**Batch: A1 Roll No.: 1911004**

**Experiment No. 3**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **Title: Implementation of Quick sort/Merge sort algorithm** |

**Objective:** To learn the divide and conquer strategy of solving the problems of different types

|  |  |
| --- | --- |
| Sr. No | Objective |
| CO 1 | Analyze the asymptotic running time and space complexity of algorithms. |
| CO 2 | Describe various algorithm design strategies to solve different problems and analyze  Complexity. |
| CO 3 | Develop string matching techniques |
| CO 4 | Describe the classes P, NP, and NP-Complete |

**CO to be achieved:**

**Books/ Journals/ Websites referred:**

1. **Ellis horowitz, Sarataj Sahni, S.Rajsekaran,” Fundamentals of computer algorithm”, University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein,” Introduction to algortihtms”,2nd Edition ,MIT press/McGraw Hill,2001**
3. **http://en.wikipedia.org/wiki/Quicksort**
4. **https://www.cs.auckland.ac.nz/~jmor159/PLDS210/qsort.html**
5. **http://www.cs.rochester.edu/~gildea/csc282/slides/C07-quicksort.pdf**
6. **http://www.sorting-algorithms.com/quick-sort**
7. **http://www.cse.ust.hk/~dekai/271/notes/L01a/quickSort.pdf**
8. **http://en.wikipedia.org/wiki/Merge\_sort**
9. **http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/Sorting/mergeSort.htm**
10. **http://www.sorting-algorithms.com/merge-sort**
11. **http://www.princeton.edu/~achaney/tmve/wiki100k/docs/Merge\_sort.html**

