CST8130 Data Structures

Lab3_Simple Sorting and Recursive Divide-and-Conquer Sorting Algorithms

Problem Description:

Make a copy of Lab2 and rename it as Lab3. Create a new class called **SortingAlgorithms** and in it create methods that implement Bubble Sort, Insertion Sort, Selection Sort, Merge Sort and Quick Sort algorithms for displaying the sorted integers from a randomly generated array. Create separate methods for each type of sorting named as *BubbleSort*, *InsertionSort*, *SelectionSort*, *MergeSort* and *QuickSort.* Rename the test driver class as **TestSearchSort**.

This lab requires use of large datasets. Modify your genRandomInts method to use SecureRandom class and generate an UNSORTED random array (named randomArr) of 1000 integers between 20 and 1000 exclusive. Examine the structure of your application and make a programming decision as to where to locate genRandomInts method. Run Tests for all 5 types of Sorting algorithms with the same randomarr values as below.

When the program is run, it displays the following menu:

Select your option in the menu below:

```
1: Initialize and populate an array of 1000 random integers.
2: Perform a recursive binary and linear search.
3: Perform iterative binary and linear search.
4: Sort the array
5: Quit
```

When Menu option 4 is selected, a submenu of all the five sorting algorithms is displayed as shown below.

Selecting a sort option in the submenu say 'B' for for Bubble sort displays the **unsorted array** followed by the **name of sorting algorithm** with **its complexity,** whether it is **in-place** or **not in-place** and the **sorted array** in that sequence as shown in the sample output further down (**To get a clear picture of the requirements look at the SAMPLE OUTPUT**):

B. Bubble Sort (Simple sorting algorithm - O(n2) Complexity)
I. Insertion Sort (Simple sorting algorithm - O(n2) Complexity)
S. Selection Sort (Simple sorting algorithm - O(n2) Complexity)
M. Merge Sort (Recursive Divide-And-Conquer - O(n log n) Complexity)
Q. Quick Sort (Recursive Divide-And-Conquer - O(n log n) Complexity)
R. Return to main menu

//**NOTE**: the random array values will be displayed instead of x1, x2, x2, x3, x4

Use the built-in *nanotime()* and *currentTimeMillis()* methods of the System class to determine how long each of the above sorting algorithm takes in nanoseconds and milliseconds. Time taken by each operation should be displayed on the screen. Hint: wrap each method in with the timing methods.

HINTS

You may create 5 copies of the unsorted data array such as bubbleArray, selectArray, insertArray, mergeArray and quickArray and pass each to its corresponding method. That way the sub options of menu option 4 will initially each display unsorted data.

Alternatively and more efficiently you can pass the unsorted array created by SecureRandom class directly to the sorting methods. That way each of the five methods invoked by menu option 4 receives unsorted data.

GOAL: For you to learn how Bubble sort, Insertion sort, Selection sort, Merge sort and Quick sort algorithms work and estimate programmatically the time each algorithm takes to sort the same data array in terms of nanoseconds milliseconds. See Sample Output below.

Check out the basic sorting algorithms here and here.

Generate Javadoc for your Lab3 application with meaningful comments and descriptions for all classes and methods.

Submit your solution as a zip folder of source files and Javadoc through this Link.

Alternatively, Submission Link is also available under **Activities =>Assignments => Lab3**

Lab 4: Grading Scheme (Total 10 Marks)

Item	Marks
SecureRandom class used • Array of 1000 random integers generated and displayed • Main menu option 1works correctly • Search operations in menu options 2 and 3 work as expected	1
Bubble Sort – unsorted array displayed • Bubble sort – simple sorting – complexity • Sorted array displayed • Timing output: Nanoseconds & Milliseconds	1.5
Insertion Sort – unsorted array displayed • Bubble sort – simple sorting – complexity • Sorted array displayed • Timing output: Nanoseconds & Milliseconds	1.5
Selection Sort – unsorted array displayed • Bubble sort – simple sorting – complexity • Sorted array displayed • Timing output: Nanoseconds & Milliseconds	1.5
Merge Sort – unsorted array displayed • Bubble sort – simple sorting – complexity • Sorted array displayed • Timing output: Nanoseconds & Milliseconds	1.5
Quick Sort – unsorted array displayed • Bubble sort – simple sorting – complexity • Sorted array displayed • Timing output: Nanoseconds & Milliseconds	1.5
Lab4 Test class Menu works + properly formatted output + code comments, JavaDoc	1.5

Sample Output: (green is user input)

