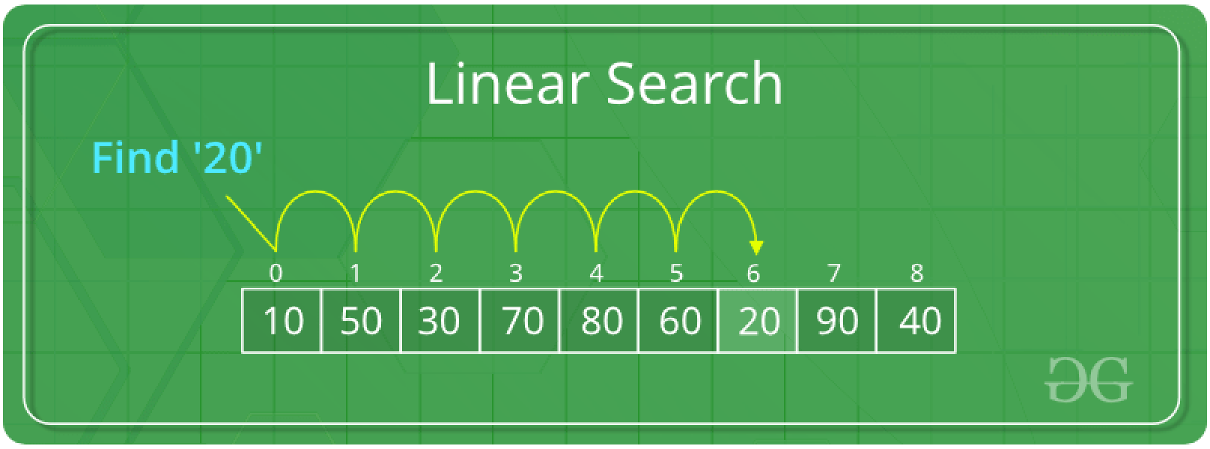
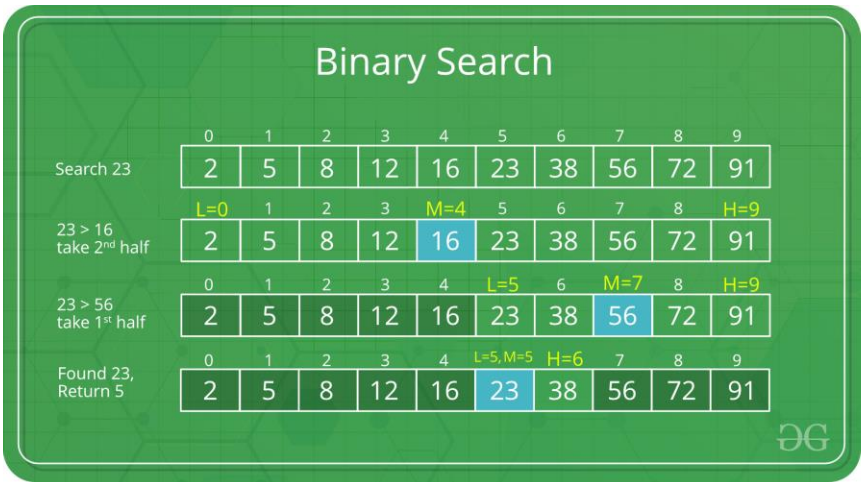
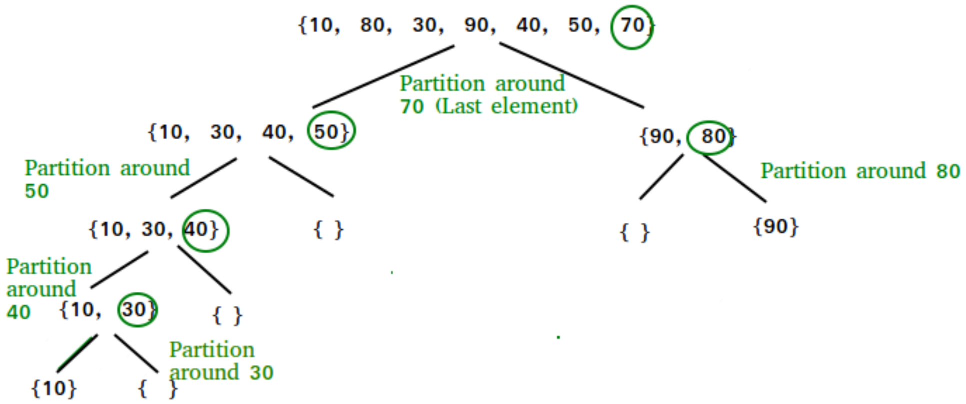


```
public class LinearSearchExample{
public static int linearSearch(int[] arr, int key){

        for(int i=0;i<arr.length;i++){
            if(arr[i] == key){
                return i;
            }
        }
        return -1;
    }
          public static void main(String a[]){
              int[] a1= {10,20,30,50,70,90};
              int key = 50;
              System.out.println(key+" is found at index: "+linearSearch(a1, key));
          }
      }
```

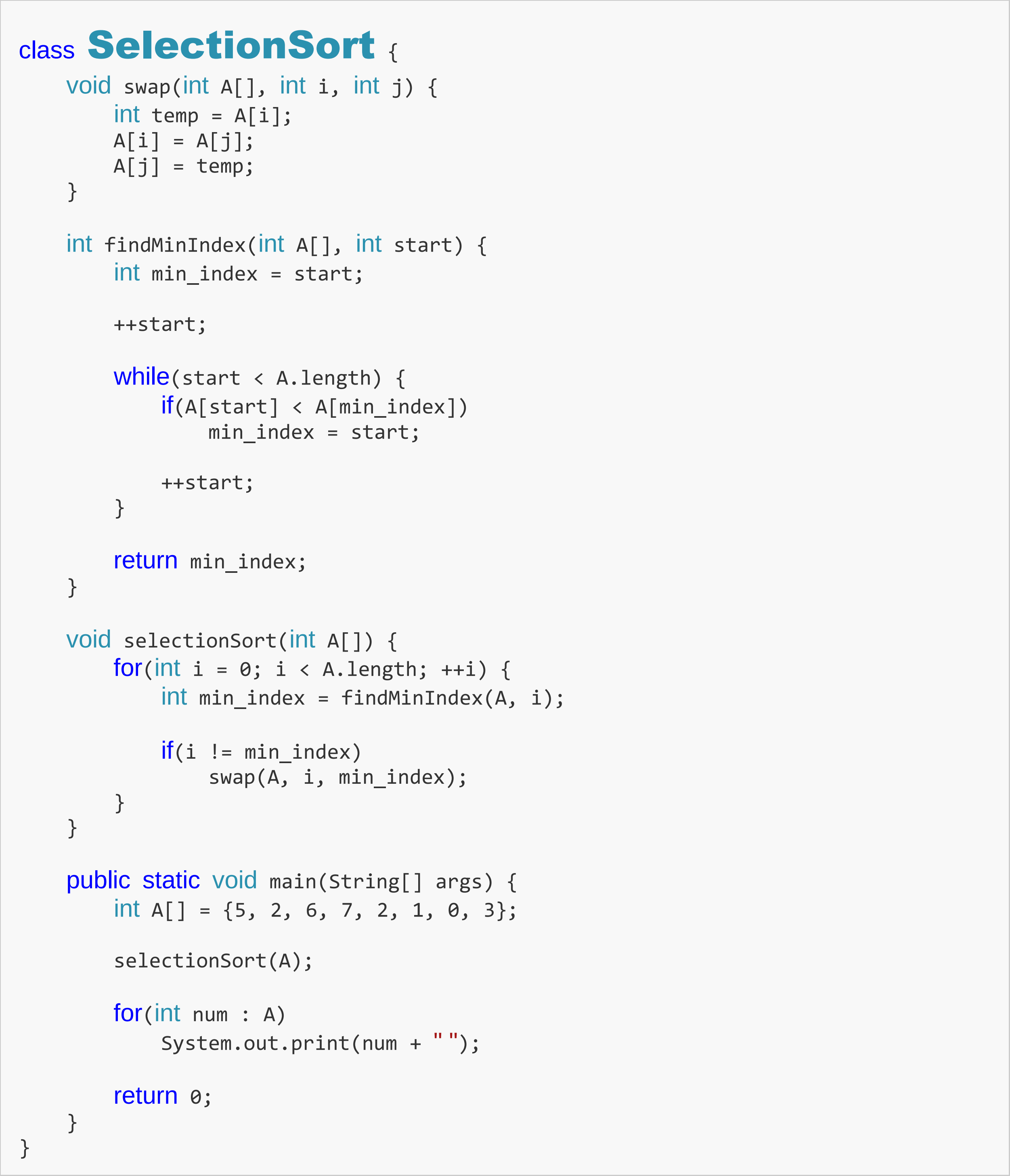


```
class BinarySearchExample{
 public static void binarySearch(int arr[], int first, int last, int key){

   int mid = (first + last)/2;
   while( first <= last ){
      if ( arr[mid] < key ){
        first = mid + 1;
      }else if ( arr[mid] == key ){
        System.out.println("Element is found at index: " + mid);
        break;
                }else{
                   last = mid - 1;
                }
                mid = (first + last)/2;
             }
             if ( first > last ){
                System.out.println("Element is not found!");
             }
           }
           public static void main(String args[]){
                  int arr[] = {10,20,30,40,50};
                  int key = 30;
                  int last=arr.length-1;
                  binarySearch(arr,0,last,key);
           }
          }
```

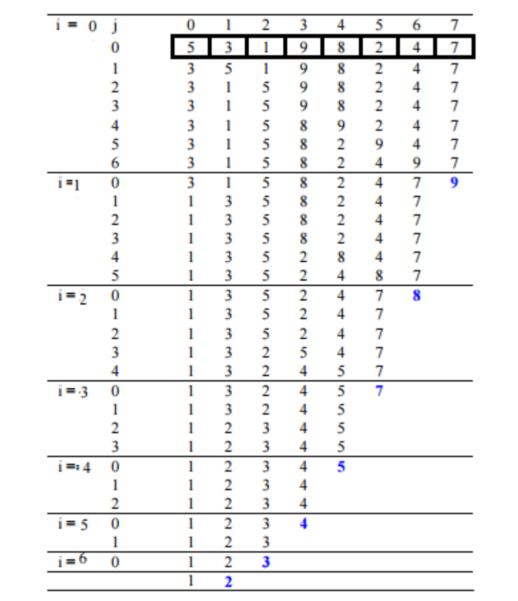


```
public class QuickSort {
public static void main(String[] args) {
        int i;
        int[] arr={90,23,101,45,65,23,67,89,34,23};
        quickSort(arr, 0, 9);
        System.out.println("\n The sorted array is: \n");
        for(i=0;i<10;i++)
        System.out.println(arr[i]);
    }
