

JAVA INTERVIEW QUESTIONS



MR KOTIYANA

Top 60 Core Java Interview Questions and Answers & Top 20 Most Asked Java Programming Questions

We are sharing Top 60 Core Java Interview Questions and Answers, Data Structures and Top 20 java interview Programming questions; these questions are frequently asked by the recruiters. Java questions can be asked from any core java topic. So we try our best to provide you the java interview questions and answers for experienced & fresher which should be in your to do list before facing java questions in technical interview.

This Page Was Intentionally Left Blank

Chapter 1 | Data Structures & Interview Questions

Sorting and Searching

Binary Search

Binary search is one of the fundamental algorithms in computer science. In order to explore it, we'll first build up a theoretical backbone, then use that to implement the algorithm properly and avoid those nasty off-by-one errors everyone's been talking about.

Finding a value in a sorted sequence

In its simplest form, binary search is used to quickly find a value in a sorted sequence (consider a sequence an ordinary array for now). We'll call the sought value the *target* value for clarity. Binary search maintains a contiguous subsequence of the starting sequence where the target value is surely located. This is called the *search space*. The search space is initially the entire sequence. At each step, the algorithm compares the median value in the search space to the target value. Based on the comparison and because the sequence is sorted, it can then eliminate half of the search space. By doing this repeatedly, it will eventually be left with a search space consisting of a single element, the target value.

For example, consider the following sequence of integers sorted in ascending order and say we are looking for the number 55:

0	5	13	19	22	41	55	68	72	81	98
---	---	----	----	----	----	----	----	----	----	----

We are interested in the location of the target value in the sequence so we will represent the search space as indices into the sequence. Initially, the search space contains indices 1 through 11. Since the search space is really an interval, it suffices to store just two numbers, the low and high indices. As described above, we now choose the median value, which is the value at index 6 (the midpoint between 1 and 11): this value is 41 and it is smaller than the target value. From this we conclude not only that the element at index 6 is not the target value, but also that no element at indices between 1 and 5 can be the target value, because all elements at these indices are smaller than 41, which is smaller than the target value. This brings the search space down to indices 7 through 11:

55	68	72	81	98
----	----	----	----	----

Proceeding in a similar fashion, we chop off the second half of the search space and are left with:

55	68
----	----

Depending on how we choose the median of an even number of elements we will either find 55 in the next step or chop off 68 to get a search space of only one element. Either way, we conclude that the index where the target value is located is 7.

If the target value was not present in the sequence, binary search would empty the search space entirely. This condition is easy to check and handle. Here is some code to go with the description:

```
binary_search(A, target):
```

```
    lo = 1, hi = size(A)
```

```
    while lo <= hi:
```

```
        mid = lo + (hi-lo)/2
```

```
        if A[mid] == target:
```

```
            return mid
```

```
        else if A[mid] < target:
```

```
            lo = mid+1
```

```
        else:
```

```
            hi = mid-1
```

```
// target was not found
```

Complexity:

Since each comparison binary search uses halves the search space, we can

assert and easily prove that binary search will never use more than (in big-oh notation) $O(\log N)$ comparisons to find the target value.

The logarithm is an awfully slowly growing function. In case you're not aware of just how efficient binary search is, consider looking up a name in a phone book containing a million names. Binary search lets you systematically find any given name using at most 21 comparisons. If you could manage a list containing all the people in the world sorted by name, you could find any person in less than 35 steps.

Program: Java implementation of recursive Binary Search

```
class BinarySearch
{
    // Returns index of x if it is present in arr[l..r], else
    // return -1
    int binarySearch(int arr[], int l, int r, int x)
    {
        if (r >= l)
        {
            int mid = l + (r - l) / 2;

            // If the element is present at the middle itself
            if (arr[mid] == x)
                return mid;

            // If element is smaller than mid, then it can only
            // be present in left subarray
            if (arr[mid] > x)
                return binarySearch(arr, l, mid - 1, x);

            // Else the element can only be present in right
            // subarray
            return binarySearch(arr, mid + 1, r, x);
        }

        // We reach here when element is not present in array
        return -1;
    }

    public static void main(String args[])
    {
        BinarySearch ob = new BinarySearch();
        int arr[] = {2, 3, 4, 10, 40};
    }
}
```

```
int n = arr.length;
int x = 10;
int result = ob.binarySearch(arr,0,n-1,x);
if (result == -1)
    System.out.println("Element not present");
else
    System.out.println("Element found at index "+result);
}
}
```

Output:

Element is present at index 3

Program: Iterative implementation of Binary Search

```
class BinarySearch
{
    // Returns index of x if it is present in arr[], else
    // return -1
    int binarySearch(int arr[], int x)
    {
        int l = 0, r = arr.length - 1;
        while (l <= r)
        {
            int m = l + (r-l)/2;

            // Check if x is present at mid
            if (arr[m] == x)
                return m;

            // If x greater, ignore left half
            if (arr[m] < x)
                l = m + 1;

            // If x is smaller, ignore right half
            else
                r = m - 1;
        }

        // if we reach here, then element was not present
        return -1;
    }

    public static void main(String args[])
    {
        BinarySearch ob = new BinarySearch();
        int arr[] = {2, 3, 4, 10, 40};
        int n = arr.length;
        int x = 10;
    }
}
```

```
int result = ob.binarySearch(arr, x);  
if (result == -1)  
    System.out.println("Element not present");  
else  
    System.out.println("Element found at index "+result);  
}  
}
```

Output:

Element is present at index 3

Bubble Sort

Bubble sort algorithm is known as the simplest sorting algorithm. In bubble sort algorithm, array is traversed from first element to last element. Here, current element is compared with the next element. If current element is greater than the next element, it is swapped.

Algorithm:

We assume **list** is an array of **n** elements. We further assume that **swap** function swaps the values of the given array elements.

```
begin BubbleSort(list)
  for all elements of list
    if list[i] > list[i+1]
      swap(list[i], list[i+1])
    end if
  end for
  return list
end BubbleSort
```

Program: Bubble sort program in java

```
public class BubbleSortExample {
    static void bubbleSort(int[] arr) {
        int n = arr.length;
        int temp = 0;
        for(int i=0; i < n; i++){
            for(int j=1; j < (n-i); j++){
                if(arr[j-1] > arr[j]){
                    //swap elements
                    temp = arr[j-1];
                    arr[j-1] = arr[j];
                    arr[j] = temp;
                }
            }
        }
    }

    public static void main(String[] args) {
        int arr[] = {3,60,35,2,45,320,5};

        System.out.println("Array Before Bubble Sort");

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

        bubbleSort(arr); //sorting array elements using bubble sort

        System.out.println("Array After Bubble Sort");
    }
}
```

```
for(int i=0; i < arr.length; i++){  
    System.out.print(arr[i] + " ");  
}
```

```
}  
}
```

Output:

Array Before Bubble Sort

3 60 35 2 45 320 5

Array After Bubble Sort

2 3 5 35 45 60 320

Insertion sort

Insertion sort is a simple sorting algorithm that works the way we sort playing cards in our hands.

Algorithm

Step 1 – If it is the first element, it is already sorted. return 1;

Step 2 – Pick next element

Step 3 – Compare with all elements in the sorted sub-list

Step 4 – Shift all the elements in the sorted sub-list that is greater than the value to be sorted

Step 5 – Insert the value

Step 6 – Repeat until list is sorted

Program: Java program for implementation of Insertion Sort

```
class InsertionSort
{
    /*Function to sort array using insertion sort*/
    void sort(int arr[])
    {
        int n = arr.length;
        for (int i=1; i<n; ++i)
        {
            int key = arr[i];
            int j = i-1;
            /* Move elements of arr[0..i-1], that are
               greater than key, to one position ahead
               of their current position */

            while (j>=0 && arr[j] > key)
            {
                arr[j+1] = arr[j];
                j = j-1;
            }
            arr[j+1] = key;
        }
        /* A utility function to print array of size n*/
        static void printArray(int arr[])
        {
            int n = arr.length;
            for (int i=0; i<n; ++i)
                System.out.print(arr[i] + " ");

            System.out.println();
        }

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