**Pre Lab/ Prior Concepts:**

Data structures, various sorting techniques

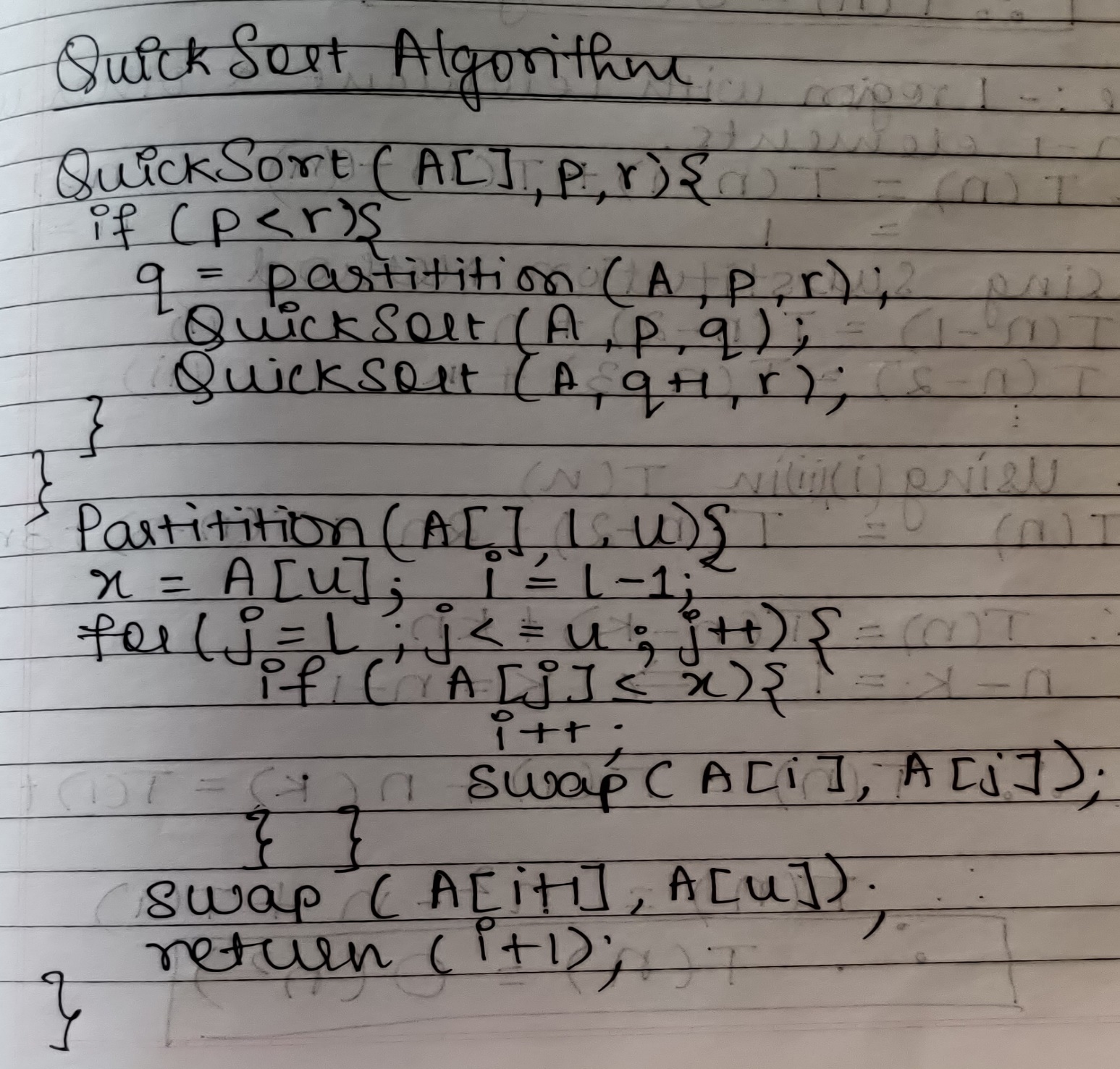
**Historical Profile:**

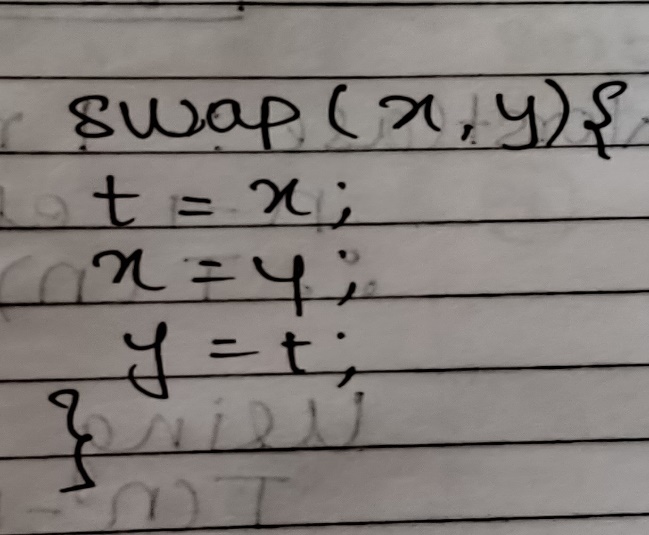
**Quicksort and merge sort are s a** divide**-**and-conquer sorting algorithm in which division is dynamically carried out. They are one the most efficient sorting algorithms.

**New Concepts to be learned:**

Number of comparisons, Application of algorithmic design strategy to any problem, Classical problem solving Vs Divide-and-Conquer problem solving.

