**Quick Sort Algorithm**

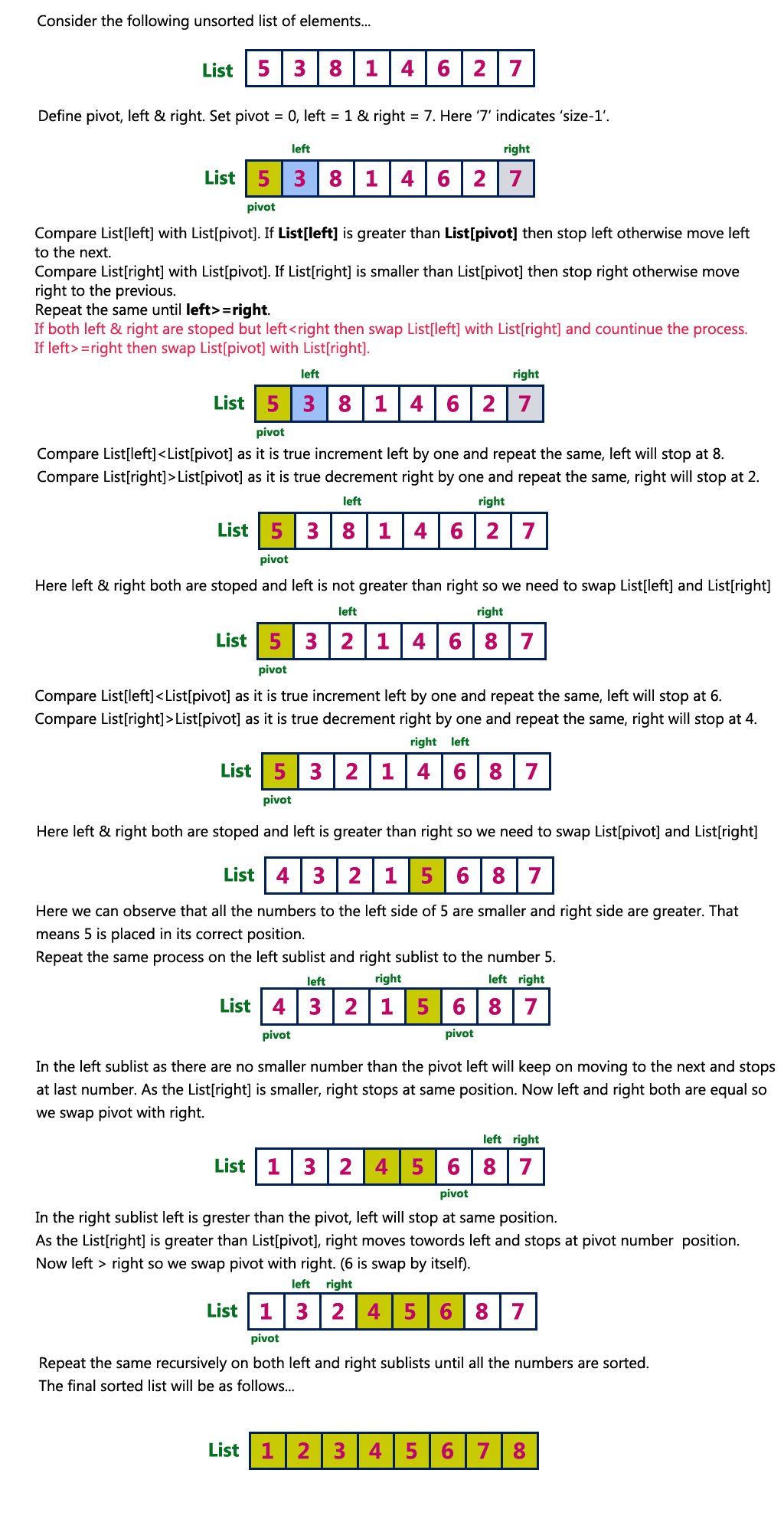
Quick sort is a fast sorting algorithm used to sort a list of elements. Quick sort algorithm is invented by **C. A. R. Hoare**.  
The quick sort algorithm attempts to separate the list of elements into two parts and then sort each part recursively. That means it use **divide and conquer** strategy. In quick sort, the partition of the list is performed based on the element called ***pivot***. Here pivot element is one of the elements in the list.  
The list is divided into two partitions such that **"all elements to the left of pivot are smaller than the pivot and all elements to the right of pivot are greater than or equal to the pivot"**.

**Step by Step Process**

In Quick sort algorithm, partitioning of the list is performed using following steps...

