CST8130: Data Structures

Lab 2 – Linear and Binary Search Algorithms: Iterative and Recursive Implementations

Introduction

Searching is a fundamental operation of computer applications and can be performed using either the inefficient linear search algorithm with a time complexity of O (n) or by using the more efficient binary search algorithm with a time complexity of O (log n).

Task Requirements

In this lab, you will write a Java program to search for an integer in an array using binary search algorithm and linear search algorithm - both implemented iteratively and recursively. You will use two built-in methods namely nanoTime() and currentTimeMillis() to compute the time taken by each of the search algorithms. A hint is given below - see (g).

Classes:

1. **BinaryLinearSearch class**

Instance variables

a. Private data members and Scanner object.
b. You may choose to include here all instance variables shared by both recursive and non-recursive methods. These may include shared array index and search key variables.

Methods

a. iterativeBinarySearch (uses iterative/looping construct)
Receives an array of integers and the search key – number to search. If the number is present in the array the method returns its index location and prints the message: '*number ___ found at index ___: Iterative Binary Search'* on the screen. If the number is not found in the array the method returns a sentinel value of -1 and prints the message '*number _ was not found*'.

b. recursiveBinarySearch (uses recursion)
A method that receives an array of randomly generated integers, the first index and last index of the array, and the number to search. The method then recursively searches for the number entered by the user on the keyboard. Returns the index position if the number is present in the array and prints the message '*number ___ found at index ___: Recursive Binary Search*' on the screen. Returns a sentinel value of -1 if the number is not found and prints the message '*number __ was not found*'. (the __ represents the search key).

c. generateRandomInts
Uses the more SecureRandom class to generate random integers between 10 and 100. Returns an array of random numbers: *randomArr*, that is, an array that will be populated with 30 randomly generated integer values mainly in the range of 10 to 100 – boundary values excluded i.e. 10<random integers <100.

This method prints the sorted array of random integers on the screen. **Note**: Both binary search and linear search methods will use same sorted array even though linear search algorithm does not always array to be sorted.

d. remainingElements
This method displays elements remaining each time a half of the array is dropped.

e. iterativeLinearSearch (uses iterative looping construct)Receives an array of integers and the search key – number to search. If the number is present in the array the method returns its index location and prints the message: '*number ___found at index ___: Iterative Linear Search*' on the screen. If the number is not found in the array the method returns a sentinel value of -1 and prints the message 'number _ was not found'.
f. recursiveLinearSearch (uses recursion)A method that receives the array of randomly generated integers (from generateRandomInts - see (c) above), the array size, and the element to search. The method then recursively searches for the number entered by the user on the keyboard. Returns the index position if the number is present in the array and prints the message '*number ___found at index ___: Recursive Linear Search*' on the screen. Returns a sentinel value of -1 if the number is not found and prints the message 'number __ was not found'. (the __ represents the search key).
g. *System.nanotime()* and *System.currentTimeMillis()* methods
Use these two methods of the System class to determine how long the iterative and recursive binary and linear search operations in (a), (b), and take in nanoseconds and milliseconds. Time taken by each operation should be displayed on the screen. Hint: wrap each method in (a) and (b) with the timing methods.

f. You may create one more additional method if you need one - not a graded requirement.

2. **Lab2BinLinSearchTest class**
   Creates a menu as follows:

   Select your option in the menu:
   1. Initialize and populate an array of 20 random integers.
   2. Perform recursive binary and linear search.
   3. Perform iterative binary and linear search.
   4. Exit.

3. **Generate JavaDoc** - to be included in the application. Built-in code comments by you are also required.

Submit a zip folder named as ***Lab2_lnamefname*** of source files and JavaDoc here or through Activities tab.

Grading Scheme (Total 10 Marks)

Item	Marks
BinaryLinearSearch class (correct access modifiers, variables, array, constructors) + **generateRandomInts**	1
iterativeBinarySearch (uses iterative/looping construct)	1
recursiveBinarySearch (recursion implemented)	2
remainingElements() – generates remaining components	1
iterativeLinearSearch (uses iterative/looping construct)	1
recursiveLinearSearch (uses recursion)	1
Timing output for Nanotime() and currentTimeMillis() methods for both LinearSearch and BinarySearch	2
Lab2BinLinsearchTest class Menu works + properly formatted output + programmer comments + JavaDoc	1