            public static int partition(int a[], int beg, int end)
            {

                int left, right, temp, loc, flag;
                loc = left = beg;
                right = end;
                flag = 0;
                while(flag != 1)
                {
                    while((a[loc] <= a[right]) && (loc!=right))
                    right--;
                    if(loc==right)
                    flag =1;
                    elseif(a[loc]>a[right])
                    {
```

```
                temp = a[loc];
                a[loc] = a[right];
                a[right] = temp;
                loc = right;
            }
            if(flag!=1)
            {
                while((a[loc] >= a[left]) && (loc!=left))
                left++;
                if(loc==left)
                flag =1;
                elseif(a[loc] <a[left])
                {
                    temp = a[loc];
                    a[loc] = a[left];
                    a[left] = temp;
                    loc = left;
                }
            }
        }
        returnloc;
    }
    static void quickSort(int a[], int beg, int end)
    {

        int loc;
        if(beg<end)
        {
            loc = partition(a, beg, end);
            quickSort(a, beg, loc-1);
            quickSort(a, loc+1, end);
        }  }}
```



```
class SelectionSort {
    void swap(int A[], int i, int j) {
        int temp = A[i];
        A[i] = A[j];
        A[j] = temp;
    }

    int findMinIndex(int A[], int start) {
        int min_index = start;

        ++start;

        while(start < A.length) {
            if(A[start] < A[min_index])
                min_index = start;

            ++start;
        }

        return min_index;
    }

    void selectionSort(int A[]) {
        for(int i = 0; i < A.length; ++i) {