```
{  
    int arr[] = {12, 11, 13, 5, 6};  
    InsertionSort ob = new InsertionSort();  
    ob.sort(arr);  
    printArray(arr);  
}  
}
```

Output:

5 6 11 12 13

Mergesort

The *Mergesort* algorithm can be used to sort a collection of objects. *Mergesort* is a so called *divide and conquer* algorithm. *Divide and conquer* algorithms divide the original data into smaller sets of data to solve the problem.

Algorithm:

Step 1 – if it is only one element in the list it is already sorted, return.

Step 2 – divide the list recursively into two halves until it can no more be divided.

Step 3 – merge the smaller lists into new list in sorted order.

Program: Java program for Merge Sort

```
class MergeSort
{
    // Merges two subarrays of arr[].
    // First subarray is arr[l..m]
    // Second subarray is arr[m+1..r]

    void merge(int arr[], int l, int m, int r)
    {
        // Find sizes of two subarrays to be merged
        int n1 = m - l + 1;
        int n2 = r - m;

        /* Create temp arrays */
        int L[] = new int [n1];
        int R[] = new int [n2];

        /*Copy data to temp arrays*/
        for (int i=0; i<n1; ++i)
            L[i] = arr[l + i];
        for (int j=0; j<n2; ++j)
            R[j] = arr[m + 1+ j];

        /* Merge the temp arrays */

        // Initial indexes of first and second subarrays
        int i = 0, j = 0;
```

```

// Initial index of merged subarray array
int k = l;
while (i < n1 && j < n2)
{
    if (L[i] <= R[j])
    {
        arr[k] = L[i];
        i++;
    }
    else
    {
        arr[k] = R[j];
        j++;
    }
    k++;
}

/* Copy remaining elements of L[] if any */
while (i < n1)
{
    arr[k] = L[i];
    i++;
    k++;
}

/* Copy remaining elements of R[] if any */
while (j < n2)
{
    arr[k] = R[j];
    j++;
    k++;
}
}

// Main function that sorts arr[l..r] using
// merge()
void sort(int arr[], int l, int r)

```

```

{
    if (l < r)
    {
        // Find the middle point
        int m = (l+r)/2;

        // Sort first and second halves
        sort(arr, l, m);
        sort(arr, m+1, r);

        // Merge the sorted halves
        merge(arr, l, m, r);
    }
}

/* A utility function to print array of size n */
static void printArray(int arr[])
{
    int n = arr.length;
    for (int i=0; i<n; ++i)
        System.out.print(arr[i] + " ");
    System.out.println();
}

public static void main(String args[])
{
    int arr[] = {12, 11, 13, 5, 6, 7};

    System.out.println("Given Array");
    printArray(arr);

    MergeSort ob = new MergeSort();
    ob.sort(arr, 0, arr.length-1);

    System.out.println("\nSorted array");
    printArray(arr);
}

```

```
}
```

Output:

Given array is

12 11 13 5 6 7

Sorted array is

5 6 7 11 12 13

Quicksort

Sort algorithms order the elements of an array according to a predefined order. Quicksort is a divide and conquer algorithm. In a divide and conquer sorting algorithm the original data is separated into two parts "divide" which are individually sorted and "conquered" and then combined

Algorithm:

If the array contains only one element or zero elements then the array is sorted.

If the array contains more than one element then:

- Select an element from the array. This element is called the "pivot element". For example select the element in the middle of the array.
- All elements which are smaller than the pivot element are placed in one array and all elements which are larger are placed in another array.
- Sort both arrays by recursively applying Quicksort to them.
- Combine the arrays.

Quicksort can be implemented to sort "in-place". This means that the sorting takes place in the array and that no additional array needs to be created.

Program: Java program for implementation of QuickSort

```
class QuickSort
{
    /* This function takes last element as pivot,
    places the pivot element at its correct
    position in sorted array, and places all
    smaller (smaller than pivot) to left of
    pivot and all greater elements to right
```

of pivot */

```
int partition(int arr[], int low, int high)
{
    int pivot = arr[high];
    int i = (low-1); // index of smaller element
    for (int j=low; j<=high-1; j++)
    {
        // If current element is smaller than or
        // equal to pivot
        if (arr[j] <= pivot)
        {
            i++;
            // swap arr[i] and arr[j]
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
    // swap arr[i+1] and arr[high] (or pivot)
    int temp = arr[i+1];
    arr[i+1] = arr[high];
    arr[high] = temp;

    return i+1;
}
```

```

/* The main function that implements QuickSort()
arr[] --> Array to be sorted,
low  --> Starting index,
high --> Ending index
*/

void sort(int arr[], int low, int high)
{
    if (low < high)
    {
        /* pi is partitioning index, arr[pi] is
        now at right place */
        int pi = partition(arr, low, high);

        // Recursively sort elements before
        // partition and after partition
        sort(arr, low, pi-1);
        sort(arr, pi+1, high);
    }
}

/* A utility function to print array of size n */
static void printArray(int arr[])
{
    int n = arr.length;
    for (int i=0; i<n; ++i)

```

```
        System.out.print(arr[i]+" ");
    System.out.println();
}
public static void main(String args[])
{
    int arr[] = {10, 7, 8, 9, 1, 5};
    int n = arr.length;
    QuickSort ob = new QuickSort();
    ob.sort(arr, 0, n-1);
    System.out.println("sorted array");
    printArray(arr);
}
}
```

Output:

Sorted array:

1 5 7 8 9 10

Selection sort

Selection sort is a simple sorting algorithm. This sorting algorithm is an in-place comparison-based algorithm in which the list is divided into two parts, the sorted part at the left end and the unsorted part at the right end. Initially, the sorted part is empty and the unsorted part is the entire list.

The smallest element is selected from the unsorted array and swapped with the leftmost element, and that element becomes a part of the sorted array. This process continues moving unsorted array boundary by one element to the right.

This algorithm is not suitable for large data sets as its average and worst case complexities are of $O(n^2)$, where **n** is the number of items.

Algorithm:

Step 1 – Set MIN to location 0

Step 2 – Search the minimum element in the list

Step 3 – Swap with value at location MIN

Step 4 – Increment MIN to point to next element

Step 5 – Repeat until list is sorted

Program: Selection Sort in Java

```
public class SelectionSortExample {  
    public static void selectionSort(int[] arr){  
        for (int i = 0; i < arr.length - 1; i++)  
        {  
            int index = i;  
            for (int j = i + 1; j < arr.length; j++){  
                if (arr[j] < arr[index]){  
                    index = j; //searching for lowest index  
                }  
            }  
            int smallerNumber = arr[index];  
            arr[index] = arr[i];  
            arr[i] = smallerNumber;  
        }  
    }  
}
```

```
public static void main(String a[]){  
    int[] arr1 = {9,14,3,2,43,11,58,22};  
    System.out.println("Before Selection Sort");  
    for(int i:arr1){  
        System.out.print(i+" ");  
    }  
    System.out.println();
```

```
//sorting array using selection sort  
selectionSort(arr1);
```

```
System.out.println("After Selection Sort");  
for(int i:arr1){  
    System.out.print(i+" ");  
}
```

