
}
```

Output

```
Enter number to check if prime or not
2? Does 2 is prime? true prime number  true
3? Does 3 is prime? true prime number  true
4? Does 4 is prime? false not prime  false
5? Does 5 is prime? true prime  true
6? Does 6 is prime? false not prime  false
7? Does 7 is prime? true prime  true
17? Does 17 is prime? true prime  true
21? Does 21 is prime? false not prime  false
131? Does 131 is prime? true prime  true
139? Does 139 is prime? true prime  true
```

83. How to add two numbers without using the plus operator in Java? (solution)

```java
/**
 * Java program to calculate sum of two number without using addition or subtraction
 * operator in Java. This solution, use bitwise and bitshift operator instead of maths operator.
 * @author Javin Paul
 */
public class AddTwoNumbersJava {

    public static void main(String args[]) {

       System.out.println(" Sum of 110 add 200 is : " + add(110, 200));
       System.out.println(" Sum of 0 and 0 is : " + add(0, 0));
       System.out.println(" Sum of -10 and +10 is : " + add(-10, 10));
       System.out.println(" Sum of -10 + 200 is : " + add(-10, 200));
       System.out.println(" Sum of 0 + 200 is : " + add(0, 200));

    }

    /*
     * Adding two number without using + or plus arithmetic operator using
     * recursion in Java. This method uses XOR and AND bitwise operator to
     * calculate sum of two numbers;
     */
    public static int add(int a, int b){
        if(b == 0) return a;
        int sum = a ^ b; //SUM of two integer is A XOR B
        int carry = (a & b) << 1;  //CARRY of two integer is A AND B
        return add(sum, carry);
    }

    /*
     * Adding two integers without any arithmetic operator and using recursion.
     * This solution also uses XOR and AND bitwise and << left shift bitshift
     * operator
     */
    public static int addIterative(int a, int b){
```

```
        while (b != 0){
            int carry = (a & b) ; //CARRY is AND of two bits

            a = a ^b; //SUM of two bits is A XOR B

            b = carry << 1; //shifts carry to 1 bit to calculate sum
        }
        return a;
    }

}
```

```
Output:
Sum of 110 add 200 is : 310
Sum of 0 and 0 is : 0
Sum of -10 and +10 is : 0
Sum of -10 + 200 is : 190
Sum of 0 + 200 is : 200
```

84. How to check if a given number is even/odd without using Arithmetic operator? (solution)

```
package example;

import java.util.Scanner;

/**
 * Java program to find if a number is even or odd in Java or
not. This Java program
 * example demonstrate two ways to check if number is even or
odd or not, first example
 * uses modulus or remainder operator denoted by % to see if
number is even or not
 * and second operator uses Bitwise AND operator to find if
the number is even or odd in Java.
 *
 * @author ocj4u
 */
public class EvenOddTest{

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

```
        //scanner to get input from user
        Scanner console = new Scanner(System.in);

        System.out.printf("Enter any number : ");

        //return the user input as integer
        int number = console.nextInt();

        //if remainder is zero than even number
        if((number %2)==0){
            System.out.printf("number %d is even number %n" ,
number); //%d -decimal %n new line

        } else{
            //number is odd in Java
            System.out.printf("number %d is odd number %n",
number);
        }

        //Finding Even and Odd number using Bitwise AND
operator in Java.

        System.out.printf("Finding number if it's even or odd
using bitwise AND operator %n");

        //For Even numbers
        //XXX0
        //0001 AND
        //0000
        if( (number&1) == 0){
            System.out.printf("number %d is even number %n" ,
number);
        }else{
            System.out.printf("number %d is odd number %n",
number);
        }

    }


}
```

Output:

```
Enter any number: 17
number 17 is an odd number
Finding number if its even or odd using bitwise AND operator
number 17 is an odd number
Enter any number: 12
number 12 is an even number
Finding number if its even or odd using bitwise AND operator
number 12 is an even number
```