* **Step 1 -**Consider the first element of the list as **pivot** (i.e., Element at first position in the list).
* **Step 2 -**Define two variables i and j. Set i and j to first and last elements of the list respectively.
* **Step 3 -**Increment i until list[i] > pivot then stop.
* **Step 4 -**Decrement j until list[j] < pivot then stop.
* **Step 5 -**If i< j then exchange list[i] and list[j].
* **Step 6 -**Repeat steps 3,4& 5 until i> j.
* **Step 7 -**Exchange the pivot element with list[j] element

**Example**

vis

**Complexity of the Quick Sort Algorithm**

To sort an unsorted list with **'n'** number of elements, we need to make **((n-1)+(n-2)+(n-3)+......+1) = (n (n-1))/2** number of comparisions in the worst case. If the list is already sorted, then it requires **'n'** number of comparisions.

**Worst Case : O(n2)**  
**Best Case : O (n log n)**  
**Average Case : O (n log n)**

# Implementaion of Quick Sort Algorithm

import java.util.\*;

class Arr

{

    static void QSort(int a[],int first,int last)

    {

        int i,j,pivot,temp;

        if(first<last)

        {

            pivot=first;

            i=first;

            j=last;

            while(i<j)

            {

                while(a[i]<=a[pivot]&&i<last)

                i++;

                while(a[j]>a[pivot])

                j--;

                if(i<j)

                {

                    temp=a[i];

                    a[i]=a[j];

                    a[j]=temp;

                }

            }

            temp=a[pivot];

            a[pivot]=a[j];

            a[j]=temp;

            QSort(a, first, j-1);

            QSort(a,j+1,last);

        }

    }

}

class T

{

    public static void main(String K[])

    {

        Scanner ob=new Scanner(System.in);

        int  i,N,;

        System.out.println("Enter a Arry size");

        N=ob.nextInt();

        int a[]=new int[N];

        System.out.println("Enter a value");

        for(i=0;i<N;i++)

        {

            a[i]=ob.nextInt();

        }

        Arr.QSort(a, 0, N-1);

        System.out.println("Sorted Array =");

        for(int b:a)

        System.out.print(b+" ");

    }

}

Merge sort

Merge Sort is a [Divide and Conquer](https://www.geeksforgeeks.org/divide-and-conquer-introduction/) algorithm. It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves. **The merge() function** is used for merging two halves. The merge(arr, l, m, r) is key process that assumes that arr[l..m] and arr[m+1..r] are sorted and merges the two sorted sub-arrays into one. See following C implementation for details.

**MergeSort(arr[], l, r)**

If r > l

**1.** Find the middle point to divide the array into two halves:

middle m = (l+r)/2

**2.** Call mergeSort for first half:

Call mergeSort(arr, l, m)

**3.** Call mergeSort for second half:

Call mergeSort(arr, m+1, r)

**4.** Merge the two halves sorted in step 2 and 3:

Call merge(arr, l, m, r)

The following diagram from [wikipedia](http://en.wikipedia.org/wiki/File:Merge_sort_algorithm_diagram.svg" \t "_blank) shows the complete merge sort process for an example array {38, 27, 43, 3, 9, 82, 10}. If we take a closer look at the diagram, we can see that the array is recursively divided in two halves till the size becomes 1. Once the size becomes 1, the merge processes comes into action and starts merging arrays back till the complete array is merged.



## Program for Merge Sort in JAVA

import java.util.\*;

class Arr

{

   static void mergesort(int a[],int i, int j)

    {

        int mid;

        if (i<j)

        {

            mid=(i+j)/2;

            mergesort(a,i,mid);

            mergesort(a, mid+1, j);

            merge(a, i,mid,mid+1, j);

        }

    }

    static void merge(int a[],int i1,int j1,int i2,int j2)

    {

        int i,j,k;

        int temp[]=new int[50];

        i=i1;

        j=i2;

        k=0;

        while(i<=j1 && j<=j2)

        {

            if(a[i]<a[j])

            {

                temp[k++]=a[i++];

            }

            else

            {

                temp[k++]=a[j++];

            }

        }

        while(i<=j1)

        {

            temp[k++]=a[i++];

        }

        while(j<=j2)

        {

            temp[k++]=a[j++];

        }

        for(i=i1,j=0;i<=j2;i++,j++)

        {

            a[i]=temp[j];

        }

    }

}

class T

{

    public static void main(String K[])

    {

        int N,i;

        Scanner ob=new Scanner(System.in);

        System.out.println("Enter Array Size");

        N=ob.nextInt();

        int a[]=new int[N];

        System.out.println("Enter a Array");

        for(i=0;i<N;i++)

        {

            a[i]=ob.nextInt();

        }

        Arr.mergesort(a,0,N-1);

        System.out.println("Sorted Array=");

        for(int b:a)

        System.out.print(b+" ");

    }

}