```
}  
}
```

Output:

Before Selection Sort

9 14 3 2 43 11 58 22

After Selection Sort

2 3 9 11 14 22 43 58

Linked List

A linked list is a sequence of data structures, which are connected together via links.

Linked List is a sequence of links which contains items. Each link contains a connection to another link. Linked list is the second most-used data structure after array. Following are the important terms to understand the concept of Linked List.

- **Link** – Each link of a linked list can store a data called an element.
- **Next** – Each link of a linked list contains a link to the next link called Next.
- **LinkedList** – A Linked List contains the connection link to the first link called First.

Linked List Representation

Linked list can be visualized as a chain of nodes, where every node points to the next node.



As per the above illustration, following are the important points to be considered.

- Linked List contains a link element called first.
- Each link carries a data field(s) and a link field called next.
- Each link is linked with its next link using its next link.
- Last link carries a link as null to mark the end of the list.

Types of Linked List:

Following are the various types of linked list.

- **Simple Linked List** – Item navigation is forward only.
- **Doubly Linked List** – Items can be navigated forward and backward.
- **Circular Linked List** – Last item contains link of the first element as next and the first element has a link to the last element as previous.

Basic Operations:

Following are the basic operations supported by a list.

- **Insertion** – Adds an element at the beginning of the list.
- **Deletion** – Deletes an element at the beginning of the list.
- **Display** – Displays the complete list.
- **Search** – Searches an element using the given key.
- **Delete** – Deletes an element using the given key.

Insertion Operation:

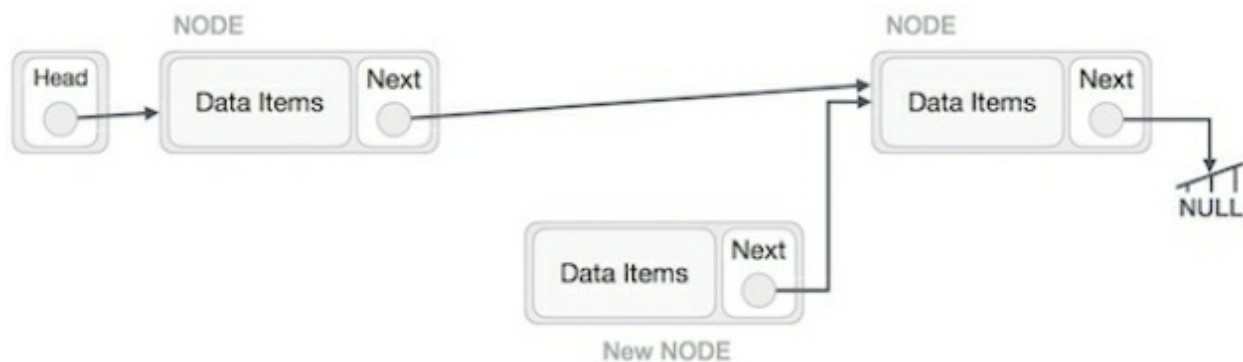
Adding a new node in linked list is a more than one step activity. We shall learn this with diagrams here. First, create a node using the same structure and find the location where it has to be inserted.



Imagine that we are inserting a node **B** (NewNode), between **A** (LeftNode) and **C** (RightNode). Then point B.next to C –

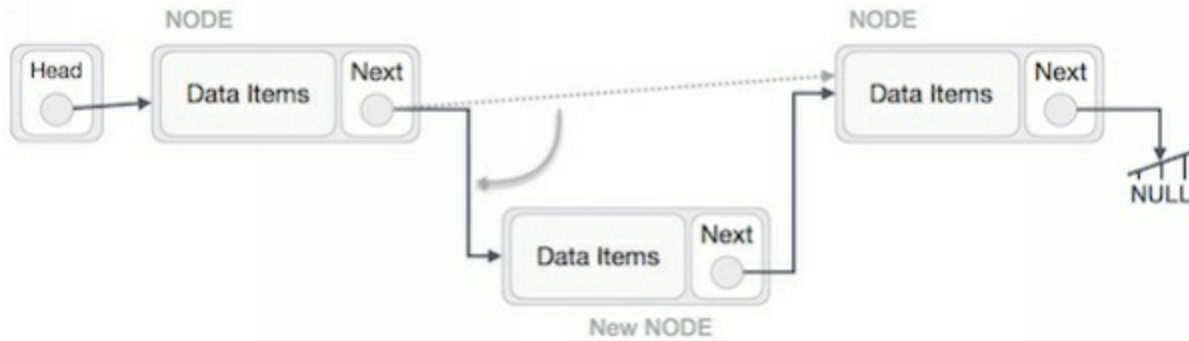
```
NewNode.next -> RightNode;
```

It should look like this –



Now, the next node at the left should point to the new node.

```
LeftNode.next -> NewNode;
```



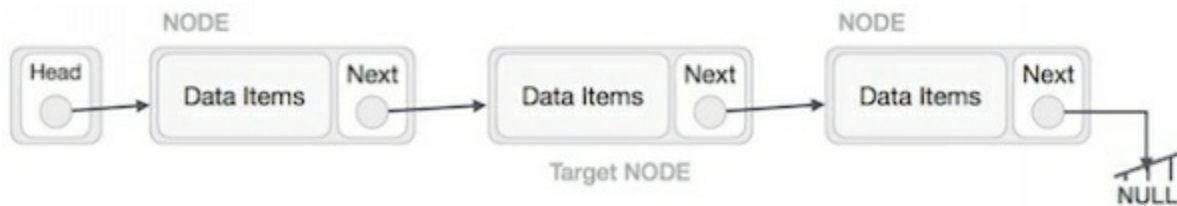
This will put the new node in the middle of the two. The new list should look like this –



Similar steps should be taken if the node is being inserted at the beginning of the list. While inserting it at the end, the second last node of the list should point to the new node and the new node will point to NULL.

Deletion Operation:

Deletion is also a more than one step process. We shall learn with pictorial representation. First, locate the target node to be removed, by using searching algorithms.



The left (previous) node of the target node now should point to the next node of the target node –

```
LeftNode.next -> TargetNode.next;
```

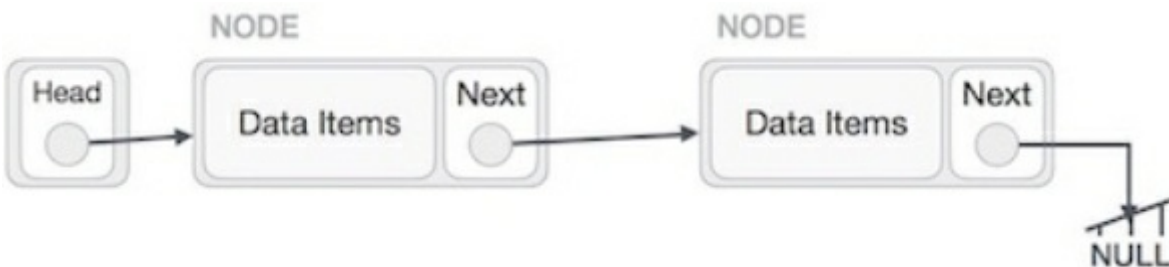


This will remove the link that was pointing to the target node. Now, using the following code, we will remove what the target node is pointing at.