85. How to print a given Pyramid structure? (solution)

Sample code in Java to Print the Pyramid Pattern

```
import java.util.Scanner;

/**
 * Simple Java Program to draw a pyramid pattern. We have used both
 * System.out.println() and System.out.print() methods to draw stars(*)
 * in pyramid shape.
 *
 * @author WINDOWS 8
 *
 */
public class PrintPyramidTest {

    public static void main(String args[]) {
        System.out.println("Pyramid pattern of star in Java : ");
        drawPyramidPattern();

        System.out.println("Pyramid of numbers in Java : ");
        drawPyramidOfNumbers();
    }

    /**
     * This method draws a pyramid pattern using asterisk character. You
can
     * replace the asterisk with any other character to draw a pyramid of
that.
     */
    public static void drawPyramidPattern() {
        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 5 - i; j++) {
                System.out.print(" ");
            }
            for (int k = 0; k <= i; k++) {
                System.out.print("* ");
            }
```

```
            System.out.println();
        }
    }


    /**
     * This method draws a pyramid of numbers.
     */
    public static void drawPyramidOfNumbers() {
        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 5 - i; j++) {
                System.out.print(" ");
            }
            for (int k = 0; k <= i; k++) {
                System.out.print(k + " ");
            }
            System.out.println();
        }
    }
}

Output :
Pyramid pattern of star in Java :
     *
    * *
   * * *
  * * * *
 * * * * *
Pyramid of numbers in Java :
     0
    0 1
   0 1 2
  0 1 2 3
 0 1 2 3 4
```

86. How to find the highest repeating word from a given file in Java? (solution)

```
import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Set;
import java.util.StringTokenizer;
import java.util.regex.Pattern;
/**
 * Java program to find count of repeated words in a file.
 *
 * @author
 */
public class Problem {

    public static void main(String args[]) {
        Map<String, Integer> wordMap = buildWordMap("C:/temp/words.txt");
        List<Entry<String, Integer>> list =
sortByValueInDecreasingOrder(wordMap);
        System.out.println("List of repeated word from file and their
count");
        for (Map.Entry<String, Integer> entry : list) {
            if (entry.getValue() > 1) {
                System.out.println(entry.getKey() + " => " +
entry.getValue());
            }
        }
```

```
    }

    public static Map<String, Integer> buildWordMap(String fileName) {
        // Using diamond operator for clean code
        Map<String, Integer> wordMap = new HashMap<>();
        // Using try-with-resource statement for automatic resource
management
        try (FileInputStream fis = new FileInputStream(fileName);
                DataInputStream dis = new DataInputStream(fis);
                BufferedReader br = new BufferedReader(new
InputStreamReader(dis))) {
            // words are separated by whitespace
            Pattern pattern = Pattern.compile("\\s+");
            String line = null;
            while ((line = br.readLine()) != null) {
                // do this if case sensitivity is not required i.e. Java =
java
                line = line.toLowerCase();
                String[] words = pattern.split(line);
                for (String word : words) {
                    if (wordMap.containsKey(word)) {
                        wordMap.put(word, (wordMap.get(word) + 1));
                    } else {
                        wordMap.put(word, 1);
                    }
                }
            }
        } catch (IOException ioex) {
            ioex.printStackTrace();
        }
        return wordMap;
    }

    public static List<Entry<String, Integer>>
sortByValueInDecreasingOrder(Map<String, Integer> wordMap) {
        Set<Entry<String, Integer>> entries = wordMap.entrySet();
        List<Entry<String, Integer>> list = new ArrayList<>(entries);
```

```
        Collections.sort(list, new Comparator<Map.Entry<String,
Integer>>() {
            @Override
            public int compare(Map.Entry<String, Integer> o1,
Map.Entry<String, Integer> o2) {
                return (o2.getValue()).compareTo(o1.getValue());
            }
        });
        return list;
    }
}

Output:
List of repeated word from file and their count
its => 2
of => 2
programming => 2
java => 2
language => 2
```

