Lab Assignment #2
Insertion Sort

Full Name: Shreyas Srinivasa
BlazerId: SSRINIVA

CS 303 Algorithms and Data Structures

Spring 2023

1. Problem Specification

The purpose of this project was to implement an algorithm for insertion sort and measure its execution time with a number of datasets of progressively increasing size. This execution time was further used to study the connection between real time performance and theoretical complexity analyses.

2. Program Design

```java
public static void insertionSort(List<Integer> arr) {
    int N = arr.size();

    int temp, currLoc;
    for (int i=1; i<N; i++){
        currLoc = i;
        while (currLoc > 0 && arr.get(currLoc - 1) > arr.get(currLoc)){
            temp = arr.get(currLoc);
            arr.set(currLoc, arr.get(currLoc - 1));
            arr.set(currLoc - 1, temp);
            currLoc--;
        }
    }
}
```

First the algorithms of insertion sort were implemented in Java code

```java
public static List<Integer> readElementsFromFile(String fileName) {
    List<Integer> elements = new ArrayList<Integer>();

    try (Scanner fileScanner = new Scanner(Paths.get(fileName))) {
        while(fileScanner.hasNextLine()) {
            String line = fileScanner.nextLine();
            elements.addAll(Arrays.asList(line.split(",")).stream().map(Integer::valueOf)
                    .collect(Collectors.toList()));
        }
        fileScanner.close();
    } catch (Exception e) {
        System.out.println("Error: " + e.toString());
    }
    return elements;
}
```

A function to read the datasets using a file name parameter was implemented.

```java
public static void main(String args[]) {

    List<Integer> sizes = Arrays.asList(1000, 2500, 5000, 10_000, 25_000, 50_000, 100_000, 250_000, 500_000,
                                        1_000_000, 2_500_000, 5000_000, 10_000_000);
    Instant start, finish;
    long timeElapsed;

    System.out.format("%-30s %-30s\n", "Dataset Size", "Time (nanoseconds)");

    for(Integer s: sizes) {
        List<Integer> elements = readElementsFromFile(String.format("%d.txt", s));
        start = Instant.now();
        insertionSort(elements);
        finish = Instant.now();
        timeElapsed = Duration.between(start, finish).toNanos();
        System.out.format("%-30d %-30d\n", s, timeElapsed);
    }
}
```

Function to calculate and print the time taken in nanoseconds by the insertion sort algorithm for each dataset was written. This produced our data for further study

3. **Output**

```
Dataset Size                   Time (nanoseconds)
1000                           103679345
2500                           130680176
5000                           306504867
10000                          659823430
25000                          3093332964
50000                          14323938173
100000                         65096937402
250000                         460705915485
500000                         2211972708759
1000000                        10414209220279
```

```
Dataset Size      Time (nanoseconds)  Time (seconds)
1000              103679345           0.103679345
2500              130680176           0.130680176
5000              306504867           0.306504867
10000             659823430           0.65982343
25000             3093332964          3.093332964
50000             14323938173         14.323938173
100000            65096937402         65.096937402
250000            460705915485        460.705915485
500000            2211972708759       2211.972708759
1000000           10414209220279      10414.209220279
```

4. Analysis and Conclusions

The running times for insertion sort:

- Worst case: $\Theta(n^2)$.
- Best case: $\Theta(n)$.
- Average case for a random array: $\Theta(n^2)$
- "Almost sorted" case: $\Theta(n)$.

If you had to make a blanket statement that applies to all cases of insertion sort, you would have to say that it runs in $O(n^2)$ time. You cannot say that it runs in $\Theta(n^2)$ time in all cases, since the best case runs in $\Theta(n)$ time. And you cannot say that it runs in $\Theta(n)$ time in all cases, since the worst-case running time is $\Theta(n^2)$.

The Time Complexity of an algorithm/code is **not** equal to the actual time required to execute a particular code, but the number of times a statement executes.

The **actual time required to execute code is machine-dependent** and also it considers network load if your machine is in LAN/WAN.

Instead of measuring actual time required in executing each statement in the code,Time Complexity considers how many times each statement executes.

There is a clear evidence to support the theoretical model of time complexity, as we do eventually see significant increases of execution time with increasing input size. Thus the time complexity analysis is an excellent indicator for real world machine specific performance.

5. References

- https://www.educative.io/courses/visual-introduction-to-algorithms/kMABY
- https://brilliant.org/wiki/insertion/
- https://www.geeksforgeeks.org/insertion-sort/