```
TargetNode.next -> NULL;
```

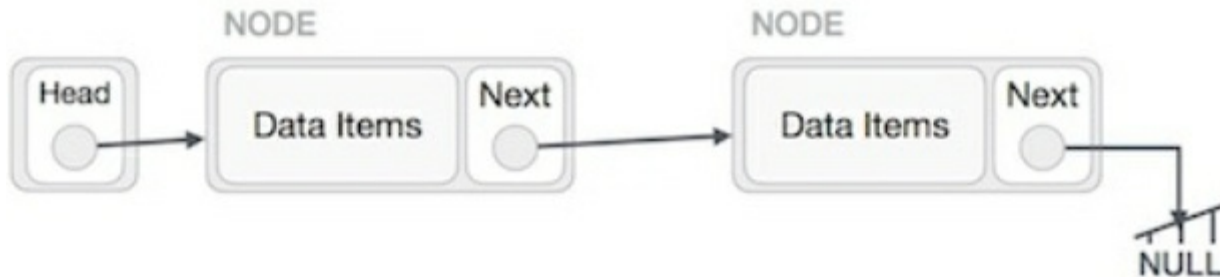


We need to use the deleted node. We can keep that in memory otherwise we can simply deallocate memory and wipe off the target node completely.

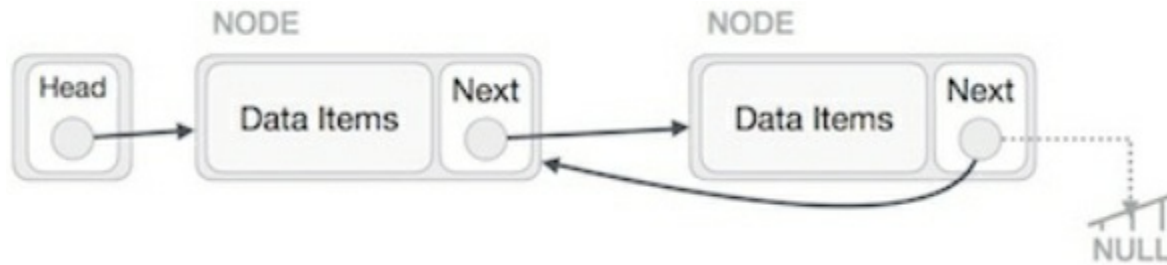


Reverse Operation:

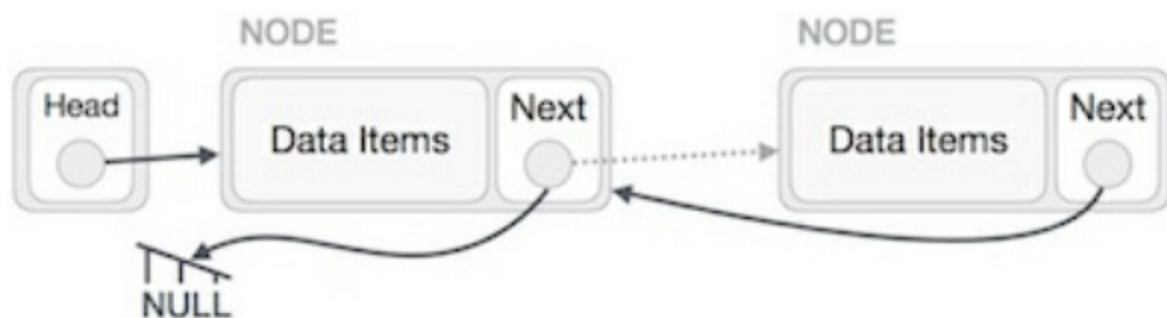
This operation is a thorough one. We need to make the last node to be pointed by the head node and reverse the whole linked list.



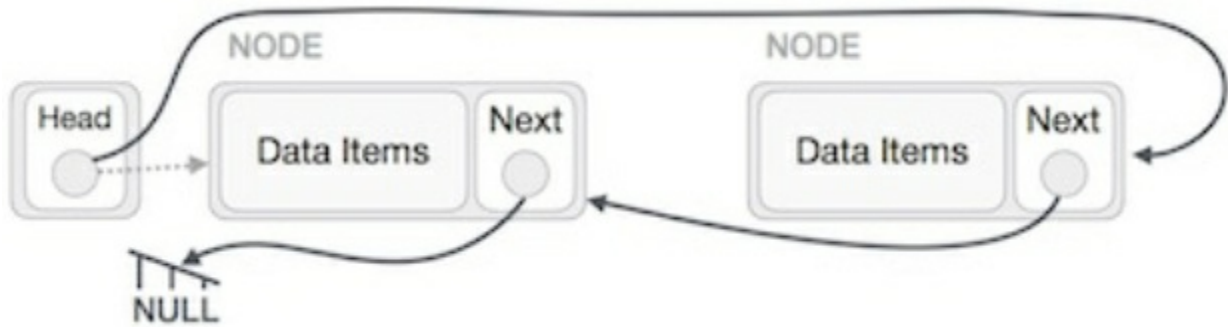
First, we traverse to the end of the list. It should be pointing to NULL. Now, we shall make it point to its previous node –



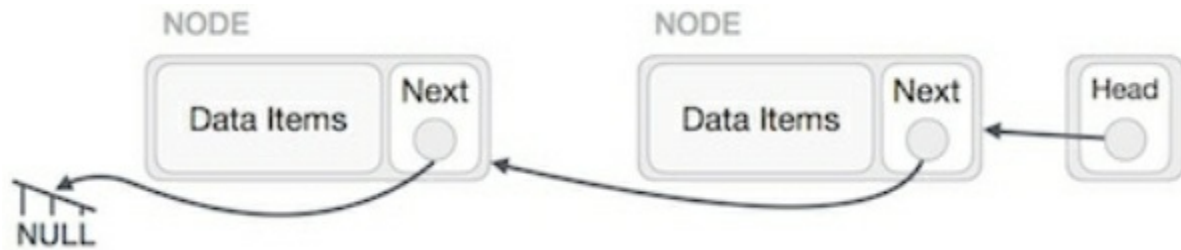
We have to make sure that the last node is not the lost node. So we'll have some temp node, which looks like the head node pointing to the last node. Now, we shall make all left side nodes point to their previous nodes one by one.



Except the node (first node) pointed by the head node, all nodes should point to their predecessor, making them their new successor. The first node will point to NULL.



We'll make the head node point to the new first node by using the temp node.



The linked list is now reversed.

Program: Given a linked list which is sorted, how will you insert in sorted way?

Algorithm:

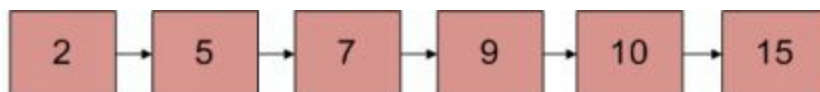
Let input linked list is sorted in increasing order.

- 1) If Linked list is empty then make the node as head and return it.
- 2) If value of the node to be inserted is smaller than value of head node then insert the node at start and make it head.
- 3) In a loop, find the appropriate node after which the input node (let 9) is to be inserted. To find the appropriate node start from head, keep moving until you reach a node GN (10 in the below diagram) who's value is greater than the input node. The node just before GN is the appropriate node (7).
- 4) Insert the node (9) after the appropriate node (7) found in step 3.

Initial Linked List



Linked List after insertion of 9



```

class LinkedList
{
    Node head; // head of list

    /* Linked list Node*/
    class Node
    {
        int data;
        Node next;
        Node(int d) {data = d; next = null; }
    }

    /* function to insert a new_node in a list. */
    void sortedInsert(Node new_node)
    {
        Node current;

        /* Special case for head node */
        if (head == null || head.data >= new_node.data)
        {
            new_node.next = head;
            head = new_node;
        }
        else {

            /* Locate the node before point of insertion. */
            current = head;

            while (current.next != null &&
                current.next.data < new_node.data)
                current = current.next;

            new_node.next = current.next;
            current.next = new_node;
        }
    }
}

```

```
}
```

```
/*Utility functions*/
```

```
/* Function to create a node */
```

```
Node newNode(int data)
{
    Node x = new Node(data);
    return x;
}
```

```
/* Function to print linked list */
```

```
void printList()
{
    Node temp = head;
    while (temp != null)
    {
        System.out.print(temp.data+" ");
        temp = temp.next;
    }
}
```

```
/* Driver function to test above methods */
```

```
public static void main(String args[])
{
    LinkedList llist = new LinkedList();
    Node new_node;
    new_node = llist.newNode(5);
    llist.sortedInsert(new_node);
    new_node = llist.newNode(10);
    llist.sortedInsert(new_node);
    new_node = llist.newNode(7);
    llist.sortedInsert(new_node);
    new_node = llist.newNode(3);
    llist.sortedInsert(new_node);
    new_node = llist.newNode(1);
    llist.sortedInsert(new_node);
}
```

```
        new_node = llist.newNode(9);  
        llist.sortedInsert(new_node);  
        System.out.println("Created Linked List");  
        llist.printList();  
    }  
}
```

Output:

Created Linked List

1 3 5 7 9 10

Program: Delete a given node in Linked List under given constraints

Given a Singly Linked List, write a function to delete a given node. Your function must follow following constraints:

- 1) It must accept pointer to the start node as first parameter and node to be deleted as second parameter i.e., pointer to head node is not global.
- 2) It should not return pointer to the head node.
- 3) It should not accept pointer to pointer to head node.

You may assume that the Linked List never becomes empty.

Let the function name be deleteNode(). In a straightforward implementation, the function needs to modify head pointer when the node to be deleted is first node.

```
class LinkedList {  
  
    static Node head;  
  
    static class Node {  
  
        int data;  
        Node next;  
  
        Node(int d) {  
            data = d;  
            next = null;  
        }  
    }  
}  
  
void deleteNode(Node node, Node n) {  
  
    // When node to be deleted is head node  
    if (node == n) {  
        if (node.next == null) {  
            System.out.println("There is only one node. The list "  
                                + "can't be made empty ");  
            return;  
        }  
    }  
}
```

```

    }

    /* Copy the data of next node to head */
    node.data = node.next.data;

    // store address of next node
    n = node.next;

    // Remove the link of next node
    node.next = node.next.next;

    // free memory
    System.gc();

    return;
}

// When not first node, follow the normal deletion process
// find the previous node
Node prev = node;
while (prev.next != null && prev.next != n) {
    prev = prev.next;
}

// Check if node really exists in Linked List
if (prev.next == null) {
    System.out.println("Given node is not present in Linked List");
    return;
}

// Remove node from Linked List
prev.next = prev.next.next;
// Free memory
System.gc();
return;
}