            int min_index = findMinIndex(A, i);

            if(i != min_index)
                swap(A, i, min_index);
        }
    }

    public static void main(String[] args) {
        int A[] = {5, 2, 6, 7, 2, 1, 0, 3};

        selectionSort(A);

        for(int num : A)
            System.out.print(num + " ");

        return 0;
    }
}
```



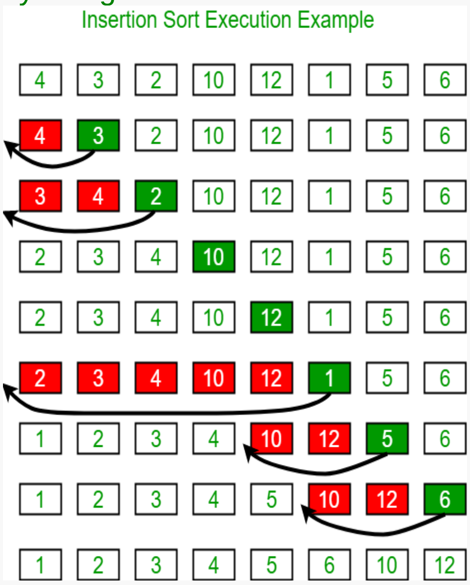
i = 0	j	0	1	2	3	4	5	6	7
	0	5	3	1	9	8	2	4	7
	1	3	5	1	9	8	2	4	7
	2	3	1	5	9	8	2	4	7
	3	3	1	5	9	8	2	4	7
	4	3	1	5	8	9	2	4	7
	5	3	1	5	8	2	9	4	7
	6	3	1	5	8	2	4	9	7
i =1	0	3	1	5	8	2	4	7	9
	1	1	3	5	8	2	4	7	
	2	1	3	5	8	2	4	7	
	3	1	3	5	8	2	4	7	
	4	1	3	5	2	8	4	7	
	5	1	3	5	2	4	8	7	
i = 2	0	1	3	5	2	4	7	8	
	1	1	3	5	2	4	7		
	2	1	3	5	2	4	7		
	3	1	3	2	5	4	7		
	4	1	3	2	4	5	7		
i = 3	0	1	3	2	4	5	7		
	1	1	3	2	4	5			
	2	1	2	3	4	5			
	3	1	2	3	4	5			
i = 4	0	1	2	3	4	5			
	1	1	2	3	4				
	2	1	2	3	4				
i = 5	0	1	2	3	4				
	1	1	2	3					
i = 6	0	1	2	3					
		1	2						

Implementation of **Bubble Sort** in Java:

```
import java.util.Scanner;
class BubbleSort {
  public static void main(String []args) {
    int n;
    Scanner in = new Scanner(System.in);
 System.out.println("Input number of integers to sort");
    n = in.nextInt();
    int array[] = new int[n];
     System.out.println("Enter " + n + " integers");
     for (int i = 0; i < n; i++)
      array[i] = in.nextInt();

    for (int i = 0; i < n - 1; i++) {
   Boolean swapped = false;
     for (int j = 0; j < n - i - 1; j++) {
       if (array[j] > array[j+1]) /* For descending order use < */
       {
         int temp = array[j];
         array[j]= array[j+1];
         array[j+1] = temp;

     swapped = true;
       }
     }
   if(!swapped)
     break;
   }
System.out.println("Sorted list of numbers:");
for (int i = 0; i < n; i++)
System.out.println(array[i]);
  }}
```

```
