Name : Kiran Tulsulkar NUID : 001057882

EECE 7205 : Fundamentals of Computer Engineering

Assignment 4

Problem 1 :

In this problem you will test and analyze the Insertion, Heap, and Quick sort algorithms. To carry out his task, write a C++ program with the following requirements:

1. Analyze the algorithms by sorting, in ascending order, the BST, AVG, and WST arrays. Each one of these three arrays has 1000 integers where:
   a. BST has 1000 integers already sorted in ascending order (e.g., 10, 20, 30, …etc.).
   b. AVG has 1000 randomly generated integers, where each integer is between 0 and 100,000.
   c. WST has 1000 integers sorted in descending order (e.g., 1000, 990, 980, …etc.).

   Note: Make sure that all algorithms are tested with identical arrays. Therefore, at the beginning of your program, create three identical versions of the BST, AVG, and WST arrays where each algorithm sorts its own version.

2. Define two global integer variables: `moves` and `comps`. Initialize these variables to zero before every call to a sorting algorithm. Add any necessary code to the algorithms' implementation to carry out the following:
   a. Increment `moves` with every movement of any of the elements to be sorted from one "place" to another. However, the following is not considered a move: making a "copy" of an element that will not result eventually of that element being copied to another location in the array (i.e., the element stays in its place). Consider that the swap operation requires three movements of two elements.
   b. Increment `comps` with each comparison operation on the elements to be sorted.
   c. After each call to the sorting algorithms functions (3 algorithms × 3 arrays = 9 calls), write to a text file, `sort.txt`, the values of `moves` and `comps` correspond to each algorithm.
3. After calling an algorithm function to sort an array, verify that the array is sorted. One way to do that is to confirm that every element *i* in the array has a value less than or equal to the value in element *i*+1 (write a function for this task and re-check your code if it returns a false verification).
4. After running your program, copy the numbers in the text file, `sort.txt`, to an excel sheet. Use these numbers to generate two graphs. One graph compares the number of moves needed by the three algorithms for the three cases: best, average, and worst. Another graph does the same but for the number of comparisons performed by the algorithms.

In your homework report, answer the following:

i. Why does the Insertion sort algorithm result in zero moves when sorting an already sorted array (the best case)?
ii. Why does the Insertion sort algorithm result in 999 comparisons when sorting an already sorted array (the best case)?
iii. From your two excel graphs, comment on the performance of the sort algorithms under different scenarios (best, average, and worst).

Name : Kiran Tulsulkar NUID : 001057882

Results :

```
Best Case :

Insertion Sort
No Error
Moves : 0
Comparisons : 999

Heap Sort
No Error
Moves : 29124
Comparisons : 30627

Quick Sort
No Error
Moves : 0
Comparisons : 501499
```

```
Average Case :

Insertion Sort
No Error
Moves : 755814
Comparisons : 252937

Heap Sort
No Error
Moves : 27120
Comparisons : 28623

Quick Sort
No Error
Moves : 12279
Comparisons : 12929
```

```
Worst Case :

Insertion Sort
No Error
Moves : 1498500
Comparisons : 500499

Heap Sort
No Error
Moves : 24948
Comparisons : 26451

Quick Sort
No Error
Moves : 1500
Comparisons : 501499
```

Output for Problem 1

Name : Kiran Tulsulkar NUID : 001057882

Q1. Why does the Insertion sort algorithm result in zero moves when sorting an already sorted array (the best case)?

In already sorted array all the numbers are in their correct positions and therefore when comparison takes place the condition becomes false leading to swap operation not being performed. Therefore, the insertion sort algorithm leads to zero moves.

Q2. Why does the Insertion sort algorithm result in 999 comparisons when sorting an already sorted array (the best case)?

For Insertion Sort 999 comparisons takes place even if the elements are already sorted because each element is compared with other in the for loop. Best case for insertion sort is O(n) implying for a best case that the algorithm will run and compare for n number of times. Hence, we get 999 comparisons for array of 1000 integers .

Q3. From your two excel graphs, comment on the performance of the sort algorithms under different scenarios (best, average, and worst).

Let us consider Best Case :

For moves operation Insertion Sort and Quick Sort have least or minimum number of moves than heapsort. For compare operation Insertion Sort has least number of moves with Complexity of O(n) and 999 Comparisons. With Quicksort having highest number of comparisons.

Let us consider Average Case :

In Average case Quicksort Algorithm has least number of move and compare operations. Whereas insertion sort having the highest number of moves and compares.

Let us consider Worst Case :

In Worst case, Quicksort is the minimum number among the three for move operations and Heapsort is the minimum number for compare operations.

In insertion and quicksort where variation in move and compare operations have high variations, the heapsort algorithm has constant values. Which makes heapsort algorithm preferable among the three. From the graph it is evident that the heapsort has constant values in all the three cases for move and compare both the operations.

Name : Kiran Tulsulkar NUID : 001057882

Graphs :