87. How to reverse given Integer in Java? (solution)

88. How to convert a decimal number to binary in Java? (solution)

```
import java.util.Scanner;

/**
 * Java Program to reverse Integer in Java, number can be negative.
 * Example 1:  x = 123, return 321
 * Example 2:  x = -123, return -321
 *
 * @author Javin Paul
 */

public class ReverseInteger{

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

```
        int input = 5678;
        int output = reverseInteger(5678);
        System.out.println("Input : " + input + " Output : " + output);
    }

    /*
     * Java method to reverse an integer value. there are couple of corner
cases
     * which this method doesn't handle e.g. integer overflow.
     */
    public static int reverseInteger(int number) {
        boolean isNegative = number < 0 ? true : false;
        if(isNegative){
            number = number * -1;
        }
        int reverse = 0;
        int lastDigit = 0;

        while (number >= 1) {
            lastDigit = number % 10; // gives you last digit
            reverse = reverse * 10 + lastDigit;
            number = number / 10; // get rid of last digit
        }

        return isNegative == true? reverse*-1 : reverse;
    }

}

Result :
Input : 5678 Output : 8765
```

89. How to check if a given year is a leap year in Java? (solution)

```
package test;

import java.util.Calendar;

/**
 *
 * Java program to find if a year is leap year or not.

 * A leap year is a year which contains 366 day, which is 1 day more of normal 365
day year.

 * Leap year comes in a interval of 4 years. In Gregorian
 * calendar in leap year February has 29 days which is 1 day more than 28 day in
normal year.
 *
 * @author
 */
public class LeapYearProgram {


    public static void main(String args[]) {

        //Testing some leap and non leap year using Java library code
        System.err.println("Is 2000 a leap year ? : " + isLeapYear(2000));
        System.err.println("Is 2012 a leap year ? : " + isLeapYear(2012));
        System.err.println("Is 1901 a leap year ? : " + isLeapYear(1901));
        System.err.println("Is 1900 a leap year ? : " + isLeapYear(1900));


        //Checking leap year without using library or API and applying logic
        System.err.println("Does 2000 a leap year : " + doesLeapYear(2000));
        System.err.println("Does 2012 a leap year : " + doesLeapYear(2012));
        System.err.println("Does 1901 a leap year : " + doesLeapYear(1901));
        System.err.println("Does 1900 a leap year : " + doesLeapYear(1900));
    }


   /*
     * This method checks whether a year is leap or not by using Java Date
     * and Time API. Calendar class has utility method to return maximum
     * number of days in a year which can be used to check if its
     * greater than 365 or not
     */
   public static boolean isLeapYear(int year){
       Calendar cal = Calendar.getInstance(); //gets Calendar based on local
timezone and locale
       cal.set(Calendar.YEAR, year); //setting the calendar year
       int noOfDays = cal.getActualMaximum(Calendar.DAY_OF_YEAR);

       if(noOfDays > 365){
           return true;
       }

       return false;
   }

   /*
    * This method uses standard logic to check leap year in Java.
    * A year is a leap year if its multiple of 400 or multiple of 4 but not 100
    */
```

```
    public static boolean doesLeapYear(int year){
        return (year%400 == 0) || ((year%100) != 0 && (year%4 == 0));
    }
}

Output:
Is 2000 a leap year ? : true
Is 2012 a leap year ? : true
Is 1901 a leap year ? : false
Is 1900 a leap year ? : false
Does 2000 a leap year : true
Does 2012 a leap year : true
Does 1901 a leap year : false
Does 1900 a leap year : false
```

90. Can you implement a Binary search Algorithm without recursion? (solution)

```
import java.util.Arrays;

import java.util.Scanner;

/**
* Java program to implement Binary Search. We have implemented Iterative
* version of Binary Search Algorithm in Java
*
* @author Javin Paul
*/
public class IterativeBinarySearch {

    public static void main(String args[]) {

        int[] list = new int[]{23, 43, 31, 12};
        int number = 12;
        Arrays.sort(list);
        System.out.printf("Binary Search %d in integer array %s %n",
number,
                          Arrays.toString(list));
        binarySearch(list, 12);

