CS 250 - Fall 2021 - Lab 06: Array Searching Algorithms

Available: Oct 4, 2021 - Due date: Oct 11, 2021 11:59 PM

Objectives

The objectives of this lab are:

- To practice more on arrays of primitive data types, String, and Objects.
- To practice on the linear search and binary search on arrays
- To practice measuring timing of searching values on large arrays

Prerequisites

- Read sections 9.1 to 9.6 of the free textbook *Java, Java, Java, 3E* and the supplemented slide of lecture 6
- Read online tutorials of Linear Search.
- Read online tutorials of Binary Search.

Part 1: Linear Search on Different Data Types

In many computer systems, a large amount of data are stored. One of the advantages of computer systems is the ability to find such data quickly - SEARCHING. In lecture 6, we study two fundamental searching algorithms: Linear Search and Binary Search.

- **Linear (Sequential) Search** typically starts at the first element in an array or ArrayList and looks through all the items one by one until it either finds the desired value and then it returns the index it found the value at or if it searches the entire array or list without finding the value it returns -1.

- **Binary Search**: can only be used on data that has been sorted or stored in order. It checks the middle of the data to see if that middle value is less than, equal, or greater than the desired value, and then based on the results that it narrows the search. It cuts the search space in half each time.

The first part of this lab is practicing linear search on different arrays of data.

Complete the following tasks:

1. Create a new Java project named `Lab06`. Create a new class `Person`. We will create an array of this `Person` class and perform an array search on it later. Copy the following code of this class to Eclipse:

```
public class Person {
    private String name;        //name of the person
    private int birthYear;      //birth year of the person
    private int ssn;         //the SSN of this person
    private String state;       //the state

    public Person(String name, int by, int ssn, String state) {
        this.name = name;
        this.birthYear = by;
        this.ssn = ssn;
        this.state = state;
    }

    public String getName() {   return name; }
    public int getBirthYear() {return birthYear;  }
    public int getSsn() {       return ssn;  }
    public String getState() { return state; }

    public String toString() {
        return "name: " + name + ", birth year: " + birthYear + ", ssn: "
+ ssn + ", state: " + state;
    }
}
```

2. Create a new class `ArraySearch` with the following starter code of this lab:

```
import java.util.Arrays;
public class ArraySearch {
    /** Search the index of a value in an integer array.
     * @param arrInt - an array containing the elements to be searched.
     * @param target - the element to be found in arrInt.
     * @return the index of the target element in the array if found; -1
otherwise.
    */
    public static int linearSearch(int[] arrInt, int target) {
        //TODO: add code below
    }

    /** Search the index of a value in an array of Person object.
     * @param arrPerson - an array of Person objects containing the
elements to be searched.
     * @param year - the birth year of the Person object to be found in
arrPerson.
```

```
     * @return the index of the target element in the array if found; -1
otherwise.
     */
    public static int linearSearch(Person[] arrPerson, int year) {
        //TODO add code below
    }

    /* the main application method */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
    }
}
```

3. Implement the incomplete `linearSearch(int[] arrInt, int target)` method. Hint: use a for loop that iterates through each integer element in the array `arrInt` (the first parameter), check if the `target` (the second parameter) is equal to the element in the array `arrInt`, and return the founded index. Otherwise, return -1 if not. After completing this method, add code to test this method in the `main()` method as follows:

    - Create an integer array arrInt with the following values: { 9, 4, 13, 43, -17 }
    - Search and print the index of `9` in such an array
    - Search and print the index of `13` in such an array
    - Search and print the index of `-17` in such an array
    - Search and print the index of `99` in such an array

4. Implement the incomplete `linearSearch(Person[] arrPerson, int year)` method. searching for the index of the first Person object in `arrPerson` that has the birth year the same as the `year` (second parameter). This method will return -1 if there is no Person object in the array with that birth year. After completing this method, add code to test the above method in the `main()` method as follows:
    - Create an array list of Person
    - Add the following Person objects to such an array list:
        - "Callum", 1985, 237860451, "MN"
        - "Blake", 1946, 867584562, "WI"
        - "Kayden", 1999, 628457851, "UT"
        - "Colin", 1946, 784583166, "IL"
        - "James", 1965, 789451263, "SD"
    - Now, search and print the index of the Person objects that have the following birth years 1985, 1999, 1965, and 1995 in the above array list `arrPerson.`

If you did it correctly, your program could display a similar output as follows.

```
linearSearch of 9 in arrInt: 0
linearSearch of 13 in arrInt: 2
linearSearch of -17 in arrInt: 4
linearSearch of 99 in arrInt: -1
linearSearch of by=1985 in arrPerson: 0
linearSearch of by=1999 in arrPerson: 2
linearSearch of by=1965 in arrPerson: 4
linearSearch of by=1995 in arrPerson: -1
```

