**Lab 6**

**Objectives** – To test/time search and sort algorithms

As you add methods to the code given, add documentation similar to what is used on the other methods (comment boxes).

Turn in IntegerList and IntegerListTest, table filled in, and the answer to a question on sorting and searching – see step 9 below.

1. The selection sort is missing two methods (swap and minIndex) – add them or copy the code for a selection sort using one method from last week’s lab or from our class website (Announcements). Use this selection sort code in the sortAscending method.

2. Change the insertion logic to sort in descending order and use it in sortDescending method

3. Make sure the selection sort and bubble sort are increasing order and the insertion sort is decreasing order by creating small arrays, sorting them, and then printing them.

4. Sort an array filled with random numbers with the selection sort for the first timing and then sort it again with the selection sort for the second timing.

5. Sort an array filled with random numbers with the insertion sort for the third timing and then sort it again with the insertion sort for the fourth timing.

6. Sort an array filled with random numbers with the enhanced bubble sort for the fifth timing and then sort it again with the enhanced bubble sort for the sixth timing.

7. **OR** instead of 4, 5, and 6 above, choose 2 (fill in a size), the choose 3, 7, 7, 8, 8, 9, to get times for the first row of the table. Repeat this sequence for each row of the table for obtaining sorting times.

8. Remember that the data in an array must be in ascending sequence to search it with the binary search method. Fill the arrays with sorted data before calling the binary search method. Since it doesn’t matter if you fill the arrays to be used with the sequential search with sorted or unsorted data, you might as well fill the search array with sorted data once, then sequentially search it and then binary search it.

9. Write a short paragraph on sorting and another on searching. Explain why the times changed (or didn’t change) for the different methods and how the algorithms were affected by the larger array sizes.

**Timing Searching and Sorting Algorithms**

In this exercise you will use an IntegerList class (in the file *IntegerList.java*) and a driver (in the file *IntegerListTest.java*) to examine the runtimes of the searching and sorting algorithms. The IntegerListTest class has several options for creating a list of a given size, filling the list with random integers or with already sorted integers, and searching or sorting the list. (NOTE: You may have used a version of these classes in the last lab.) Save these files to your directory and run IntegerListTest a few times to explore the options.

The runtimes of the sorting and searching algorithms can be examined using the Java method *System.currentTimeMillis()*, which returns the current system time in milliseconds. (Note that it returns a long, not an int.) You will have to import java.util.\* to have access to this method. In IntegerListTest, just get the system time immediately before and immediately after you perform any of the searches or sorts. Then subtract the first from the second, and you have the time required for the operation in milliseconds. WARNING: Be sure you are not including any input or output in your timed operations; these are very expensive and will swamp your algorithm times!

Add appropriate calls to System.currentTimeMillis() to your program, run it and fill out the tables below. Note that you will use much larger arrays for the search algorithms than for the sort algorithms; do you see why? Also note that the first couple of times you run a method you might get longer runtimes as it loads the code for that method. Ignore these times and use the "steady-state" times you get on subsequent runs. On a separate sheet, explain the times you see in terms of the known complexities of the algorithms. Remember that the most interesting thing is not the absolute time required by the algorithms, but how the time changes as the size of the input increases (doubles here).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Array Size | Selection Sort (random array) | Selection Sort  (sorted array) | Insertion Sort  (random array) | Insertion Sort  (sorted array) | Bubble Sort  (random array) | Bubble Sort  (sorted array) |
| 10,000 |  |  |  |  |  |  |
| 20,000 |  |  |  |  |  |  |
| 40,000 |  |  |  |  |  |  |
| 80,000 |  |  |  |  |  |  |

|  |  |  |
| --- | --- | --- |
| Array Size | Linear Search (unsuccessful) | Binary Search (unsuccessful) |
| 100,000 |  |  |
| 200,000 |  |  |
| 400,000 |  |  |
| 800,000 |  |  |
| 1,600,000 |  |  |

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// FILE: IntegerList.java

//

// Purpose: Define an IntegerList class with methods to create, fill,

// sort, and search in a list of integers.

//

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import java.util.Scanner;

public class IntegerList

{

int[] list; //values in the list

//------------------------------------------------------------

// Constructor -- takes an integer and creates a list of that

// size. All elements default to value 0.

