**COMP2080**

**Assignment 1 (10%)**

**Due: 6th March 2022 11:30PM**

**Given: 13th February 2022**

**Submission Instructions:**

1. **Fill in your full name and student number in the spaces provided further down this page.**
2. **Fill in all results into the tables provided.**
3. **Paste the code for each class and main program into this document after the question.**
4. **Upload this document to blackboard.**

* You **must** have your name and student ID number commented in all code submitted. **Failure to do this will result in a loss of marks**.
* All submissions should at least compile.
* **Non-compiling assignments will NOT be marked and will be given a grade of 0**.
* All your submissions should be suitably documented.

Name: Minh Nhat Vo

Student ID: 101224717

**Description:**

The purpose of this question is to allow the student to independently compare the run time of various **sorting and searching** algorithms. This will allow the student to get a better understanding of time complexity. It also aims to build an appreciation of the effects that the size and organization of data have on the speed of algorithms.

The sorting algorithms that will be examined are:

1. Selection sort
2. Insertion sort
3. Merge sort
4. Quick Sort

All sorting algorithms must sort the arrays in **descending** order.

Each sorting and searching algorithm will be comparatively run on arrays of the following sizes:

1. fifty (50)
2. one thousand (1,000)
3. ten thousand (10,000)
4. one hundred thousand (100,000)
5. one million (1,000,000)

**Each sort and search is to be tested on the same data set to strive for some consis**tency.

**Sorting Methodology and Requirements:**

1. Create a **core data set** called “***coreData***” which must be a single array of size one million (1,000,000) integers filled with random numbers between one (1) and two million (2,000,000).
2. For each comparative sorting test, four (4) **copies** (one for each sort to be tested) consisting of the same data from “***coreData”*** must be made. For example, if the comparative test is on one thousand (1,000) elements, four arrays of size (1000) should be made and filled with a **copy** of the first one thousand (1000) elements of “***coreData”***.

All recorded times **must** be measured using System.nanoTime() for consistency. An example of how to use it is given below.

long start = System.nanoTime();

//code to be tested

long end = System.nanoTime();

long timeTaken = end – start;

|  |  |
| --- | --- |
| **Test data size** | **Time unit** |
| 50 | nanoseconds |
| 1,000 | nanoseconds |
| 10,000 | nanoseconds |
| 100,000 | nanoseconds |
| 1,000,000 | milliseconds |

**The unit for printing the time must be done according to the following table:**

**Output requirements:**

The **name of each sorting algorithm tested** and **the time it took to sort the data** must be shown grouped by the test data size. For example if the test data size is fifty (50), the time taken for all the sorting algorithms to sort the data must be shown one after another.

Fill in the tables below with the results of your tests.

Sorting Comparison

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Number of Items** | | | | |
| Selection Sort | **50** | **1,000** | **10,000** | **100,000** | **1,000,000** |
| Run 1 | 33734 | 2531567 | 53447426 | 5052303561 | 456088 |
| Run 2 | 29153 | 2838287 | 52377349 | 5140945031 | 569891 |
| Run 3 | 24353 | 2809207 | 53187708 | 5207744387 | 497714 |
| Run 4 | 24315 | 2527113 | 51819854 | 5110656198 | 528857 |
| Run 5 | 25237 | 3001056 | 52775624 | 4957325469 | 544302 |
| Average | 27358 | 2757646 | 43395592.2 | 5093794929 | 519370.4 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Number of Items** | | | | |
| Insertion Sort | **50** | **1,000** | **10,000** | **100,000** | **1,000,000** |
| Run 1 | 20887 | 1830895 | 13305399 | 789953704 | 101994 |
| Run 2 | 15230 | 2274567 | 12789848 | 805764689 | 92047 |
| Run 3 | 13930 | 1982595 | 18012293 | 814489956 | 86842 |
| Run 4 | 14866 | 1996890 | 20558764 | 816033302 | 87594 |
| Run 5 | 14078 | 2078362 | 13164306 | 787986467 | 86803 |
| Average | 15798.2 | 2032661.8 | 15566122 | 682845623.6 | 91056 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Number of Items** | | | | |
| Merge Sort | **50** | **1,000** | **10,000** | **100,000** | **1,000,000** |
| Run 1 | 17995 | 907522 | 13459387 | 1001863618 | 99030 |
| Run 2 | 14077 | 1258748 | 13179345 | 1026436708 | 108095 |
| Run 3 | 13057 | 1166838 | 12435498 | 1043601421 | 106670 |
| Run 4 | 17716 | 920291 | 13221451 | 861598729 | 106447 |
| Run 5 | 13010 | 1158147 | 12865974 | 1089116083 | 98125 |
| Average | 15171 | 1082309.2 | 13032331 | 1004523312 | 103673.4 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Number of Items** | | | | |
| Quick Sort | **50** | **1,000** | **10,000** | **100,000** | **1,000,000** |
| Run 1 | 15194 | 906133 | 11138344 | 877768761 | 91413 |
| Run 2 | 11965 | 1018585 | 10672737 | 960350188 | 92756 |
| Run 3 | 11108 | 1048101 | 10715951 | 967769649 | 90775 |
| Run 4 | 11870 | 980072 | 10923427 | 1131880282 | 97452 |
| Run 5 | 10795 | 1133809 | 11141261 | 961988293 | 118759 |
| Average | 12186.4 | 1017340 | 10918344 | 979951434.6 | 491155 |

