```
public class MergeSort {
void merge(int arr[], int l, int m, int r){

       int n1 = m - l + 1;
       int n2 = r - m;


       int L[] = new int [n1];
       int R[] = new int [n2];


       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];
       int i = 0, j = 0;
       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++;
       }
       while (i < n1) {
           arr[k] = L[i];
           i++;
           k++;
       }
       while (j < n2) {
           arr[k] = R[j];
           j++;
           k++;
       }
   }
   void sort(int arr[], int l, int r) {
       if (l < r) {
           int m = (l+r)/2;
           sort(arr, l, m);
           sort(arr , m+1, r);
           merge(arr, l, m, r);
       }
   }
   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, 40, 25, 55, 16, 87};
       System.out.println("Given Array");
       printArray(arr);
       MergeSort ob = new MergeSort();
```

```
        ob.sort(arr, 0, arr.length-1);
        System.out.println("Sorted array");
        printArray(arr);
    }
}
```

|| Quick Sort

```
class Quicksort {

  static int partition(int array[], int low, int high) {


    int pivot = array[high];

    int i = (low - 1);


    for (int j = low; j < high; j++) {
      if (array[j] <= pivot) {


        i++;


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

    }


    int temp = array[i + 1];
    array[i + 1] = array[high];
    array[high] = temp;

    return (i + 1);
  }

  static void quickSort(int array[], int low, int high) {
    if (low < high) {

      int pi = partition(array, low, high);

      // recursive call on the left of pivot
      quickSort(array, low, pi - 1);

      // recursive call on the right of pivot
      quickSort(array, pi + 1, high);
    }
  }
}
```

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

    int[] data = { 8, 7, 2, 1, 0, 9, 6 };
    System.out.println("Unsorted Array");
    System.out.println(Arrays.toString(data));

    int size = data.length;
    Quicksort.quickSort(data, 0, size - 1);

    System.out.println("Sorted Array in Ascending Order ");
    System.out.println(Arrays.toString(data));
  }
}
```

Bubble Sort

```
                                public class BubbleSort {
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]){
temp = arr[j-1];
arr[j-1] = arr[j];
arr[j] = temp;
}

                }
         }

    }
public static void main(String[] args) {
    int arr[] ={3,66,85,2100,455,320,475};
    System.out.println("Before Bubble Sort");
    for(int i=0; i < arr.length; i++){
    System.out.print(arr[i] + " ");
                }
System.out.println();
 bubbleSort(arr);
 System.out.println(" After Bubble Sort");
                for(int i=0; i < arr.length; i++){
                        System.out.print(arr[i] + " ");
                }

        }
}
```

INSERTION SORT

```
                        public class InsertionSort{
    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;
        }
    }

    public static void main(String a[]){
        int[] arr1 = {47,114,325,122,543,1411,518,322};
        System.out.println("Before Sort");
        for(int i:arr1){
            System.out.print(i+" ");
        }
        System.out.println();

        insertionSort(arr1);

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


                class Stack
{
    private int arr[];
    private int top;
    private int capacity;
    Stack(int size)
    {
        arr = new int[size];
        capacity = size;
        top = -1;
    }
    public void push(int x)
    {
        if (isFull())
        {
            System.out.println("Overflow\nProgram Terminated\n");
            System.exit(-1);
        }

        System.out.println("Inserting " + x);
        arr[++top] = x;
    }
    public int pop()
    {
        if (isEmpty())
        {
            System.out.println("Underflow Program Terminated");
            System.exit(-1);
        }

        System.out.println("Removing " + peek());
```

```
        return arr[top--];
    }
    public int peek()
    {
        if (!isEmpty()) {
            return arr[top];
        }
        else {
            System.exit(-1);
        }

        return -1;
    }
    public int size() {
        return top + 1;
    }
    public boolean isEmpty() {
        return top == -1;
    }
    public boolean isFull() {
        return top == capacity - 1;
    }
}
```

Impementation of Stack