        System.out.printf("Binary Search %d in integer array %s %n", 43,
                          Arrays.toString(list));
        binarySearch(list, 43);
```

```
        list = new int[]{123, 243, 331, 1298};
        number = 331;
        Arrays.sort(list);
        System.out.printf("Binary Search %d in integer array %s %n",
number,
                          Arrays.toString(list));
        binarySearch(list, 331);

        System.out.printf("Binary Search %d in integer array %s %n", 331,
                          Arrays.toString(list));
        binarySearch(list, 1333);

        // Using Core Java API and Collection framework
        // Precondition to the Arrays.binarySearch
        Arrays.sort(list);

        // Search an element
        int index = Arrays.binarySearch(list, 3);

    }

    /**
     * Perform a binary Search in Sorted Array in Java
     *
     * @param input
     * @param number
     * @return location of element in array
     */
    public static void binarySearch(int[] input, int number) {
        int first = 0;
        int last = input.length - 1;
        int middle = (first + last) / 2;

        while (first <= last) {
            if (input[middle] &lt; number) {
                first = middle + 1;
            } else if (input[middle] == number) {
```

```
                System.out.printf(number + " found at location %d %n",
middle);
                break;
            } else {
                last = middle - 1;
            }
            middle = (first + last) / 2;
        }
        if (first &gt; last) {
            System.out.println(number + " is not present in the list.\n");
        }
    }
}
```

```
Output
Binary Search 12 in integer array [12, 23, 31, 43]
12 found at location 0
Binary Search 43 in integer array [12, 23, 31, 43]
43 found at location 3
Binary Search 331 in integer array [123, 243, 331, 1298]
331 found at location 2
Binary Search 331 in integer array [123, 243, 331, 1298]
1333 is not present in the list.
```

91. What is Depth First Search Algorithm for a binary tree? (solution)

```
Algorithm Inorder(tree)
   1. Traverse the left subtree, i.e., call Inorder(left-subtree)
   2. Visit the root.
   3. Traverse the right subtree, i.e., call Inorder(right-subtree)
```

```
// Java program for different tree traversals

/* Class containing left and right child of current
   node and key value*/
class Node {
    int key;
```

```
    Node left, right;

    public Node(int item)
    {
        key = item;
        left = right = null;
    }
}

class BinaryTree {
    // Root of Binary Tree
    Node root;

    BinaryTree()
    {
        root = null;
    }

    /* Given a binary tree, print its nodes in inorder*/
    void printInorder(Node node)
    {
        if (node == null)
            return;

        /* first recur on left child */
        printInorder(node.left);

        /* then print the data of node */
        System.out.print(node.key + " ");

        /* now recur on right child */
        printInorder(node.right);
    }

    // Wrappers over above recursive functions
    void printInorder() { printInorder(root); }

    // Driver method
    public static void main(String[] args)
    {
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(5);

        System.out.println("\nInorder traversal of binary tree is ");
        tree.printInorder();
    }
}
```

Output:

```
Inorder traversal of binary tree is

4 2 5 1 3
```

92. What is the difference between Comparison and Non-Comparison Sorting Algorithms? (answer)

https://javarevisited.blogspot.com/2017/02/difference-between-comparison-quicksort-and-non-comparison-counting-sort-algorithms.html

93. How do implement Sieve of Eratosthenes Algorithms for Prime Number? (solution)

```
import org.junit.Test;
import static org.junit.Assert.*;

/**
 * This class Generates prime numbers upto a given limit using the
 * Sieve of Eratosthenes algorithm. In this algorithm, we create
 * an array of integers starting from 2, then find the first uncrossed
 * integer, and cross out all its multiple. The process is repeated
 * until there are no more multiples in the array.
 */
public class PrimeNumberGenerator {

    private enum Marker{
        CROSSED, UNCROSSED;
    }
    private static Marker[] crossedOut;
    private static int[] primes;

    public static int[] generatePrimeNumbersUpto(int limit){
        if(limit < 2){
            return new int[0];