/* Utility function to print a linked list */
void printList(Node head) {

```

```

while (head != null) {
    System.out.print(head.data + " ");
    head = head.next;
}
System.out.println("");
}

public static void main(String[] args) {
    LinkedList list = new LinkedList();
    list.head = new Node(12);
    list.head.next = new Node(15);
    list.head.next.next = new Node(10);
    list.head.next.next.next = new Node(11);
    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(2);
    list.head.next.next.next.next.next.next.next = new Node(3);
    System.out.println("Given Linked List :");
    list.printList(head);
    System.out.println("");
    // Let us delete the node with value 10
    System.out.println("Deleting node : " + head.next.next.data);
    list.deleteNode(head, head.next.next);
    System.out.println("Modified Linked list :");
    list.printList(head);
    System.out.println("");
    // Lets delete the first node
    System.out.println("Deleting first Node");
    list.deleteNode(head, head);
    System.out.println("Modified Linked List");
    list.printList(head);

}
}

```

Output:

Given Linked List: 12 15 10 11 5 6 2 3

Deleting node 10:

Modified Linked List: 12 15 11 5 6 2 3

Deleting first node

Modified Linked List: 15 11 5 6 2 3

Top 60 Most Asked Core Java Interview Questions!

1. What is the most important feature of Java?

Java is a platform independent language.

2. What do you mean by platform independence?

Platform independence means that we can write and compile the java code in one platform (For example: Windows) and can execute the class in any other supported platform (Linux, Solaris, etc).

3. What is a JVM?

JVM is Java Virtual Machine which is a run time environment for the compiled java class files.

4. Are JVM's platforms independent?

JVM's are not platform independent. JVM's are platform specific run time implementation provided by the vendor.

5. What is the difference between a JDK and a JVM?

JDK is Java Development Kit which is for development purpose and it includes execution environment also. But JVM is purely a run time environment and hence you will not be able to compile your source files using a JVM.

6. What is a pointer and does Java support pointers?

Pointer is a reference handle to a memory location. Improper handling of pointers leads to memory leaks and reliability issues hence Java doesn't support the usage of pointers.

7. What is the base class of all classes?

`java.lang.Object`

8. Does Java support multiple inheritance?

Java doesn't support multiple inheritance.

9. Is Java a pure object oriented language?

Java uses primitive data types and hence is not a pure object oriented language.

10. Are arrays primitive data types?

In Java, Arrays are objects.

11. What is difference between Path and Classpath?

Path and Classpath are operating system level environment variables. Path is used to define where the system can find the executable (.exe) files and classpath is used to specify the location .class files.

12. What are local variables?

Local variables are those which are declared within a block of code like methods. Local variables should be initialized before accessing them.

13. What are instance variables?

Instance variables are those which are defined at the class level. Instance variables need not be initialized before using them as they are automatically initialized to their default values.

14. How to define a constant variable in Java?

The variable should be declared as `static` and `final`. So only one copy of the variable exists for all instances of the class and the

value can't be changed also.

`static final int MAX_LENGTH = 50;` is an example for constant.

15. Should a `main()` method be compulsorily declared in all java classes?

No not required. `main()` method should be defined only if the source class is a java application.

16. What is the return type of the `main()` method?

`Main()` method doesn't return anything hence declared `void`.

17. Why is the `main()` method declared static?

`main()` method is called by the JVM even before the instantiation of the class hence it is declared as `static`.

18. What is the argument of `main()` method?

`main()` method accepts an array of String object as an argument.

19. Can a `main()` method be overloaded?

Yes. You can have any number of `main()` methods with different method signature and implementation in the class.

20. Can a `main()` method be declared final?

Yes. Any inheriting class will not be able to have it's own default `main()` method.

21. Does the order of public and static declaration matter in `main()` method?

No. It doesn't matter but `void` should always come before `main()`.

22. Can a source file contain more than one class declaration?

Yes a single source file can contain any number of Class declarations but only one of the class can be declared as `public`.

23. What is a package?

Package is a collection of related classes and interfaces. package declaration should be first statement in a java class.

24. Which package is imported by default?

`java.lang` package is imported by default even without a package declaration.

25. Can a class declared as private be accessed outside its package?

Not possible.

26. Can a class be declared as protected?

The protected access modifier cannot be applied to class and interfaces. Methods, fields can be declared `protected`, however methods and fields in a interface cannot be declared `protected`.

27. What is the access scope of a protected method?

A `protected` method can be accessed by the classes within the same package or by the subclasses of the class in any package.

28. What is the purpose of declaring a variable as final?

A `final` variable's value can't be changed. `final` variables should be initialized before using them.

29. What is the impact of declaring a method as final?

A method declared as `final` can't be overridden. A sub-class can't have the same method signature with a different implementation.

30. I don't want my class to be inherited by any other class. What should i do?

You should declare your class as `final`. But you can't define your class

as **final**, if it is an **abstract** class. A class declared as **final** can't be extended by any other class.

31. Can you give few examples of final classes defined in Java API?

`java.lang.String`, `java.lang.Math` are **final** classes.

32. How is **final** different from **finally** and **finalize()**?

final is a modifier which can be applied to a class or a method or a variable. **final** class can't be inherited, **final** method can't be overridden and **final** variable can't be changed.

Finally is an exception handling code section which gets executed whether an exception is raised or not by the try block code segment. **finalize()** is a method of `Object` class which will be executed by the JVM just before garbage collecting object to give a final chance for resource releasing activity.

33. Can a class be declared as static?

We cannot declare top level class as static, but only inner class can be declared static.

```
public class Test
{
    static class InnerClass
    {
        public static void InnerMethod()
        { System.out.println("Static Inner Class!"); }
    }
    public static void main(String args[])
    {
        Test.InnerClass.InnerMethod();
    }
}
//output: Static Inner Class!
```

34. When will you define a method as static?

When a method needs to be accessed even before the creation of the object of the class then we should declare the method as `static`.

35. What are the restrictions imposed on a static method or a static block of code?

A static method should not refer to instance variables without creating an instance and cannot use "this" operator to refer the instance.

36. I want to print "Hello" even before main() is executed. How will you achieve that?

Print the statement inside a static block of code. Static blocks get executed when the class gets loaded into the memory and even before the creation of an object. Hence it will be executed before the `main()` method. And it will be executed only once.

37. What is the importance of static variable?

static variables are class level variables where all objects of the class refer to the same variable. If one object changes the value then the change gets reflected in all the objects.

38. Can we declare a static variable inside a method?

Static variables are class level variables and they can't be declared inside a method. If declared, the class will not compile.

39. What is an Abstract Class and what is its purpose?

A Class which doesn't provide complete implementation is defined as an abstract class. Abstract classes enforce abstraction.

40. Can a abstract class be declared final?

Not possible. An abstract class without being inherited is of no use and hence will result in compile time error.

41. What is use of a abstract variable?

Variables can't be declared as abstract. Only classes and methods can be declared as **abstract**.

42. Can you create an object of an abstract class?

Not possible. Abstract classes can't be instantiated.

43. Can a abstract class be defined without any abstract methods?

Yes it's possible. This is basically to avoid instance creation of the class.

44. Class C implements Interface I containing method m1 and m2 declarations. Class C has provided implementation for method m2. Can i create an object of Class C?

No not possible. **Class C** should provide implementation for all the methods in the **Interface I**. Since **Class C** didn't provide implementation for **m1** method, it has to be declared as **abstract**. Abstract classes can't be instantiated.

45. Can a method inside a Interface be declared as final?

No not possible. Doing so will result in compilation error. **public** and **abstract** are the only applicable modifiers for method declaration in an **interface**.

46. Can an Interface implement another Interface?

Interfaces doesn't provide implementation hence a interface cannot implement another interface.

47. Can an Interface extend another Interface?

Yes an Interface can inherit another Interface, for that matter an Interface can extend more than one Interface.

48. Can a Class extend more than one Class?
Not possible. A Class can extend only one class but can implement any number of Interfaces.

49. Why is an Interface be able to extend more than one Interface but a Class can't extend more than one Class?
Basically Java doesn't allow multiple inheritance, so a Class is restricted to extend only one Class. But an Interface is a pure abstraction model and doesn't have inheritance hierarchy like classes (do remember that the base class of all classes is Object). So an Interface is allowed to extend more than one Interface.

50. Can an Interface be final?
Not possible. Doing so will result in compilation error.

51. Can a class be defined inside an Interface?
Yes it's possible.

52. Can an Interface be defined inside a class?
Yes it's possible.

53. What is a Marker Interface?
An Interface which doesn't have any declaration inside but still enforces a mechanism.

54. Which object oriented Concept is achieved by using overloading and overriding?
Polymorphism.

55. Why does Java not support operator overloading?
Operator overloading makes the code very difficult to read and maintain. To maintain code simplicity, Java doesn't support operator overloading.

56. Can we define private and protected modifiers for variables in interfaces?
No.

57. What is Externalizable?

Externalizable is an Interface that extends Serializable Interface. And sends data into Streams in Compressed Format. It has two methods, `writeExternal(ObjectOutput out)` and `readExternal(ObjectInput in)`

58. What modifiers are allowed for methods in an Interface?

Only `public` and `abstract` modifiers are allowed for methods in interfaces.

59. What is a local, member and a class variable?

Variables declared within a method are "local" variables.

Variables declared within the class i.e not within any methods are "member" variables (global variables).

Variables declared within the class i.e not within any methods and are defined as "static" are class variables.

60. What is an abstract method?

An abstract method is a method whose implementation is deferred to a subclass.

20 Most Asked Programming Questions!

Q.1) PROGRAM TO CHECK UNIQUE NUMBER IN JAVA