// example of merge sort in Java
// merge function take two intervals
// one from start to midMERGE SORT
// second from mid+1, to end
// and merge them in sorted order
void merge(int Arr[], int start, int mid, int end) {

    // create a temp array
    int temp[] = new int[end - start + 1];

    // crawlers for both intervals and for temp
    int i = start, j = mid+1, k = 0;

    // traverse both arrays and in each iteration add smaller o
f both elements in temp while(i <= mid && j <= end) {
        if(Arr[i] <= Arr[j]) {
            temp[k] = Arr[i];
            k += 1; i += 1;
        }
        else {
            temp[k] = Arr[j];
            k += 1; j += 1;
        }}// add elements left in the first interval
    while(i <= mid) {
        temp[k] = Arr[i];
        k += 1; i += 1;
    }

    // add elements left in the second interval
    while(j <= end) {
        temp[k] = Arr[j];
        k += 1; j += 1;
    }

    // copy temp to original interval
    for(i = start; i <= end; i += 1) {
```

```
                Arr[i] = temp[i - start]
        }
}// Arr is an array of integer type
// start and end are the starting and ending index of current inte
rval of Arr
void mergeSort(int Arr[], int start, int end) {

        if(start < end) {
                int mid = (start + end) / 2;
                mergeSort(Arr, start, mid);
                mergeSort(Arr, mid+1, end);
                merge(Arr, start, mid, end);
        }
}-----------------------------------------------
// Java program for implementation of Insertion Sort
public 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();
    }
    // Driver method
    public static void main(String args[])
    {
        int arr[] = {12, 11, 13, 5, 6};
        InsertionSort ob = new InsertionSort();
        ob.sort(arr);
        printArray(arr);
    }}
```