**Algorithm** **Recursive Quick Sort:**





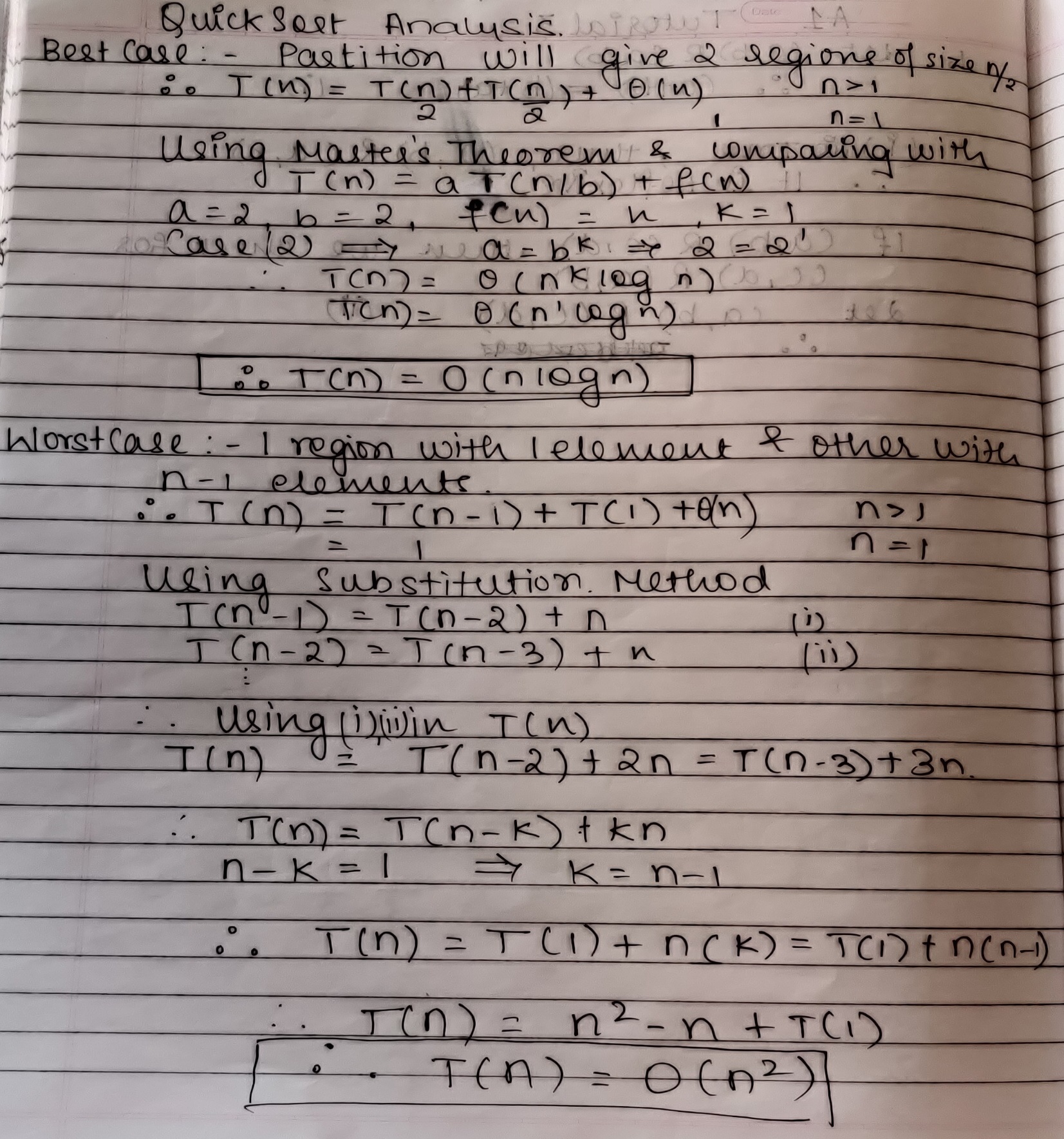
**The space complexity of Quick Sort:**

Space Complexity = O(1) as we don’t use extra space in Quick Sort For Sorting Elements .It is In-place Algorithm not taking recursive call into consideration. It makes use of stack for Recurssion.

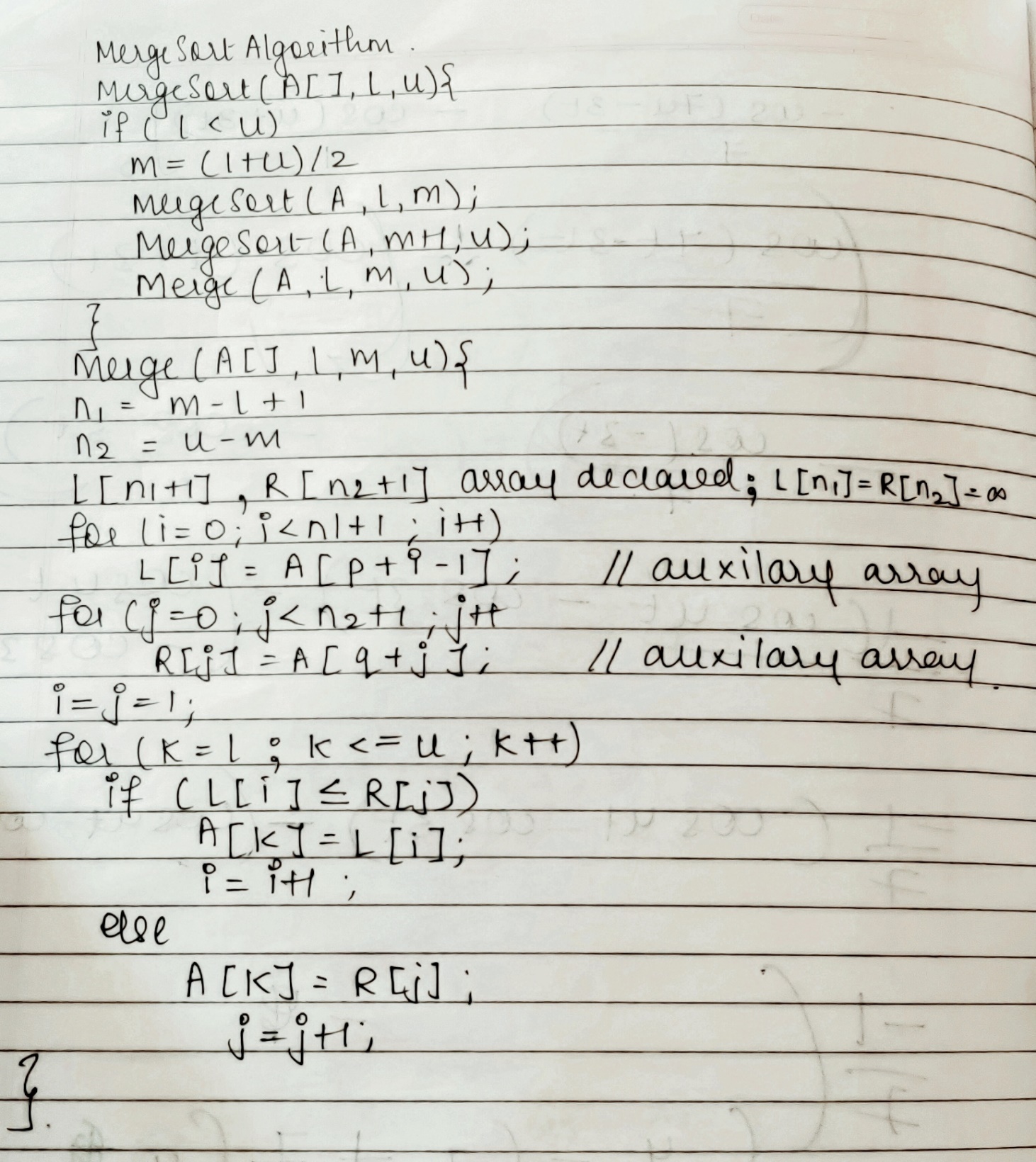
**Derivation of best case and worst case time complexity (Quick Sort)**