```
import java.util.*;
import java.io.*;

public class IsUnique
{

    public static boolean isUniqueUsingHash(String word)
    {
        char[] chars = word.toCharArray();

        Set<Character> set = new HashSet<Character>();
        for (char c : chars)

            if (set.contains(c))
                return false;
            else
                set.add(c);
            return true;

    }

    public static boolean isUniqueUsingSort(String word)
    {
        char[] chars = word.toCharArray();

        if (chars.length <= 1)
            return true;
        Arrays.sort(chars);

        char temp = chars[0];

        for (int i = 1; i < chars.length; i++)
        {
```

```
    if (chars[i] == temp)
        return false;
    temp = chars[i];
}

return true;

}

public static void main(String[] args) throws IOException
{
    System.out.println(isUniqueUsingHash("Word") ? "Unique" : "Not Unique");
    System.out.println(isUniqueUsingSort("Nootunique") ? "Unique" : "Not Unique");
}

}
```

Output:

```
Unique
Not Unique
```

Q.2) PROGRAM TO FIND PERMUTATION OF A STRING

```
import java.util.*;
import java.io.*;

public class CheckPermutations
{

    public static boolean isPermutation(String s1, String s2)
    {
        char[] a = s1.toCharArray();

        char[] b =
s2.toCharArray();

        Arrays.sort(a);
        Arrays.sort(b);
        if (a.length != b.length)
            return false;
            for (int i = 0; i <
a.length; i++)
            {

                if (a[i] != b[i]) return false;

            }

            return true;

        }

    public static void main(String[] args)
    {
        System.out.println(isPermutation("abc", "cba") ? "It is a permutation":
        "It is not a permutation");

        System.out.println(isPermutation("test", "estt") ? "It is a permutation": "It is not a
        permutation");
```

```
System.out.println(isPermutation("testt", "estt") ? "It is a permutation": "It is not a permutation");
```

```
}
```

```
}
```

Output:

```
It is a permutation
```

```
It is a permutation
```

```
It is not a permutation
```

Q.3) PROGRAM TO PUT HTML LINKS AROUND URLS STRINGS

```
import java.util.*;
import java.io.*;

public class URLify
{
    public static char[] URLify(char[] chars, int len)
    {
        int spaces = countSpaces(chars, len);

        int end = len - 1 + spaces * 2;
        for (int i = len - 1; i >= 0; i--)
        {
            if (chars[i] == ' ')
            {
                chars[end - 2] = '%';
                chars[end - 1] = '2';
                chars[end] = '0';
                end -= 3;
            }
            else
            {
                chars[end] = chars[i]; end--;
            }
        }
        return chars;
    }
}
```

```
static int countSpaces(char[] chars, int len)
{
    int count = 0;

    for (int i = 0; i < len; i++)
        if (chars[i] == ' ') count++;

    return count;
}

public static void main(String[] args) throws IOException
{
    char[] chars = "Mr John Smith ".toCharArray(); System.out.println(URLify(chars,
    13));
}
}
```

Output:

Mr%20John%20Smith

Q.4) PROGRAM TO CHECK PALINDROME PERMUTATIONS OF A STRING

```
import java.util.*;

public class PalindromePermutation {

    public static boolean permuteHash(String str)
    {
        Map<Character, Integer> map = new HashMap<Character, Integer>();
        for (int i = 0; i < str.length(); i++) {

            Character c = Character.toLowerCase(str.charAt(i));

            if (!Character.isLetter(c))
                continue;

            if (map.containsKey(c))
                map.put(c, map.get(c) + 1);
            else
                map.put(c, 1);
        }

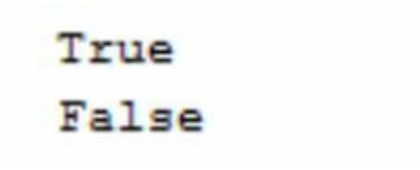
        int odd = 0;

        for (Character key : map.keySet())
            if (map.get(key) % 2 != 0)
                odd++;

        if (odd > 1)
            return false;
```

```
else  
return true;  
}  
public static void main(String[] args)  
{  
System.out.println(permuteHash("Tact Coa") ? "True" : "False");  
System.out.println(permuteHash("test") ? "True" : "False");  
}
```

Output:



```
True  
False
```

Q.5) PROGRAM TO COMPRESS STRING

```
public class StringCompression
{
    public String compress(String input)
    {
        char[] cs = input.toCharArray();
        char temp = cs[0];
        int i = 0, j = 0, count = 0, len = cs.length;
        while(j < len)
        {
            cs[i++] = temp;
            while (j < len && temp == cs[j])
            {
                j++;
                count++;
            }
            if(j < len)
                temp = cs[j++];
            cs[i++] = String.valueOf(count).charAt(0);
            count = 1;
        }
        return new String(cs, 0, i);
    }
}
```

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

StringCompression compression = new StringCompression();
System.out.println(compression.compress("aabbcc"));
}
}
```

Output:

a2b3c3

Q.6) PROGRAM TO ROTATE MATRIX

```
import java.util.*;

public class RotateMatrix {

    public static void rotate(int[][] matrix)
    {
        int n = matrix.length;

        for (int layer = 0; layer < n / 2; layer++)
        {
            int first = layer;
            int last = n - 1 - layer;
            for (int i = first; i < last; i++)
            {
                int offset = i - first;
                int top = matrix[first][i];

                // left -> top
                matrix[first][i] = matrix[last-offset][first];
                // bottom -> left
                matrix[last-offset][first] = matrix[last][last-offset];
                // right -> bottom matrix[last][last-offset] = matrix[i][last];
                // top -> right matrix[i][last] = top;
            }
        }
    }
}
```

```
public static void main(String[] args)
{
    int[][] arr = new int[][]
    {

        {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}

    };

    rotate(arr);

    for (int[] a : arr)
    System.out.println(Arrays.toString(a));
}
```

Output:

```
[21, 16, 11, 6, 1]
[22, 17, 12, 7, 2]
[23, 18, 13, 8, 3]
[24, 19, 14, 9, 4]
[25, 20, 15, 10, 5]
```

Q.7) PROGRAM TO CONVERT ALL 1 INTO ZERO MATRIX.

```
import java.util.*;

public class ZeroMatrix {

    public static void zero(int[][] matrix) {
        int m = matrix.length;

        int n = matrix[0].length;
        boolean[] row = new boolean[m];
        boolean[] col = new boolean[n];

        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                if (matrix[i][j] == 0) {
                    row[i] = true;
                    col[j] = true;
                }
            }
        }

        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                if (row[i] == true || col[j] == true)
                    matrix[i][j] = 0;
            }
        }
        return;
    }

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

```
int[][] matrix = new int[][]
{
    {0, 1, 1, 1, 0},
    {1, 0, 1, 0, 1},
    {1, 1, 1, 1, 1},
    {1, 0, 1, 1, 1},
    {1, 1, 1, 1, 1}
};

for (int i = 0; i < matrix.length; i++)
    System.out.println(Arrays.toString(matrix[i]));
zero(matrix);
System.out.println();

for (int i = 0; i < matrix.length; i++)
    System.out.println(Arrays.toString(matrix[i]));
}
}
```

Output:

```
[0, 0, 0, 0, 0]
[0, 0, 0, 0, 0]
[0, 0, 1, 0, 0]
[0, 0, 0, 0, 0]
[0, 0, 1, 0, 0]
```


Q.8) PROGRAM TO ROTATE STRING.