**Searching Methodology and Requirements:**

1. Create a **core data set** called “***coreData***” which must be a single array of size one million (1,000,000) integers filled with random numbers between one (1) and two million (2,000,000). **This array must then be sorted in descending order**. (You may use the sorted array from the first part).
2. For each comparative search test, you must search for the number 2,500,000 (two million five hundred thousand). This number does not exist in the array and represents the worst case for both searches. You must repeat the tests 5 times with binary search and linear search, timing how long it takes to complete for each dataset size. Place the results in the table below.

All recorded times **must** be measured using System.nanoTime() for consistency.

Fill in the tables below with the results of your tests.

Searching Comparison

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Number of Items** | | | | |
| Binary Search | **50** | **1,000** | **10,000** | **100,000** | **1,000,000** |
| Run 1 | 10213 | 8530 | 9686 | 26008 | 0.005 |
| Run 2 | 10133 | 7829 | 10195 | 20656 | 0.006 |
| Run 3 | 10150 | 11361 | 9399 | 40722 | 0.006 |
| Run 4 | 4401 | 8550 | 9150 | 25570 | 0.0058 |
| Run 5 | 5218 | 7567 | 6834 | 28344 | 0.007 |
| Average | 8023 | 8767.4 | 9052.8 | 28260 | 0.006 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Number of Items** | | | | |
| Linear Search | **50** | **1,000** | **10,000** | **100,000** | **1,000,000** |
| Run 1 | 10072 | 74806 | 417527 | 612627 | 4.19 |
| Run 2 | 14825 | 43210 | 534654 | 287660 | 4.3 |
| Run 3 | 11953 | 46244 | 487001 | 1078580 | 4.23 |
| Run 4 | 8659 | 61571 | 342612 | 1095929 | 4.09 |
| Run 5 | 7005 | 41257 | 350784 | 906498 | 4.45 |
| Average | 10502.8 | 53417.6 | 426515.6 | 796258.8 | 4.25 |

PASTE YOUR CLASSES FOR THE ASSIGNMENT AND MAIN HERE:

Your code must have your name commented in each new class.

**Class name: coreData**

package as1;

//Minh Nhat Vo - 101224717

public class coreData {

private int[] data\_array;

private int counter;

private int dataSize;

public coreData(int size)

{

dataSize = size;

counter = 0;

data\_array = new int[dataSize];

}

public void showData(int quantity)

{

if(quantity <= counter){

for(int order = 0; order < quantity ; order++)

{

System.out.print(data\_array[order] + " ");

}

System.out.println("");

}

}

public void addData(int item)

{

if(counter < dataSize)

{

data\_array[counter] = item;

counter++;

}

}

// Sorting the arrays in descending order.

public void \_Sort\_By\_Selection()

{

for(int starter = 0; starter < counter - 1; starter++)

{

int smallestLocker = starter;

for(int index = starter + 1; index < counter; index++ )

{

if(data\_array[index] > data\_array[smallestLocker]){

smallestLocker = index;

}

}

int tempLocker = data\_array[starter];

data\_array[starter] = data\_array[smallestLocker];

data\_array[smallestLocker] = tempLocker;

}

}

// Sorting the arrays in descending order.

public void \_Sort\_By\_Insertion()

{

for(int start = 1; start < counter; start++){

int temp = data\_array[start];

int prePos = start - 1;

while(prePos >= 0 && data\_array[prePos] < temp){

data\_array[prePos + 1] = data\_array[prePos];

prePos--;

}

data\_array[prePos + 1] = temp;

}

}

// Sorting the arrays in descending order.

public void \_Sort\_By\_Merge(){

mergeOperator(0, counter - 1);

}

public void mergeOperator(int left, int right){

if (left < right){

int mid = (left + right)/2;

mergeOperator(left, mid);

mergeOperator(mid + 1, right);

merge(left, mid, right);

}

}

public void merge(int left, int mid, int right){

int[] leftArray = new int[mid - left +1];

int[] rightArray = new int[right - mid];

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

leftArray[i] = data\_array[left + i];

}

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

rightArray[i] = data\_array[mid + 1 + i];

}

int pOfLeft = 0, pOfRight = 0, pM = left;

while(pOfLeft < leftArray.length && pOfRight < rightArray.length){

if(leftArray[pOfLeft] > rightArray[pOfRight]){

data\_array[pM] = leftArray[pOfLeft];

pOfLeft++;