        }else{
            uncrossIntegerUpto(limit);
            crossOutMultiples();
            putUncrossedIntegersIntoPrimes();
            return primes;
        }
    }

    private static void uncrossIntegerUpto(int limit) {
```

```
    crossedOut = new Marker[limit + 1];
    for(int i = 2; i<crossedOut.length; i++){
        crossedOut[i] = Marker.UNCROSSED;
    }

}

private static void crossOutMultiples() {
    int iterationLimit = determineIterationLimit();
    for (int i = 2; i<= iterationLimit; i++){
        if(notCrossed(i)){
           crossOutMultipleOf(i);
        }
    }

}

private static int determineIterationLimit() {
    // Every multiple in the array has a prime factor
    // that is less than or equal to the square root of
    // the array size, so we don't have to cross out
    // multiples of numbers larger than that root.
   double iterationLimit = Math.sqrt(crossedOut.length);
   return (int) iterationLimit;
}

private static boolean notCrossed(int i) {
    return crossedOut[i] == Marker.UNCROSSED;
}


private static void crossOutMultipleOf(int i) {
    for(int multiple = 2*i;
            multiple < crossedOut.length;
            multiple += i){
        crossedOut[multiple] = Marker.CROSSED;
    }
```

```
    }


    private static void putUncrossedIntegersIntoPrimes() {
        primes = new int[numberOfUncrossedIntegers()];
        for(int j = 0, i = 2; i<crossedOut.length; i++){
            if(notCrossed(i)){
                primes[j++] = i;
            }
        }

    }

    private static int numberOfUncrossedIntegers() {
        int count = 0;
        for(int i = 2; i<crossedOut.length; i++){
            if(notCrossed(i)){
                count++;
            }
        }
        return count;
    }

}
```

s

Unit test to Check Prime Number in Java

And, here are some of the units tests to check whether our program is working properly or not

```
import static org.junit.Assert.*;

import org.junit.Test;

/**
 * Junit test cases to test our Sieve of Eratosthenes algorithm
 * for generating prime numbers upto a given integer.
 * @author WINDOWS 8
 *
```

```
 */
public class PrimeNumberGeneratorTest{

    public PrimeNumberGeneratorTest(){
        System.out.println("Generator prime numbers using"
                + " Sieve of Eratosthenes algorithm");
    }

    @Test
    public void testPrimes(){
        int[] primeUptoZero =
PrimeNumberGenerator.generatePrimeNumbersUpto(0);
        assertEquals(0, primeUptoZero.length);

        int[] primeUptoTwo =
PrimeNumberGenerator.generatePrimeNumbersUpto(2);
        assertEquals(1, primeUptoTwo.length);
        assertEquals(2, primeUptoTwo[0]);

        int[] primeUptoThree =
PrimeNumberGenerator.generatePrimeNumbersUpto(3);
        assertEquals(2, primeUptoThree.length);
        assertEquals(2, primeUptoThree[0]);
        assertEquals(3, primeUptoThree[1]);

        int[] primeUptoHundred =
PrimeNumberGenerator.generatePrimeNumbersUpto(100);
        assertEquals(25, primeUptoHundred.length);
        assertEquals(97, primeUptoHundred[24]);

    }

    @Test
    public void testExhaustive(){
        for(int i = 2; i<700; i++){

verifyPrimeList(PrimeNumberGenerator.generatePrimeNumbersUpto(i));
        }
```

```
    }

    private void verifyPrimeList(int[] listOfPrimes) {
       for(int i = 0; i<listOfPrimes.length; i++){
           verifyPrime(listOfPrimes[i]);
       }

    }

    private void verifyPrime(int number) {
        for (int factor = 2; factor < number; factor++){
            assertTrue(number%factor != 0);
        }

    }
}
```