```
import java.util.*;

public class StringRotation {

    public static boolean isRotation(String s1, String s2)
    {
        if (s1.length() != s2.length())
            return false;
        String s3 = s1 + s1;
        return s3.contains(s2);
    }

    public static void main(String[] args)
    {
        System.out.println(isRotation("waterbottle", "erbottlewat") ? "True" :
            "False");
        System.out.println(isRotation("waterbottl", "erbottlewat") ? "True" : "False");
        System.out.println(isRotation("pandeesh", "deeshpand") ? "True" : "False");
    }
}
```

Output:

```
True
False
False
```

Q.9) PROGRAM TO ADD TWO NUMBERS WITHOUT USING PLUS ('+') SIGN

```
public class AddWithoutPlus
{
    public static int add(int a, int b)
    {
        if (b > a)
        {
            int temp = b;
            b = a;
            a = temp;
        }

        int carry = 0;
        while (b != 0)
        {
            carry = a & b;
            a = a ^ b;
            b = carry << 1;
        }
        return a;
    }

    public static int recursiveAdd(int a, int b)
    {
        if (b == 0)

        return a;
        else

        return recursiveAdd(a ^ b, (a & b) << 1);
    }

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

```
{  
int a = 12;  
  
int b = 34;  
System.out.println(add(a, b));  
  
System.out.println(recursiveAdd(b, a));  
  
}
```

Output:

46

46

Q.10) PROGRAM TO REMOVE DUPLICATES CHARACTER FROM A STRING

```
import java.util.*;

public class RemoveDups {
    public static class Node {
        Node next;

        char val;

        public Node(char val)
        {
            this.val = val;
        }

        public String toString() {

            StringBuilder sb = new StringBuilder();
            Node temp = this;

            while (temp != null)
            {
                sb.append(temp.val);
                temp = temp.next;
            }

            return sb.toString();

        }
    }

    public static void removeDups(Node node)
    {
```

```
Set<Character> set = new HashSet<Character>();  
set.add(node.val);
```

```
Node prev = node;  
Node temp = node.next;
```

```
while (temp != null)  
{  
    if (set.contains(temp.val))  
    {  
        prev.next = temp.next;  
    }
```

```
    else  
    {  
        set.add(temp.val);
```

```
        prev = temp;  
    }
```

```
    temp = temp.next;
```

```
}
```

```
}
```

```
public static void main(String[] args)  
{  
    Node a = new Node('F');  
    Node b = new Node('O');  
    Node c = new Node('L');  
    Node d = new Node('L');
```

```
Node e = new Node('O');  
Node f = new Node('W');  
Node g = new Node(' ');  
Node h = new Node('U');  
Node i = new Node('P');
```

```
a.next = b;  
b.next = c;  
c.next = d;  
d.next = e;  
e.next = f;  
f.next = g;  
g.next = h;  
h.next = i;
```

```
System.out.println(a);  
removeDupes(a);  
System.out.println(a);  
}  
}
```

Output:

```
FOLLOW UP  
FOLW UP
```

Q.11) PROGRAM TO RETURN A CHARACTER FROM THE STRING.

```
import java.util.*;

public class ReturnKth {
    public static class Node {
        Node next;
        char val;
        public Node(char val) {
            this.val = val;
        }
        public String toString() {
            StringBuilder sb = new StringBuilder();
            Node temp = this;
            while (temp != null) {
                sb.append(temp.val);
                temp = temp.next;
            }
            return sb.toString();
        }
    }
    public static Node returnKth(Node node, int k) {
        k--;
        Node first = node;
        Node last = node;
        for (int i = 0; i < k; i++)
            last = last.next;
        while (last.next != null) {
            last = last.next;
            first = first.next;
        }

        return first;
    }
}
```

```
public static void main(String[] args) {  
    Node a = new Node('a');  
    Node b = new Node('b');  
    Node c = new Node('c');  
    Node d = new Node('d');  
    Node e = new Node('e');  
    a.next = b;  
    b.next = c;  
    c.next = d;  
    d.next = e;  
    System.out.println(a);  
    System.out.println(returnKth(a, 2).val);  
}  
}
```

Output:

```
abcde  
d
```


Q.12) PROGRAM TO REMOVE MIDDLE CHARACTER FROM A STRING

```
import java.util.*;

public class DeleteMiddle
{
    public static class Node
    {
        Node next;
        char val;

        public Node(char val)
        {
            this.val = val;
        }

        public String toString()
        {
            StringBuilder sb = new StringBuilder();
            Node temp = this;
            while (temp != null)
            {
                sb.append(temp.val);
                temp = temp.next;
            }
            return sb.toString();
        }
    }

    public static boolean deleteMiddle(Node node) {
        if (node == null || node.next == null)
        {
            return false;
        }
        else
        {
            {
```

```
node.val = node.next.val;
node.next = node.next.next;
return true;
}
}

public static void main(String[] args ) {
    Node a = new Node('a');
    Node b = new Node('b');
    Node c = new Node('c');
    Node d = new Node('d');
    Node e = new Node('e');
    a.next = b;
    b.next = c;
    c.next = d;
    d.next = e;
    System.out.println(a);
    deleteMiddle(c);
    System.out.println(a);
}
}
```

Output:

abcde

abde

Q.13) WRITE A PROGRAM FOR BUBBLE SORT IN JAVA

```
public class MyBubbleSort {

    // logic to sort the elements
    public static void bubble_srt(int array[])
    {
        int n = array.length;
        int k;
        for (int m = n; m >= 0; m--) {
            for (int i = 0; i < n - 1; i++) {
                k = i + 1;
                if (array[i] > array[k]) {
                    swapNumbers(i, k, array);
                }
            }
            printNumbers(array);
        }
    }

    private static void swapNumbers(int i, int j, int[] array)
    {
        int temp;
        temp = array[i];
        array[i] = array[j];
        array[j] = temp;
    }

    private static void printNumbers(int[] input) {

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

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

Output:

```
2, 4, 6, 9, 12, 23, 0, 1, 34,  
2, 4, 6, 9, 12, 0, 1, 23, 34,  
2, 4, 6, 9, 0, 1, 12, 23, 34,  
2, 4, 6, 0, 1, 9, 12, 23, 34,  
2, 4, 0, 1, 6, 9, 12, 23, 34,  
2, 0, 1, 4, 6, 9, 12, 23, 34,  
0, 1, 2, 4, 6, 9, 12, 23, 34,  
0, 1, 2, 4, 6, 9, 12, 23, 34,  
0, 1, 2, 4, 6, 9, 12, 23, 34,  
0, 1, 2, 4, 6, 9, 12, 23, 34,
```

Q.14) WRITE A PROGRAM FOR INSERTION SORT IN JAVA.

```
public class MyInsertionSort {

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

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

    public static void insertionSort(int array[])
    {
        int n = array.length;
        for (int j = 1; j < n; j++) {
            int key = array[j];
            int i = j-1;
            while ( ( i > -1 ) && ( array [i] > key ) ) {
                array [i+1] = array [i];
                i--;
            }

            array[i+1] = key;
            printNumbers(array);
        }
    }
}
```

Output:

```
2, 4, 9, 6, 23, 12, 34, 0, 1,  
2, 4, 9, 6, 23, 12, 34, 0, 1,  
2, 4, 6, 9, 23, 12, 34, 0, 1,  
2, 4, 6, 9, 23, 12, 34, 0, 1,  
2, 4, 6, 9, 12, 23, 34, 0, 1,  
2, 4, 6, 9, 12, 23, 34, 0, 1,  
0, 2, 4, 6, 9, 12, 23, 34, 1,  
0, 1, 2, 4, 6, 9, 12, 23, 34,
```

Q.15) WRITE A PROGRAM TO IMPLEMENT HASHCODE AND EQUALS.

Description:

The hashCode of a Java Object is simply a number, it is 32-bit signed int, that allows an object to be managed by a hash-based data structure. We know that hash code is an unique id number allocated to an object by JVM. But actually speaking,

Hash code is not an unique number for an object.

If two objects are equals then these two objects should return same hash code. So we have to implement hashCode() method of a class in such way that if two objects are equals, ie compared by equal() method of that class, then those two objects must return same hash code. If you are overriding hashCode you need to override equals method also.

The below example shows how to override equals and hashCode methods. The class Price overrides equals and hashCode. If you notice the hashCode implementation, it always generates unique hashCode for each object based on their state, ie if the object state is same, then you will get same hashCode. A HashMap is used in the example to store Price objects as keys. It shows though we generate different objects, but if state is same, still we can use this as key.