Best Case: O(nlog n)

Worst Case: O(n2)



**Algorithm Merge Sort**



**The space complexity of Merge sort:**

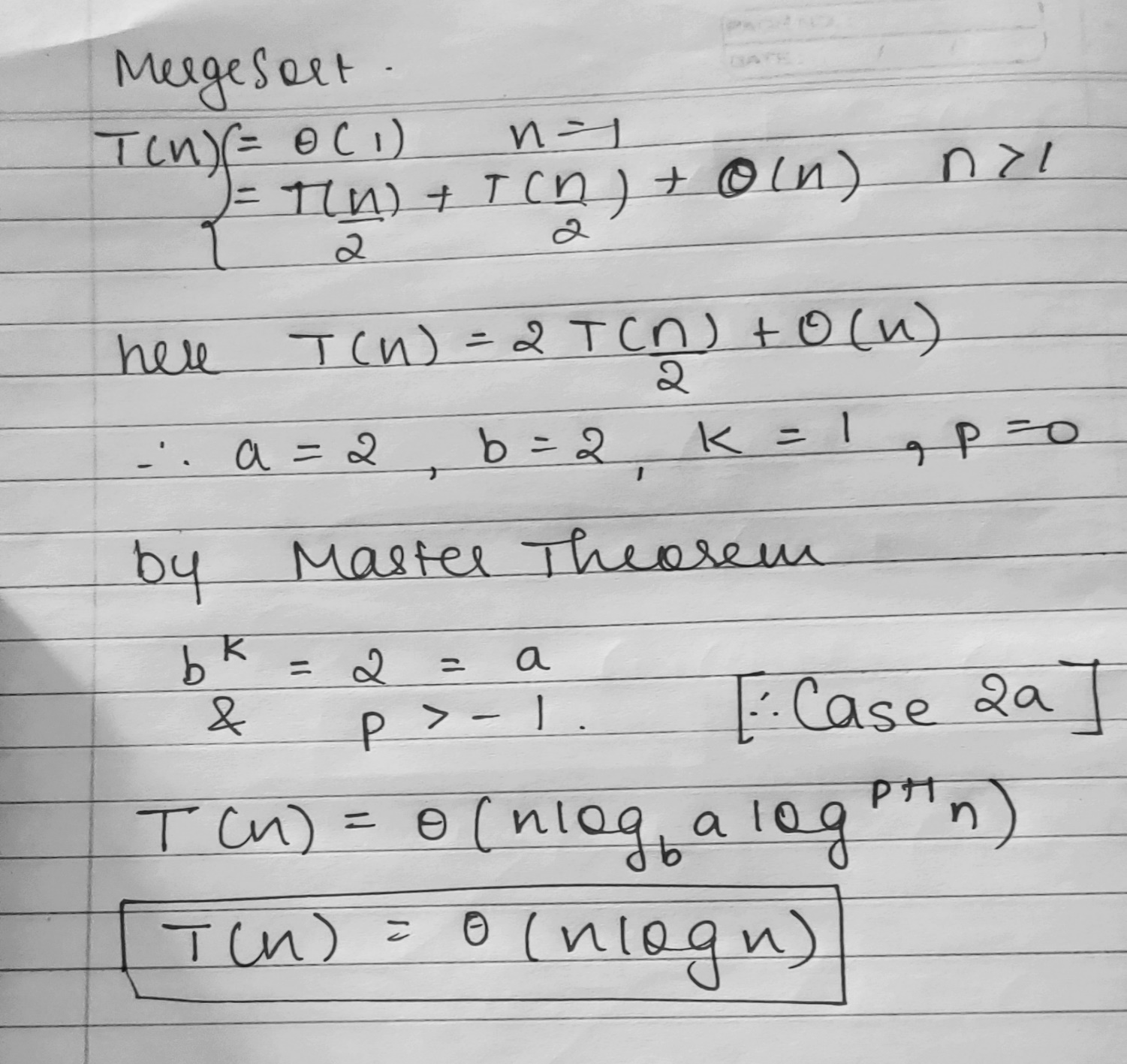
Space Complexity = O(n) as we make use of extra space in Merge Sort For Merging & Sorting Elements.It is not In-place Algorithm. It also makes use of stack for Recursive call.