```
class Main
{
    public static void main (String[] args)
    {
        Stack stack = new Stack(3);

        stack.push(1);
        stack.push(2);

        stack.pop();
        stack.pop();

        stack.push(3);

        System.out.println("The top element is " + stack.peek());
        System.out.println("The stack size is " + stack.size());

        stack.pop();

        if (stack.isEmpty()) {
            System.out.println("The stack is empty");
        }
        else {
            System.out.println("The stack is not empty");
        }
    }
}
```

implementation od Linked list

```
public class LinkedList {

	Node head;
	static class Node {

		int data;
		Node next;

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

	// Method to insert a new node
	public static LinkedList insert(LinkedList list, int data)
	{
		Node new_node = new Node(data);

		if (list.head == null) {
			list.head = new_node;
		}
		else {
			Node last = list.head;
			while (last.next != null) {
				last = last.next;
			}
			last.next = new_node;
		}
		return list;
	}
	public static void printList(LinkedList list)
	{
		Node currNode = list.head;

		System.out.print("LinkedList: ");
		while (currNode != null) {
			System.out.print(currNode.data + " ");
			currNode = currNode.next;
		}
	}

	public static void main(String[] args)
	{

		LinkedList list = new LinkedList();

		list = insert(list, 1);
		list = insert(list, 2);
```

```
            list = insert(list, 3);
            list = insert(list, 4);
            list = insert(list, 5);
            list = insert(list, 6);
            list = insert(list, 7);
            list = insert(list, 8);
            printList(list);
      }
}
```

Doubly linked list

```
public class DoublyLinkedList {
    class Node{
        int data;
        Node previous;
        Node next;

        public Node(int data) {
            this.data = data;
        }
    }
    Node head, tail = null;
    public void addNode(int data) {
        Node newNode = new Node(data);
        if(head == null) {
            head = tail = newNode;
            l
            head.previous = null;

            tail.next = null;
        }
        else {

            tail.next = newNode;

            newNode.previous = tail;

            tail = newNode;

            tail.next = null;
        }
    }

    public void display() {
        Node current = head;
        if(head == null) {
            System.out.println("List is empty");
            return;
        }
        System.out.println("Nodes of doubly linked list: ");
        while(current != null) {


            System.out.print(current.data + " ");
            current = current.next;
        }
```

```
    }

    public static void main(String[] args) {

        DoublyLinkedList dList = new DoublyLinkedList();

        dList.addNode(1);
        dList.addNode(2);
        dList.addNode(3);
        dList.addNode(4);
        dList.addNode(5);

        dList.display();
    }
}
```

Heap sort

```
public class HeapSort
{
      public void sort(int arr[])
      {
            int n = arr.length;


            for (int i = n / 2 - 1; i >= 0; i--)
                  heapify(arr, n, i);


            for (int i=n-1; i>=0; i--)
            {

                  int temp = arr[0];
                  arr[0] = arr[i];
                  arr[i] = temp;


                  heapify(arr, i, 0);
            }
      }


      void heapify(int arr[], int n, int i)
      {
            int largest = i; // Initialize largest as root
            int l = 2*i + 1; // left = 2*i + 1
            int r = 2*i + 2; // right = 2*i + 2


            if (l < n && arr[l] > arr[largest])
                  largest = l;

            // If right child is larger than largest so far
            if (r < n && arr[r] > arr[largest])
                  largest = r;
            if (largest != i)
            {
                  int swap = arr[i];
```

```
                arr[i] = arr[largest];
                arr[largest] = swap;


                heapify(arr, n, largest);
          }
    }

    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};
          int n = arr.length;

          HeapSort ob = new HeapSort();
          ob.sort(arr);

          System.out.println("Sorted array is");
          printArray(arr);
    }
}



                          ## Selection Sort

                          public class SelectionSort{
  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 = {119,144,312,210,143,111,358,122};
      System.out.println("Before  Sort");
      for(int i:arr1){
          System.out.print(i+" ");
      }
      System.out.println();

      selectionSort(arr1);
```

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