UNIVERSITY INSTITUTE of COMPUTING
Asia's Fastest Growing University

NAAC GRADE A+ ACCREDITED UNIVERSITY

Experiment: 1

Sort a given set of elements using the Quick sort

Student Name: Nitish Rai **UID:** 23MCA20326

Branch: Computers Application **Section/Group:** 4(A)

Semester: 2nd **Date of Performance:** 19/01/2024

Subject Name: Design and Analysis of Algorithms **Subject Code:** 23CAH-511

A. The task to be done:

In this experiment, we'll delve into the quicksort algorithm, tackling the challenge of ordering a set of elements. We'll choose a pivot element, and then partition the set: elements smaller than the pivot migrate left, while larger ones shift right. This dance recursively for each sub-partition until every element finds its rightful place in the sorted array.

B. Steps of Experiment:

- Understand the problem and divide it into parts.
- Install JDK: If Java isn't installed on your system, download and install it.
- Open IDE(Integrated development environment) like VS code.
- Start by creating a new Java file with a .java extension (e.g., program.java).
- Write your code in a structural manner taking care of indentation to maintain readability.
- Execute your code.

C. Algorithm

Here's the pseudocode for Quick Sort in points, choosing the last element as the pivot:

1. Initialization:

- Define the list of points**.**
- Set the low and high indices to 0 and the length of points - 1, respectively.

2. Recursive Partition:

- If low is greater than or equal to high, the sublist is already sorted, so return.
- Set the pivot as the point at the high index.
- Initialize i as low - 1.

3. Partitioning Loop:

- Iterate through the points from low to high - 1:
    - If the current point's value (points[j]) is less than or equal to the pivot value:
        - Increment i.
        - Swap the points at i and j.
    -
-
- Increment high by 1 to place the pivot in its final position.

4. Recursive Calls:

- Recursively call quickSort with points, low, and i.
- Recursively call quickSort with points, i + 1, and high - 1.

D. Steps For Experiment

Practical Code:

```
import java.util.Scanner;

public class
ExperimentOneQuickSort_DAA {

  public static void quickSort(int
ary[], int low, int high) {
    if (low < high) {
      int pivort = partition(ary, low,
high);

      quickSort(ary, low, pivort - 1);
      quickSort(ary, pivort + 1,
high);
    }
  }

  private static int partition(int[]
ary, int low, int high) {
    int pivot = ary[high];
    int i = low - 1;

    for (int j = low; j < high; j++) {
      if (ary[j] < pivot) {
        i++;
        int temp = ary[i];
        ary[i] = ary[j];
        ary[j] = temp;
      }
    }
    i++;
    int temp = ary[i];
    ary[i] = ary[high];
    ary[high] = temp;
    return i;
  }
```

```
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    System.out.println("Enter Size: ");
    int size = scn.nextInt();
    int ary[] = new int[size];
    System.out.println("Enter the data in array: ");
    for (int i = 0; i < ary.length; i++) {
      ary[i] = scn.nextInt();
    }
    int low = 0;
    int high = ary.length - 1;
    quickSort(ary, low, high);

    for (int i : ary) {
      System.out.print(i + " ");
    }

    scn.close();
  }
}
```

E. Output:

PS C:\Users\Nitish\Desktop\DAA> cd "c:\Users\Nitish\Desktop\DAA\" ; if ($?) { javac ExperimentOneQuickSort_DAA.java } ; if ($?) { java ExperimentOneQuickSort_DAA }
Enter Size:
7
Enter the data in array:
2 9 5 6 7 6 1
1 2 5 6 6 7 9
PS C:\Users\Nitish\Desktop\DAA>

F. Time Complexity:

Worst: O(n^2)

Average: O(nlogn)

Learning outcomes:

- Understand the concepts of functions, recursion and quick sort.
- Understand how to take user inputs.
- Understand the concept of the creation of an array.
- Understanding the concept of printing the output with the help of for each loop.
- Understanding how to deal with errors regarding array index out of bound.