**Derivation of best case and worst-case time complexity (Merge Sort)**

Best Case : O(nlog n)

Avg Case :O(nlog n)

Worst Case: O(nlog n)



**Implementation Code & Output:**

**Merge Sort**

import java.util.\*;

class MergeSort{

public static Random R;

public static void main(String[] args) {

Scanner ob = new Scanner(System.in);

System.out.println("Merge Sort");

System.out.println("Enter the no of elements ");

int n = ob.nextInt();

int[] A = new int[n];

R = new Random();

for(int i = 0; i < n; i++)

A[i] = (R.nextInt(n\*n)-i+1);

long start = System.currentTimeMillis();

System.out.println("Unsorted Array");

prt(A,0,A.length-1);

mergeSort(A,0,A.length-1);

System.out.println("\nSorted Array");

prt(A,0,A.length-1);

long end = System.currentTimeMillis();

System.out.println("\nTime Taken = "+(end - start)+" in millisec");

}

public static void mergeSort(int A[],int l,int u){

int m;

if (l<u){

m = (l+u)/2;

mergeSort(A,l,m);

prt(A,l,m);

mergeSort(A,m+1,u);

prt(A,m+1,u);

A = merge(A,l,m,u);

}

}

public static int[] merge(int A[],int l,int m,int u){

int n1 = m-l+1,n2 = u-m,i,j,k;

int[] L = new int[n1+1],R = new int[n2+1];

for(i=0;i<n1;i++) L[i]=A[l+i];

for(j=0;j<n2;j++) R[j]=A[m+j+1];

L[n1]=100000;R[n2]=100000;

System.out.print("Left Sub Array\t");

prt(L,0,n1-1);

System.out.print("Right Sub Array\t");

prt(R,0,n2-1);

for(i=0,j=0,k=l;k<=u;k++){

if (L[i]<=R[j])

A[k]=L[i++];

else

A[k]=R[j++];

}System.out.print("Merged Array\t");

prt(A,l,u);

return A;

}

public static void prt(int A[],int l,int u){

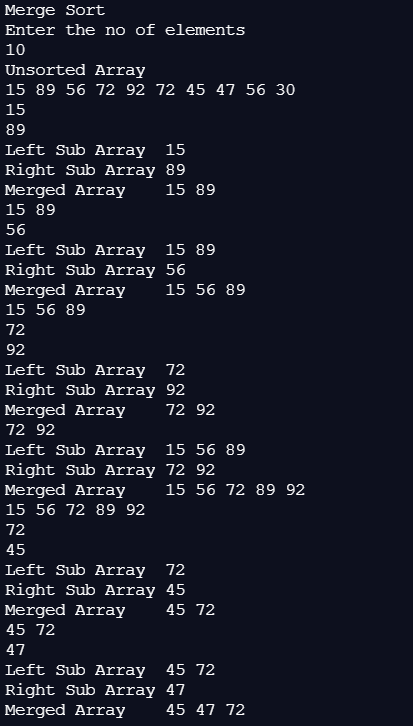
for(int i=l;i<=u;i++)

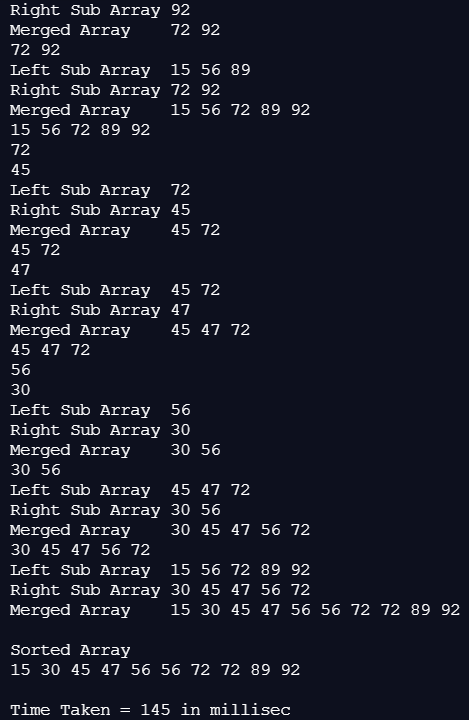
System.out.print(A[i]+" ");

System.out.println();

}

}





**Quick Sort**

import java.util.\*;

class QuickSort {

public static Random R = new Random();;

public static Scanner ob = new Scanner(System.in);

public static void main(String[] args) {

System.out.println("Quick Sort");

System.out.println("Enter the no of elements ");

int n = ob.nextInt();

int[] A = new int[n];

for(int i = 0; i < n; i++)