//------------------------------------------------------------

public IntegerList(int size)

{

list = new int[size];

}

//------------------------------------------------------------

// randomize -- fills the array with randomly generated integers

// between 1 and 100, inclusive

//------------------------------------------------------------

public void randomize()

{

int max = list.length;

for (int i=0; i<list.length; i++)

list[i] = (int)(Math.random() \* max) + 1;

}

//------------------------------------------------------------

// fillSorted -- fills the array with sorted values

//------------------------------------------------------------

public void fillSorted()

{

for (int i=0; i<list.length; i++)

list[i] = i + 2;

}

//------------------------------------------------------------

// print -- prints array elements with indices, one per line

//------------------------------------------------------------

public String toString()

{

String s = "";

for (int i=0; i<list.length; i++)

s += i + ":\t" + list[i] + "\n";

return s;

}

//------------------------------------------------------------

// linearSearch -- takes a target value and returns the index

// of the first occurrence of target in the list. Returns -1

// if target does not appear in the list

//------------------------------------------------------------

public int linearSearch(int target)

{

int location = -1;

for (int i=0; i<list.length && location == -1; i++)

if (list[i] == target)

location = i;

return location;

}

//------------------------------------------------------------

// sortIncreasing -- uses selection sort

//------------------------------------------------------------

public void sortIncreasing()

{

for (int i=0; i<list.length-1; i++)

{

int minIndex = minIndex(list, i);

swap(list, i, minIndex);

}

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// FILE: IntegerListTest.java

//

// Purpose: Provide a menu-driven tester for the IntegerList class.

//

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import java.util.Scanner;

public class IntegerListTest

{

static IntegerList list = new IntegerList(10);

static Scanner scan = new Scanner(System.in);

//------------------------------------------------------

// main -- creates an initial list, then repeatedly prints

// the menu and does what the user asks until they quit

//------------------------------------------------------

public static void main(String[] args)

{

printMenu();

int choice = scan.nextInt();

while (choice != 0)

{

dispatch(choice);

printMenu();

choice = scan.nextInt();

}

}

//------------------------------------------------------

// dispatch -- takes a choice and does what needs doing

//------------------------------------------------------

public static void dispatch(int choice)

{

int loc;

int val;

long time1, time2;

switch(choice)

{

case 0:

System.out.println("Bye!");

break;

case 1:

System.out.println(list);

break;

case 2:

System.out.println("How big should the list be?");

list = new IntegerList(scan.nextInt());

System.out.println("List is created.");

break;

case 3:

list.randomize();

System.out.println("List is filled with random elements.");

break;

case 4:

list.fillSorted();

System.out.println("List is filled with sorted elements.");

break;

case 5:

System.out.print("Enter the value to look for: ");

val = scan.nextInt();

loc = list.linearSearch(val);

if (loc != -1)

System.out.println("Found at location " + loc);

else

System.out.println("Not in list");

break;

case 6:

System.out.print("Enter the value to look for: ");

val = scan.nextInt();

loc = list.binarySearch(val);

if (loc != -1)

System.out.println("Found at location " + loc);

else

System.out.println("Not in list");

break;

case 7:

list.sortIncreasing();

System.out.println("List has been sorted.");

break;

case 8:

list.sortDecreasing();

System.out.println("List has been sorted.");

break;

default:

System.out.println("Sorry, invalid choice");

}

}

//------------------------------------------------------

// printMenu -- prints the user's choices

//------------------------------------------------------

public static void printMenu()

{

System.out.println("\n Menu ");

System.out.println(" ====");

System.out.println("0: Quit");

System.out.println("1: Print the list");

System.out.println("2: Create a new list of a given size");

System.out.println("3: Fill the list with random ints in range 1-length");

System.out.println("4: Fill the list with already sorted elements");

System.out.println("5: Use linear search to find an element");

System.out.println("6: Use binary search to find an element " +

"(list must be sorted in increasing order)");

System.out.println("7: Use selection sort to sort the list into " +

" increasing order");

System.out.println("8: Use insertion sort to sort the list into " +

" decreasing order");

System.out.print("\nEnter your choice: ");

}