***Problem Defination:-***

Implement Randomised quick sort on mobile number data set. The mobile number can be compressed to 2 digit value. The program is expected to handle data set of [0-99] mobile numbers. If it is required to generate group of mobile numbers, demonstrate the application of hashing to develop suitable strategy.

***Detailed Description:-***

A dataset of 100 mobile numbers has to be taken . Each 10-digit number has to be converted into 2-digit number by the application of hashing. These 2 digit numbers have to be sorted with the use of randomized quick sort. The sorting has to be displayed step by step and the pivot elements and the last elements of the sub- arrays have to be stored into the BST. The Traversal of the BST will give us the sorted list.

***Technology Stack: -***

The entire coding is done in Java with NetBeans IDE 8.0.1.

***Classes and Functions: -***

The program makes the use of 4 classes: -

1. Quicksort (Main Class)

2. hash

3. sort\_class

4. BST

***1.Quicksort: -***

This is the main class where the array of hundred mobile numbers is initialized. Each number is in the form of a string which is converted into integer by taking individual string in the ***array(a)*** and then using that ***array(a)*** as a passing argument into the ***quadraticprobing function*** of the ***class hash*** which returns a 2-digit number for that string.

The ***finalarray*** is the array that stored these 2 digit values and also is final sorted array.

***2.hash: -***

This class has a method “quadraticprobing”. This class is responsible to convert each mobile number into a 2-digit number***.***

***int quadraticprobing(int data[])***: -

The function accepts array as an argument. This array contains 10 digits of one mobile number. This array undergoes quadratic hashing. The hashed elements get stored in hash\_array and each of the element is multiplied with the index to get a 2-digit number. If the number is greater than 99 then it is subtracted by some random number (0-100). If it is less than 0 then a random number (0-99) is added to it till it becomes a 2-digit number.

This 2-digit number is returned back.

***3.sort\_class: -***

The methods that make this class are:- 1.void quicksort(int a[],int p,int q)

2.void display()

3.int partition(int arr[],int p,int q)

1***. void quicksort(int a[],int p,int q):-***

The function calls another function i.e. partition if p<q. The partition function results an index which divides the array (a) into 2 halfs. The index is called pivot. The pivot is stored in array(pivot\_arr). The partition method is called for all the partitions that we get by pivot. It also counts the number of times the quick sort is being implemented to get the final sorted array.

***2.void display (): -***

Used to display the array(pivot\_arr) and insert each element of this array into the BST. Atlast the BST is displayed using inorder Traversal.

***3.int partition(int arr[],int p,int q):-***

The method selects a random number(r) between p and q and divides the array into 2 parts. First part with elements less than arr[r] and second part with number greater than arr[r]. The length of both these arrays should be greater than or equal to (q-p)/4. If so, then r is the pivot and the function returns r.

***4.BST: -***

The class that is used to insert nodes into the binary search tree.

***Key Functionality: -***

The Program shows the count of the iterations used to sort 100 numbers. It was found that for every successful run, the count of iterations were different. Thus it gave us the maximum and minimum count of iterations and thus helping us to compare it with the deterministic algorithm of quick sort.This algorithm is not optimal, but also it won’t **always** give the worst case complexity for the same input because every time the partitioning done is different.