CS 1103-01 Programming 2 - AY2023-T1

Solutions for Assignment Unit 1

In this solution, I created a special class "BenchmarkingSortingAlgorithms" in the file BenchmarkingSortingAlgorithms.java. That class contains the code of the solution.

To run this class I created a main class "BenchmarkingSortingAlgorithmsMain" in the file BenchmarkingSortingAlgorithmsMain.java. In that class, I create an object of the class BenchmarkingSortingAlgorithms, above, and as I do, I also run that class code.

While you form your assessment of others' work, remember that the problem posed can be solved in many different forms and that the solution I have provided is only one *recommended* solution.

File 1: "BenchmarkingSortingAlgorithms"

```
/* University of the People - Introduction to Programming 2 - Assignment solution for Unit 1

Part 1: Benchmarking Sorting Algorithms
The same task can take vastly different amounts of time, depending on the algorithm that is used to perform the task. You are familiar with simple sorting algorithms such as insertion sort and selection sort. (See Section 7.4 in the textbook.) While these methods work fine for small arrays, for larger arrays they can take an unreasonable amount of time. The question is whether we can do any better.
Java has some built-in sorting methods. They can be found in the class named Arrays in the package java.util. The one that you will use in this lab is Arrays.sort(A), which sorts the entire array A into increasing order. (Actually, there are different methods for different array base types, but all the methods have the same name and are used in the same way. You will be using an array of ints in this lab.)

You should write a program that does the following:
- Create two arrays of type int[]. Both arrays should be the same size, and the size should be given by a constant in the program so that you can change it easily.
- Fill the arrays with random integers. The arrays should have identical contents, with the same random numbers in both arrays. To generate random integers with a wide range of sizes, you could use (int)(Integer.MAX_VALUE * Math.random()).
- Sort the first array using either Selection Sort or Insertion Sort. You should add the sorting method to your program; you can copy it from Section 7.4 if you want. (It is a good idea to check that you got the sorting method correct by using it to sort a short array and printing out the result.)
- Time how long it takes to sort the array and print out the time.
- Now, sort the second (identical) array using Arrays.sort(). Again, time how long it takes, and print out the time.
You should run your program using array sizes of 1,000, 10,000, and 100,000. Record the sort times. Add a comment to t
of the program that reports the times.
Note: The general method for getting the run time of a code segment is:
long startTime = System.currentTimeMillis();
```

```
doSomething();
long runTime = System.currentTimeMillis() - startTime;
This gives the run time in milliseconds. If you want the time in seconds, you can use runTime/1000.0.
*/


package assignment1java2term3;

import java.util.*;

public class BenchmarkingSortingAlgorithms {
// Compute benchmarks of two different sorting techniques

int maxArraySize=10000; // Array Size
int[] sortingArray1 = new int[maxArraySize]; // First Array
int[] sortingArray2 = new int[maxArraySize]; // Second Array


public BenchmarkingSortingAlgorithms(){
// Class Constructor

for (int i = 0; i < sortingArray1.length; i++) {
// Filling two arrays with the same random numbers.
sortingArray1[i]=(int)(Integer.MAX_VALUE * Math.random());
sortingArray2[i]=sortingArray1[i];
}

long startTimeArray1 = System.currentTimeMillis(); // Start computing time for SelectionSort
selectionSort(sortingArray1); // Sorting Array1 with SelectionSort
long runTimeArray1 = System.currentTimeMillis() - startTimeArray1; // Time to run the SelectionSort

long startTimeArray2 = System.currentTimeMillis();// Start computing time for Arrays.sort
Arrays.sort(sortingArray2); // Sorting Array2 with Arrays.sort
long runTimeArray2 = System.currentTimeMillis() - startTimeArray2; // Time to run the Arrays.sort

System.out.println("SelectionSort time(sec):"+runTimeArray1/1000.0); // Print results
System.out.println("Arrays Sort time(sec):"+runTimeArray2/1000.0); // Print results
}


static void selectionSort(int[] A) {
// Sort A in increasing order, using selection sort
for (int lastPlace = A.length-1; lastPlace > 0; lastPlace--) {
// Find the largest item among A[0], A[1], ...,
// A[lastPlace], and move it into position lastPlace
// by swapping it with the number that is currently
// in position lastPlace.
int maxLoc = 0; // Location of largest item seen so far.
for (int j = 1; j <= lastPlace; j++) {
if (A[j] > A[maxLoc]) {
// Since A[j] is bigger than the maximum we have seen
// so far, j is the new location of the maximum value
// we have seen so far.
maxLoc = j;
```

```
}
int temp = A[maxLoc]; // Swap largest item with A[lastPlace].
A[maxLoc] = A[lastPlace];
A[lastPlace] = temp;
} // end of for loop
}


}
```

File 2: "BenchmarkingSortingAlgorithmsMain"

```
/* University of the People - Introduction to Programming 2 - Assignment solution for Unit 1
Part 1: Benchmarking Sorting Algorithms
The same task can take vastly different amounts of time, depending on the algorithm that is used to perform the task. You are familiar with simple sorting algorithms such as insertion sort and selection sort. (See Section 7.4 in the textbook.) While these methods work fine for small arrays, for larger arrays they can take an unreasonable amount of time. The question is whether we can do any better.
Java has some built-in sorting methods. They can be found in the class named Arrays in the package java.util. The one that you will use in this lab is Arrays.sort(A), which sorts the entire array A into increasing order. (Actually, there are different methods for different array base types, but all the methods have the same name and are used in the same way. You will be using an array of ints in this lab.)
You should write a program that does the following:
- Create two arrays of type int[]. Both arrays should be the same size, and the size should be given by a constant in the program so that you can change it easily.
- Fill the arrays with random integers. The arrays should have identical contents, with the same random numbers in both arrays. To generate random integers with a wide range of sizes, you could use (int)(Integer.MAX_VALUE * Math.random()).
- Sort the first array using either Selection Sort or Insertion Sort. You should add the sorting method to your program; you can copy it from Section 7.4 if you want. (It is a good idea to check that you got the sorting method correct by using it to sort a short array and printing out the result.)
- Time how long it takes to sort the array and print out the time.
- Now, sort the second (identical) array using Arrays.sort(). Again, time how long it takes, and print out the time.
You should run your program using array sizes of 1,000, 10,000, and 100,000. Record the sort times. Add a comment to the top of the program that reports the times.
Note: The general method for getting the run time of a code segment is:
long startTime = System.currentTimeMillis();
doSomething();
long runTime = System.currentTimeMillis() - startTime;
This gives the run time in milliseconds. If you want the time in seconds, you can use runTime/1000.0.
*/

package assignment1java2term3;

public class BenchmarkingSortingAlgorithmsMain {

public static void main(String[] args) {

BenchmarkingSortingAlgorithms SortingTest = new BenchmarkingSortingAlgorithms();
// Create new class of type BenchmarkingSortingAlgorithms

}

}
```

Last modified: Wednesday, 13 May 2020, 12:42 PM

UoPeople Clock (GMT-5)

All activities close on Wednesdays at 11:55 PM, except for Learning Journals/Portfolios which close on Thursdays at 11:55 PM **always following the clock at the top of the page.**

Due dates/times *displayed* in activities will vary with your chosen time zone, however you are still bound to the **11:55 PM GMT-5** deadline.

◄ Mines

Jump to...

Solutions for Exercises Unit 1 ►