A[i] = (R.nextInt(n\*n)+1-i);

long start = System.currentTimeMillis();

System.out.println("Unsorted Array");

prt(A,0,A.length-1);

System.out.println("Enter 0.Pivot = Random Element 1.Pivot = last Element");

int ans = ob.nextInt();

quickSort(A,0,A.length-1,ans);

System.out.println("\nSorted Array");

prt(A,0,A.length-1);

long end = System.currentTimeMillis();

System.out.println("\nTime Taken = "+(end - start)+" in millisec");

}

public static void quickSort(int[] A,int l,int u,int ans){

int m;

if(l<u){

m = partition(A,l,u,ans);

System.out.print("Left Array\t\t");

prt(A,l,m-1);

quickSort(A,l,m-1,ans);

System.out.print("Right Array\t\t");

prt(A,m,u);

quickSort(A,m,u,ans);

}

}

public static int partition(int[] A,int l,int u,int ans){

int x,i,j,t;

if(ans==0)

randomize(A,l,u);

x = A[u];

System.out.println(x+" is pivot element");

System.out.print("Before Partition\t\t");

prt(A,l,u);

i = l-1;

for(j=l;j<u;j++)

if(A[j]<=x){

i++;

t=A[i];

A[i]=A[j];

A[j]=t;

}

t=A[i+1];

A[i+1]=A[u];

A[u]=t;

System.out.print("After Partition\t \t");

prt(A,l,u);

return (i+1);

}

public static void randomize(int A[],int l,int u){

int r = R.nextInt(u-l+1)+l;

int t=A[u];

A[u]=A[r];

A[r]=t;

}

public static void prt(int A[],int l,int u){

for(int i=l;i<=u;i++)

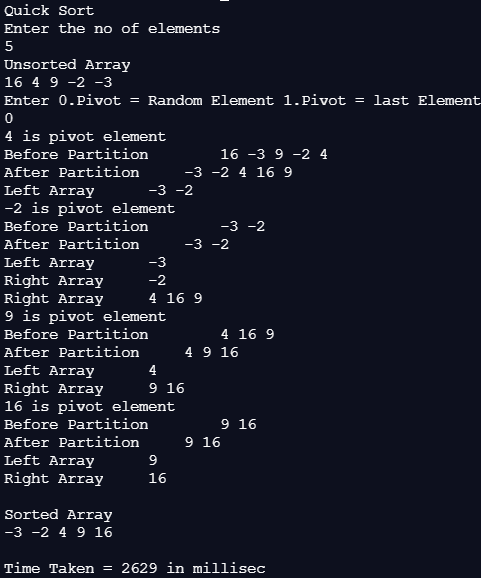
System.out.print(A[i]+" ");

System.out.println();