NOTE: The three dots (...) at the end on the array imply that your randomArr will contain much more data that is, at least 1000 integers.

```
Select your option in the menu below:
1: Initialize and populate an array of 1000 random integers.
2: Perform a recursive binary and linear search.
3: Perform iterative binary and linear search.
```

```
4: Sort the array
5: Quit
>1
Array of randomly generated integers:
[28, 32, 73, 54, 26, 95, 25, 96, 29, 71, 98, 36, 21, 45, 32, 86, 95, 80, 95, 37 ...]

Select your option in the menu below:
1: Initialize and populate an array of 1000 random integers.
2: Perform a recursive binary and linear search.
3: Perform iterative binary and linear search.
4: Sort the array
5: Quit
>2
Please enter an integer value to search:
>45
[28, 32, 73, 54, 26, 95, 25, 96, 29, 71, 98, 36, 21, 45, 32, 86, 95, 80, 95, 37 ...]

45 was found at index position 13 : Recursive Binary Search

Time taken in nanoseconds: 30900
Time taken in milliseconds: 2

45 was found at index position 13 : Recursive Linear Search

Time taken in nanoseconds: 705900
Time taken in milliseconds: 2

Select your option in the menu below:
1: Initialize and populate an array of 1000 random integers.
2: Perform a recursive binary and linear search.
3: Perform iterative binary and linear search.
4: Sort the array
5: Quit
>3
Please enter an integer value to search:
>45
[28, 32, 73, 54, 26, 95, 25, 96, 29, 71, 98, 36, 21, 45, 32, 86, 95, 80, 95, 37 ...]

45 was found at index position 13 : Iterative Binary Search

Time taken in nanoseconds: 45500
Time taken in milliseconds: 18

450 was found at index position 13 : Iterative Linear Search

Time taken in nanoseconds: 655500
Time taken in milliseconds: 10

Select your option in the menu below:
1: Initialize and populate an array of 1000 random integers.
2: Perform a recursive binary and linear search.
3: Perform iterative binary and linear search.
4: Sort the array - Go to submenu
5: Quit

>4

Select a sorting algorithm to sort the data array

     B. Bubble Sort
     I. Insertion Sort
     S. Selection Sort
     M. Merge Sort
     Q. Quick Sort
     R. Return to Main Menu
```

```
>B

[28, 32, 73, 54, 26, 95, 25, 96, 29, 71, 98, 36, 21, 45, 32, 86, 95, 80, 95, 37 ...]

Bubble Sort: Simple sorting algorithm  - O(n2) Complexity   -

[21, 26, 25, 28, 29, 32,32, 36, 37, 45, 54, 71, 73, 80, 86, 95, 95, 95, 96, 98...]

Time taken in nanoseconds: 2745500
Time taken in milliseconds: 68

Select a sorting algorithm to sort the data array

     B. Bubble Sort
     I. Insertion Sort
     S. Selection Sort
     M. Merge Sort
     Q. Quick Sort
     R. Return to Main Menu

>I

[28, 32, 73, 54, 26, 95, 25, 96, 29, 71, 98, 36, 21, 45, 32, 86, 95, 80, 95, 37...]

Insertion Sort: Simple sorting algorithm  - O(n2) Complexity  -

[21, 26, 25, 28, 29, 32,32, 36, 37, 45, 54, 71, 73, 80, 86, 95, 95, 95, 96, 98...]

Time taken in nanoseconds: 2175500
Time taken in milliseconds: 48

Select a sorting algorithm to sort the data array

     B. Bubble Sort
     I. Insertion Sort
     S. Selection Sort
     M. Merge Sort
     Q. Quick Sort
     R. Return to Main Menu

>S

[28, 32, 73, 54, 26, 95, 25, 96, 29, 71, 98, 16, 11, 45, 32, 86, 95, 80, 95, 37...]

Selection Sort: Simple sorting algorithm  - O(n2) Complexity  -

[21, 26, 25, 28, 29, 32,32, 36, 37, 45, 54, 71, 73, 80, 86, 95, 95, 95, 96, 98...]

Time taken in nanoseconds: 2155500
Time taken in milliseconds: 38

Select a sorting algorithm to sort the data array

     B. Bubble Sort
     I. Insertion Sort
     S. Selection Sort
     M. Merge Sort
     Q. Quick Sort
     R. Return to Main Menu

>M

[28, 32, 73, 54, 26, 95, 25, 96, 29, 71, 98, 16, 11, 45, 32, 86, 95, 80, 95, 37...]

Merge Sort: Recursive Divide-And-Conquer - O(n log n) Complexity -

[21, 26, 25, 28, 29, 32,32, 36, 37, 45, 54, 71, 73, 80, 86, 95, 95, 95, 96, 98...]
```

```
Time taken in nanoseconds: 55500
Time taken in milliseconds: 12

Select a sorting algorithm to sort the data array

     B. Bubble Sort
     I. Insertion Sort
     S. Selection Sort
     M. Merge Sort
     Q. Quick Sort
     R. Return to Main Menu

>Q

[28, 32, 73, 54, 26, 95, 25, 96, 29, 71, 98, 16, 11, 45, 32, 86, 95, 80, 95, 37...]

Quick Sort: Recursive Divide-And-Conquer - O(n log n) Complexity -

[21, 26, 25, 28, 29, 32,32, 36, 37, 45, 54, 71, 73, 80, 86, 95, 95, 95, 96, 98...]

Time taken in nanoseconds: 86500

Time taken in milliseconds: 14

>R

Returning to main menu...

Select your option in the menu below:
1: Initialize and populate an array of 1000 random integers.
2: Perform a recursive binary and linear search.
3: Perform iterative binary and linear search.
5: Quit
>5

Exiting...
```