```

import java.util.HashMap;

public class MyHashCodeImpl {

    public static void main(String a[]) {
        HashMap<Price, String> hm = new HashMap<Price, String>();
        hm.put(new Price("Banana", 20), "Banana");
        hm.put(new Price("Apple", 40), "Apple");
        hm.put(new Price("Orange", 30), "Orange");
        //creating new object to use as key to get value
        Price key = new Price("Banana", 20);
        System.out.println("HashCode of the key: "+key.hashCode());
        System.out.println("Value from map: "+hm.get(key));
    }
}

class Price {
    private String item;
    private int price;
    public Price(String itm, int pr) {
        this.item = itm;
        this.price = pr;
    }
    public int hashCode() {
        System.out.println("In hashCode");
        int hashCode = 0;
        hashCode = price*20;
        hashCode += item.hashCode();
        return hashCode;
    }

    public boolean equals(Object obj) {
        System.out.println("In equals");
        if (obj instanceof Price) {
            Price pp = (Price) obj;
            return (pp.item.equals(this.item) && pp.price == this.price);
        }
    }
}

```



```

else {
    return false;
}

public String getItem() {
    return item;
}
public void setItem(String item) {
    this.item = item;
}
public int getPrice() {
    return price;
}
public void setPrice(int price) {
    this.price = price;
}
public String toString(){
    return "item: "+item+" price: "+price;
}
}

```

Output:

```

In hashCode
In hashCode
In hashCode
In hashCode
HashCode of the key: 1982479637
In hashCode
In equals
Value from map: Banana

```

Q.16) HOW TO GET DISTINCT ELEMENTS FROM AN ARRAY BY AVOIDING DUPLICATE ELEMENTS?

```
public class MyDisticntElements {

    public static void printDistinctElements(int[] arr){

        for(int i=0;i<arr.length;i++){
            boolean isDistinct = false;
            for(int j=0;j<i;j++){
                if(arr[i] == arr[j]){
                    isDistinct = true;
                    break;
                }
            }
            if(!isDistinct){
                System.out.print(arr[i]+" ");
            }
        }
    }

    public static void main(String a[]) {

        int[] nums = {5,2,7,2,4,7,8,2,3};
        MyDisticntElements.printDistinctElements(nums);
    }
}
```

Output:



5 2 7 4 8 3

Q.17) WRITE A PROGRAM TO FIND THE SUM OF THE FIRST 1000 PRIME NUMBERS.

```
public class Main {  
    public static void main(String args[]) {  
        int number = 2;  
        int count = 0;  
        long sum = 0;  
        while(count < 1000){  
            if(isPrimeNumber(number)){  
                sum += number;  
                count++;  
            }  
            number++;  
        }  
        System.out.println(sum);  
    }  
  
    private static boolean isPrimeNumber(int number){  
        for(int i=2; i<=number/2; i++){  
            if(number % i == 0){  
                return false;  
            }  
        }  
        return true;  
    }  
}
```

Output:



3682913

Q.18) WRITE A PROGRAM TO REMOVE DUPLICATES FROM SORTED ARRAY.

```
public class MyDuplicateElements
{
    public static int[] removeDuplicates(int[] input)
    {
        int j = 0;
        int i = 1;
        //return if the array length is less than 2
        if(input.length < 2)
        {
            return input;
        }

        while(i < input.length)
        {
            if(input[i] == input[j])
            {
                i++;
            }else
            {
                input[++j] = input[i++];
            }
        }
        int[] output = new int[j+1];
        for(int k=0; k<output.length; k++){
            output[k] = input[k];
        }

        return output;
    }
}
```

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

```
}  
}
```

Output:

```
2 3 6 8 9 10 12
```

Q.19) FIND LONGEST SUBSTRING WITHOUT REPEATING CHARACTERS .

```
import java.util.HashSet;  
import java.util.Set;  
  
public class MyLongestSubstr {  
  
    private Set<String> subStrList = new HashSet<String>();  
    private int finalSubStrSize = 0;  
  
    public Set<String> getLongestSubstr(String input){  
        //reset instance variables  
        subStrList.clear();
```

```

        finalSubStrSize = 0;
        // have a boolean flag on each character ascii value
        boolean[] flag = new boolean[256];
        int j = 0;
        char[] inputCharArr = input.toCharArray();
        for (int i = 0; i < inputCharArr.length; i++) {
            char c = inputCharArr[i];
            if (flag[c]) {
                extractSubString(inputCharArr, j, i);
                for (int k = j; k < i; k++) {
                    if (inputCharArr[k] == c) {
                        j = k + 1;
                        break;
                    }
                    flag[inputCharArr[k]] = false;
                }
            }

        } else {
            flag[c] = true;
        }

    }

    extractSubString(inputCharArr, j, inputCharArr.length);
    return subStrList;
}

private String extractSubString(char[] inputArr, int start, int end) {

    StringBuilder sb = new StringBuilder();
    for (int i = start; i < end; i++) {
        sb.append(inputArr[i]);
    }
    String subStr = sb.toString();
    if (subStr.length() > finalSubStrSize) {
        finalSubStrSize = subStr.length();
        subStrList.clear();
        subStrList.add(subStr);
    } else if (subStr.length() == finalSubStrSize) {
        subStrList.add(subStr);
    }
}

```

```
    return sb.toString();  
}
```

```
public static void main(String a[]) {  
    MyLongestSubstr mls = new MyLongestSubstr();  
    System.out.println(mls.getLongestSubstr("java2novice"));  
    System.out.println(mls.getLongestSubstr("java_language_is_sweet"));  
    System.out.println(mls.getLongestSubstr("java_java_java_java"));  
    System.out.println(mls.getLongestSubstr("abcabcbb"));  
}  
}
```

Output:

```
[a2novice]  
[uage_is]  
[_jav, va_j]  
[cab, abc, bca]
```


Q.20) HOW TO SORT A STACK USING A TEMPORARY STACK?

```
import java.util.Stack;

public class StackSort
{
    public static Stack<Integer> sortStack(Stack<Integer> input)
    {
        Stack<Integer> tmpStack = new Stack<Integer>();
        System.out.println("===== debug logs =====");
        while(!input.isEmpty())
        {
            int tmp = input.pop();
            System.out.println("Element taken out: "+tmp);
            while(!tmpStack.isEmpty() && tmpStack.peek() > tmp)
            {
                input.push(tmpStack.pop());
            }
            tmpStack.push(tmp);
            System.out.println("input: "+input);
            System.out.println("tmpStack: "+tmpStack);
        }
        System.out.println("===== debug logs ended =====");
        return tmpStack;
    }

    public static void main(String a[])
    {
        Stack<Integer> input = new Stack<Integer>();
        input.add(34);
        input.add(3);
        input.add(31);
        input.add(98);
    }
}
```

```

        input.add(92);
        input.add(23);
        System.out.println("input: "+input);
        System.out.println("final sorted list: "+sortStack(input));
    }
}

```

Output:

```

input: [34, 3, 31, 98, 92, 23]
===== debug logs =====
Element taken out: 23
input: [34, 3, 31, 98, 92]
tmpStack: [23]
Element taken out: 92
input: [34, 3, 31, 98]
tmpStack: [23, 92]
Element taken out: 98
input: [34, 3, 31]
tmpStack: [23, 92, 98]
Element taken out: 31
input: [34, 3, 98, 92]
tmpStack: [23, 31]
Element taken out: 92
input: [34, 3, 98]
tmpStack: [23, 31, 92]
Element taken out: 98
input: [34, 3]
tmpStack: [23, 31, 92, 98]
Element taken out: 3
input: [34, 98, 92, 31, 23]
tmpStack: [3]
Element taken out: 23
input: [34, 98, 92, 31]
tmpStack: [3, 23]
Element taken out: 31
input: [34, 98, 92]
tmpStack: [3, 23, 31]
Element taken out: 92
input: [34, 98]
tmpStack: [3, 23, 31, 92]

```

Element taken out: 98
input: [34]
tmpStack: [3, 23, 31, 92, 98]
Element taken out: 34
input: [98, 92]
tmpStack: [3, 23, 31, 34]
Element taken out: 92
input: [98]
tmpStack: [3, 23, 31, 34, 92]
Element taken out: 98
input: []
tmpStack: [3, 23, 31, 34, 92, 98]
===== debug logs ended =====
final sorted list: [3, 23, 31, 34, 92, 98]

Thank You!