Fig 6.1: The sample output of Part 1's program

Part 2: Binary Search on different data types

The second part of this lab will guide you through implementing binary search on a different array of int, String, and Person. Binary search only works if the array is sorted. We will learn different algorithms to sort an array later. In this lab, similar to the first lab, we will use the `Arrays.sort()` methods provided by Java to sort before performing a binary search. Below are the general steps to perform a binary search on an array:

- Start with `left = 0` and `right = array length - 1` (the index of the last element).
- Calculates the middle index `middle = (left + right) / 2`. Remember that integer division gives an integer result so 1.5 becomes 1.
- It compares the value at the middle index with the target value (the value you are searching for). If the value matches, return the middle index.
- If the target value is less than the value at the middle, it sets `right = middle - one`.
- If the target value is greater than the value at the middle, it sets `left = middle + 1`
- It left is greater than right, the search value couldn't be found and returns -1

The following activities guide you to complete the tasks in this part.

1. Implement a method `binarySearch(int[] arrInt, int target)` in the `ArraySearch` class that uses binary search to search the target integer (second parameter) in the array of integers (first parameter). Copy the following skeleton code to the `ArraySearch` class:

```
/**
 * Use binary search to search for the index target in arrInt.
 * Return -1 if not found
 */
public static int binarySearch(int[] arrInt, int target) {
    //TODO: add code below
}
```

2. Implement this method following the general steps at the beginning of part 2. Then, similar to 1c, test this binarySearch method on the integer array `arrInt` to search and print the indexes of the following elements `9, 13, -17, 99` with the following steps:
    - Sorted the array `arrInt` using the `Arrays.sort()` method
    - Use binarySearch method to search and print the indexes of the following elements `9, 13, -17, 99`

3. Implement the method `public static int binarySearch(Person[] arrPerson, int year)`. This method returns the index of a Person index which has the birth year equal to the search value. When you search for such a Person object, compare the birth year of the object element with the search year value.

4. Test the method `binarySearch(Person[] arrPerson, int year)` in the `main()` method as follows:
    - First, we need to sort the array list of Person `arrPerson` in ascending order of the birth year. You need to follow the method in Lab 04, which uses either Comparable or Comparator interface. You need to create a new Java class named `PersonBirthYearComparator` that implements the interface `Comparator<Person>` and implements the required method `compareTo()`. Refer back to Lab 04 solution to implement this step.
    - Now, sort the arrPerson by calling `Arrays.sort()` method with two arguments: `arrPerson` and `a new instance of PersonBirthYearComparator`.
    - Similar to 1e, but using `binarySearch` method to search and print the index of the Person objects that have the following birth year 1985, 1999, 1965, and 1995 in the sorted array list `arrPerson`

If you did correctly, your program could display a similar output as follows:

```
binarySearch of 9 in arrInt: 2
binarySearch of 13 in arrInt: 3
binarySearch of -17 in arrInt: 0
binarySearch of 99 in arrInt: -1
binarySearch of by=1985 in arrPerson: 3
```

```
binarySearch of by=1999 in arrPerson: 4
binarySearch of by=1965 in arrPerson: 2
binarySearch of by=1995 in arrPerson: -1
```

Fig 6.2: The sample output of Part 2's program

LAB ASSIGNMENTS

Count the number of comparisons in search methods:

- Modify the method `linearSearch(int[] arrInt, int target)` to count and print the total number of comparisons.
- Modify the method `binarySearch(int[] arrInt, int target)` to count and print the total number of comparisons.
- Create an integer array named `arrBigInt` that contains 1000 random integers in the range 0-999. Hint: Use `(int)(Math.random()*1000) to generate a random integer in the range 0-999`. Sort this `arrBigInt` array
- Finally, perform testing of linear search and binary search on `arrBigInt` 5 times, each time does as below:
    - Generate a random integer `target` in the range 0-999
    - Call linearSearch to search the index of the `target` in `arrBigInt` and print the number of comparisons.
    - Call binarySearch to search the index of the `target` in `arrBigInt` and print the number of comparisons.

The following image is a sample output of this Lab Assignment, which gives you an idea of how the modification code works:

```
linearSearch Found 135 with 132 comparisons
binarySearch Found 135 with 7 comparisons
linearSearch Not found 946 with 1000 comparisons
binarySearch Not found 946 with 10 comparisons
linearSearch Not found 63 with 1000 comparisons
binarySearch Not found 63 with 10 comparisons
linearSearch Found 870 with 893 comparisons
binarySearch Found 870 with 10 comparisons
linearSearch Not found 358 with 1000 comparisons
binarySearch Not found 358 with 10 comparisons
```

Fig 6.3: The sample output of Lab Assignment 6

Lab Result Submission

You need to submit the results of the following sections to the D2L assignment item of Lab06:

- Complete the required Lab Assignment

- Compress the whole Eclipse source code of your Lab06 project in zip format using the Export function