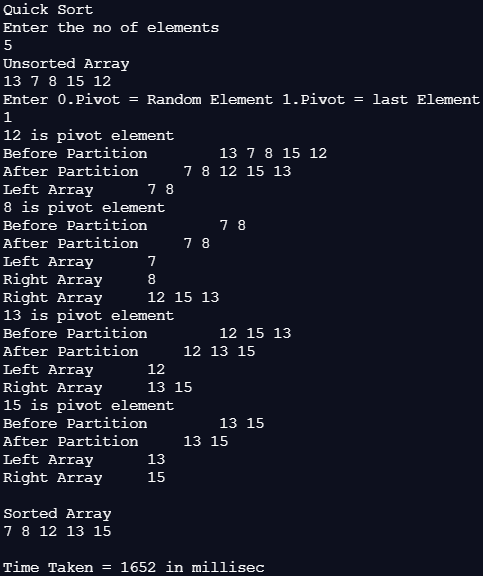
}

}

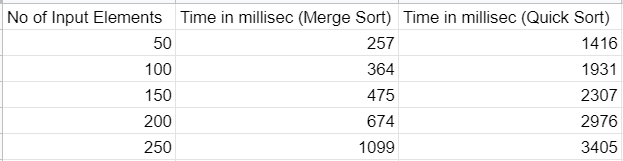
**Random Pivot**

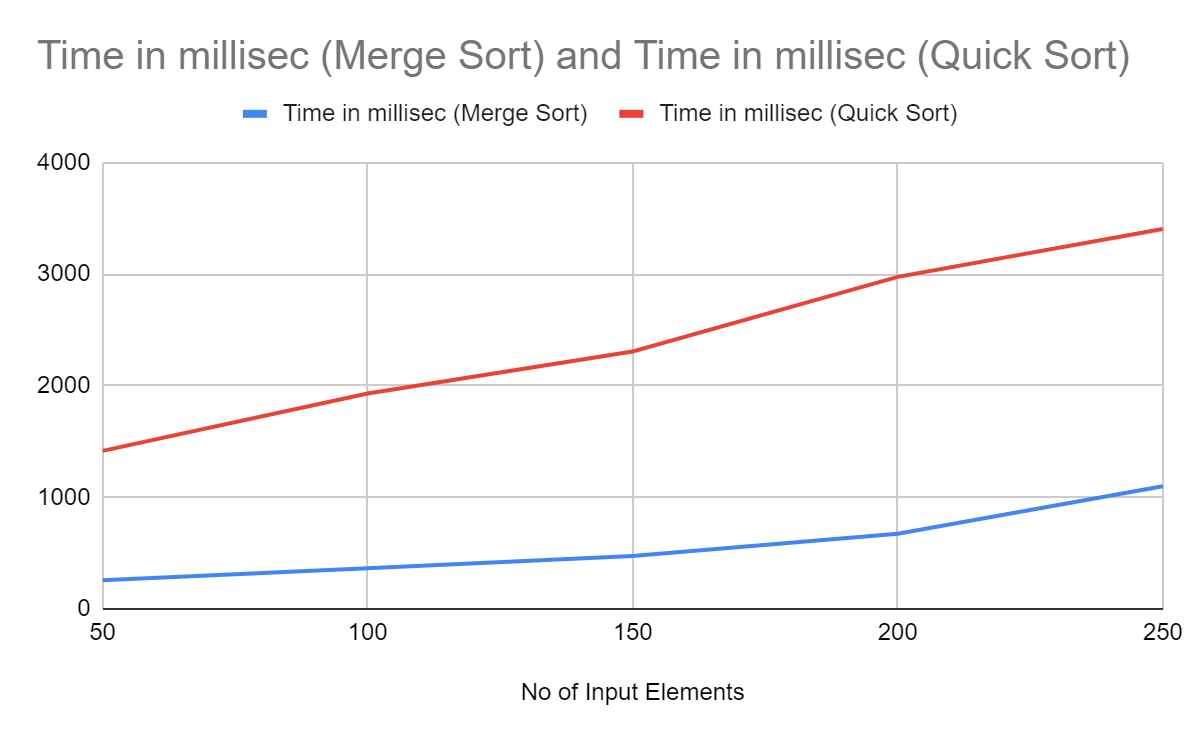


**Last Element Pivot**

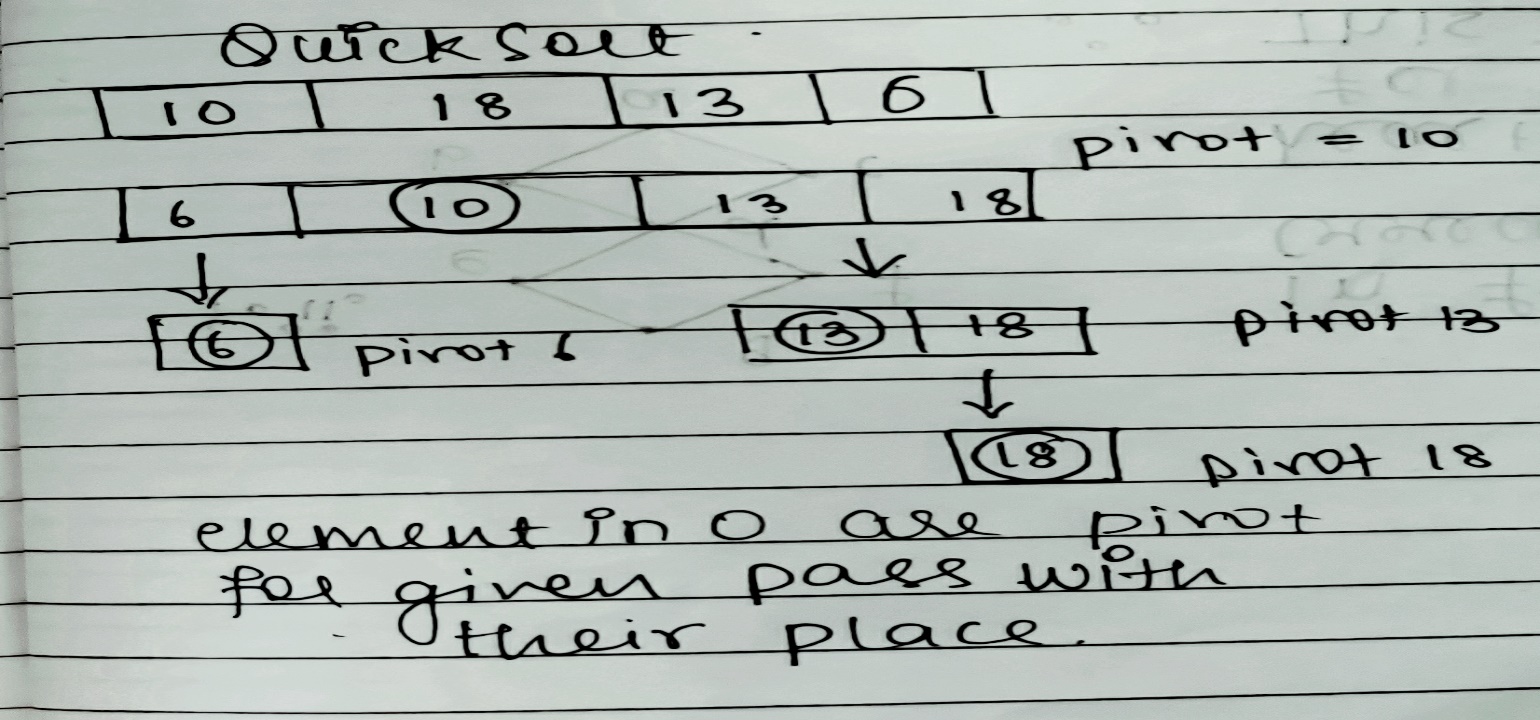


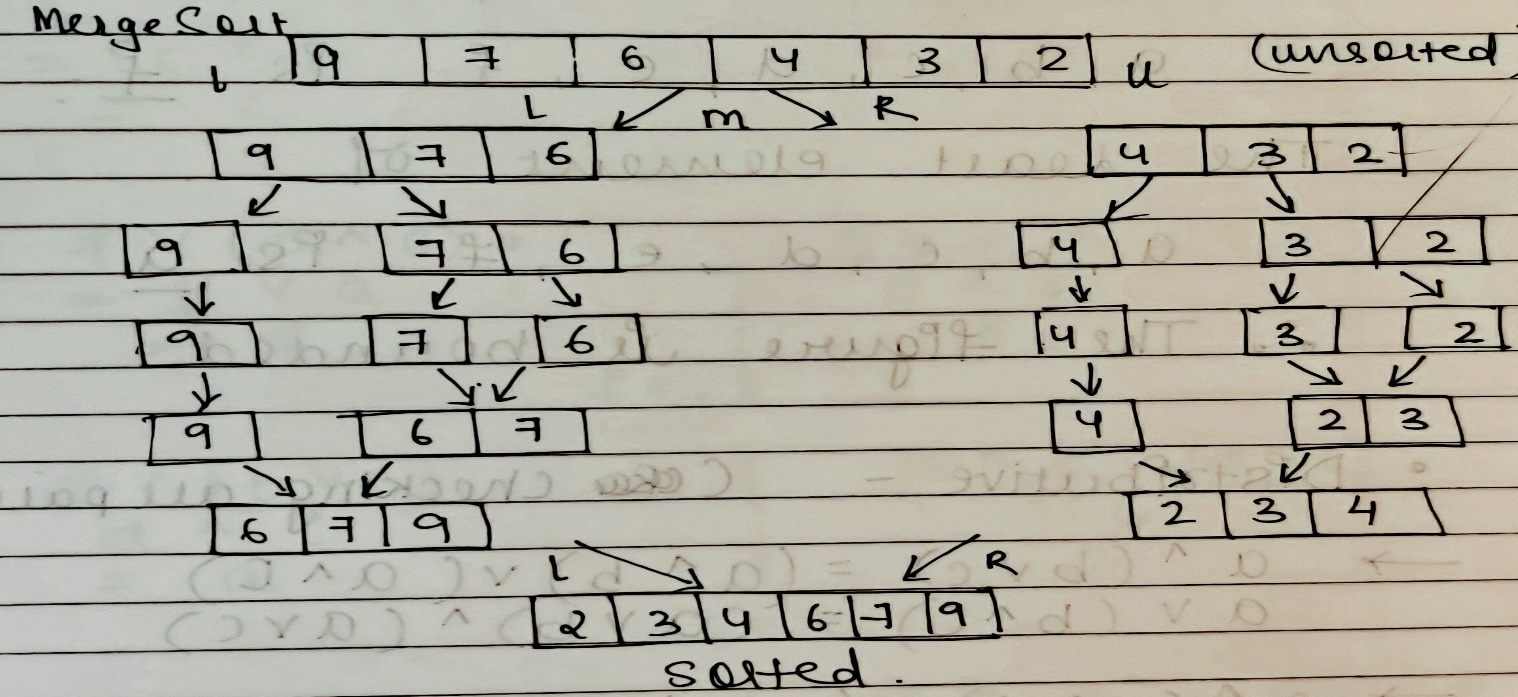
**Graph :**



****

**Example for quicksort/Merge tree for merge sort:**





**Conclusion:**

Thus we have successfully understood & implemented Quick Sort & Merge Sort in Java using Divide & Conquer method, derived Best & Worst Case Condition for both. Thus, from Graph we can conclude that Quick Sort takes less time than Merge Sort for Small Input but as Size Of Input is Increased, Merge Sort takes less time than Quick Sort**.**