Sample Output: Lab 2 CST8130

```
Select your option in the menu below:
1. Initialize and populate an array of 20 random integers.
2. Perform a recursive binary and linear search.
3. Perform iterative binary and linear search.
4. Exit.
1
Array of randomly generated integers:
[12, 14, 26, 27, 34, 37, 37, 38, 49, 55, 57, 60, 62, 68, 69, 69, 70, 70, 73, 74, 75, 81, 82, 84, 86, 90, 90, 96, 97, 99]
Select your option in the menu below:
1. Initialize and populate an array of 20 random integers.
2. Perform a recursive binary and linear search.
3. Perform iterative binary and linear search.
4. Exit.
```

```
2
Please enter an integer value to search: 69
12 14 26 27 34 37 37 38 49 55 57 60 62 68 69 69 70 70 73 74 75 81 82 84 86 90 90 96 97 99
69 was found at index position 15:  recursive Binary Search
Time taken in nanoseconds: 707100
Time taken in milliseconds: 1
69 was found at index position 15:  recursive Linear Search
Time taken in nanoseconds: 907100
Time taken in milliseconds: 2
Select your option in the menu below:
1. Initialize and populate an array of 30 random integers.
2. Perform a recursive binary and linear search.
3. Perform iterative binary and linear search.
4. Exit.
2
Please enter an integer value to search: 68
12 14 26 27 34 37 37 38 49 55 57 60 62 68 69 69 70 70 73 74 75 81 82 84 86 90 90 96 97 99
12 14 26 27 34 37 37 38 49 55 57 60 62 68 69
       49 55 57 60 62 68 69
          62 68 69
68 was found at index position 13: recursive Binary Search
Time taken in nanoseconds: 760000
Time taken in milliseconds: 0
68 was found at index position 13: recursive Linear Search
Time taken in nanoseconds: 860000
Time taken in milliseconds: 2
Select your option in the menu below:
1. Initialize and populate an array of 30 random integers.
2. Perform a recursive binary and linear search.
3. Perform iterative binary and linear search.
4. Exit.
2
Please enter an integer value to search: 71
12 14 26 27 34 37 37 38 49 55 57 60 62 68 69 69 70 70 73 74 75 81 82 84 86 90 90 96 97 99
             70 70 73 74 75 81 82 84 86 90 90 96 97 99
             70 70 73 74 75 81 82
             70 70 73
               73
71 was not found : recursive Binary Search
Time taken in nanoseconds: 1028100
Time taken in milliseconds: 0
71 was not found : recursive Linear Search
Time taken in nanoseconds: 298100
Time taken in milliseconds: 1
Select your option in the menu below:
1. Initialize and populate an array of 30 random integers.
2. Perform a recursive binary and linear search.
3. Perform iterative binary and linear search.
4. Exit.
3
Please enter an integer value to search: 69
12 14 26 27 34 37 37 38 49 55 57 60 62 68 69 69 70 70 73 74 75 81 82 84 86 90 90 96 97 99
69 was found at index position 15: Iterative Binary Search
Time taken in nanoseconds: 424000
Time taken in milliseconds: 1
69 was found at index position 15: Iterative Linear Search
Time taken in nanoseconds: 624000
Time taken in milliseconds: 2
Select your option in the menu below:
1. Initialize and populate an array of 30 random integers.
2. Perform a recursive binary and linear search.
3. Perform iterative binary and linear search.
4. Exit.
3
Please enter an integer value to search: 68
```

```
12 14 26 27 34 37 37 38 49 55 57 60 62 68 69 69 70 70 73 74 75 81 82 84 86 90 90 96 97 99
12 14 26 27 34 37 37 38 49 55 57 60 62 68 69
      49 55 57 60 62 68 69
         62 68 69
68 was found at index position 13: Iterative Binary Search
Time taken in nanoseconds: 1321600
Time taken in milliseconds: 2
68 was found at index position 13: Iterative Linear Search
Time taken in nanoseconds: 2321600
Time taken in milliseconds: 2
Select your option in the menu below:
1. Initialize and populate an array of 30 random integers.
2. Perform a recursive binary and linear search.
3. Perform iterative binary and linear search.
4. Exit.
3
Please enter an integer value to search: 71
12 14 26 27 34 37 37 38 49 55 57 60 62 68 69 69 70 70 73 74 75 81 82 84 86 90 90 96 97 99
         62 68 69 69 70 70 73 74 75 81 82 84 86 90 90 96 97 99
         62 68 69 69 70 70 73 74 75
            70 73 74 75
            70 73
            70
71 was not found: Iterative Binary Search
Time taken in nanoseconds: 545700
Time taken in milliseconds: 0
71 was not found: Iterative Linear Search
Time taken in nanoseconds: 745700
Time taken in milliseconds: 1
Select your option in the menu below:
1. Initialize and populate an array of 30 random integers.
2. Perform a recursive binary and linear search.
3. Perform iterative binary and linear search.
4. Exit.
5
Please choose the option 1 to 4.
Select your option in the menu below:
1. Initialize and populate an array of 30 random integers.
2. Perform a recursive binary and linear search.
3. Perform iterative binary and linear search.
4. Exit.
j
*****Input Mismatch Exception*****
Select your option in the menu below:
1. Initialize and populate an array of 20 random integers.
2. Perform a recursive binary and linear search.
3. Perform iterative binary and linear search.
4. Exit.

4

exiting...
```