}else{

data\_array[pM] = rightArray[pOfRight];

pOfRight++;

}

pM++;

}

while(pOfLeft < leftArray.length){

data\_array[pM] = leftArray[pOfLeft];

pOfLeft++;

pM++;

}

while(pOfRight < rightArray.length){

data\_array[pM] = rightArray[pOfRight];

pOfRight++;

pM++;

}

}

public void \_Sort\_By\_Quick(){

quickOperator(0, counter - 1);

}

private void quickOperator(int lo, int hi){

if(lo < hi){

int pivot = data\_array[hi];

quickOperator(lo, pivot - 1);

quickOperator(pivot + 1, hi);

}

}

private void swap(int location1,int location2){

int temp =data\_array[location1];

data\_array[location1] = data\_array[location2];

data\_array[location2 ]= temp;

}

private int partition(int lo, int hi){

int compPoint = data\_array[hi];

int marker= lo-1,tempPoint;

for (int presPos=lo;presPos<hi;presPos++){

if (data\_array[presPos] > compPoint){

marker++;

tempPoint =data\_array[marker];

data\_array[marker] = data\_array[presPos];

data\_array[presPos ]= tempPoint;

}

}

tempPoint = data\_array[marker+1];

data\_array[marker+1] = data\_array[hi];

data\_array[hi] = tempPoint;

return marker + 1;

}

// Make Ordered and Descending Array

public void addInorder(int item){

if(counter<dataSize){

data\_array[counter] = item;

counter++;

}

int temp = data\_array[counter-1];

int presPos =counter-2;

while (presPos>=0 && data\_array[presPos]<temp){

data\_array[presPos+1]=data\_array[presPos];

presPos--;

}

data\_array[presPos+1]=temp;

}

//

public int binarySearch(int key){

int lo=0,hi=counter-1,mid;

while (lo<=hi){

mid = (lo+hi)/2;

if (data\_array[mid] == key)

return mid;

if (data\_array[mid] > key)

lo=mid+1;

else

hi=mid-1;

}

return -1;

}

public int linearSearch(int key){

int res;

for(int x=0; x<counter; x++){

if(key == data\_array[x])

return res = x;

}

return -1;

}

}

**Main:**

package as1;

//Minh Nhat Vo - 101224717

import java.util.\*;

public class As1 {

public static void main(String[] args) {

Random rand = new Random();

// int array\_50 = 50;

// int array\_1000 = 1000;

// int array\_10000 = 10000;

// int array\_100000 = 100000;

int array\_1000000 = 1000000;

// coreData srt = new coreData(array\_1000000);

// coreData srt1 = new coreData(array\_1000000);

// coreData srt2 = new coreData(array\_1000000);

// coreData srt3 = new coreData(array\_1000000);

coreData str4 = new coreData(array\_1000000);

coreData str5 = new coreData(array\_1000000);

for(int x=0; x<array\_1000000; x++){

int val = rand.nextInt(2000000 - 1 + 1) + 1;

// srt.addData(val);

// srt1.addData(val);

// srt2.addData(val);

// srt3.addData(val);

str4.addInorder(val);

str5.addInorder(val);

}

// srt.showData(10);

//

// long start = System.nanoTime();

// srt.\_Sort\_By\_Selection();

// long end = System.nanoTime();

// long timeTaken = end - start;

// System.out.println("\nThe SELECTION sort took "+ timeTaken +" nanoseconds.\n");

// srt.showData(10);

//

// System.out.println("-------------------------------------------");

//

//

// srt1.showData(10);

//

// start = System.nanoTime();

// srt1.\_Sort\_By\_Insertion();

// end = System.nanoTime();

// timeTaken = end - start;

// System.out.println("\nThe INSERTION sort took " + timeTaken + " nanoseconds.\n");

// srt1.showData(10);

//

// System.out.println("-------------------------------------------");

//

// srt2.showData(10);

//

// start = System.nanoTime();

// srt2.\_Sort\_By\_Insertion();

// end = System.nanoTime();

// timeTaken = end - start;

// System.out.println("\nThe MERGE sort took " + timeTaken + " nanoseconds.\n");

// srt2.showData(10);

//

// System.out.println("-------------------------------------------");

//

// srt3.showData(10);

//

// start = System.nanoTime();

// srt3.\_Sort\_By\_Insertion();

// end = System.nanoTime();

// timeTaken = end - start;

// System.out.println("\nThe QUICK sort took " + timeTaken + " nanoseconds.\n");

// srt3.showData(10);

str4.showData(10);

System.out.print("Enter a value to search: ");

Scanner sc = new Scanner(System.in);

int userChoice = sc.nextInt();

long start = System.nanoTime();

int ind = str4.binarySearch(userChoice);

long end = System.nanoTime();

long timeTaken = end - start;

System.out.println("Binary search: "+ ind + " took " + timeTaken + " nanoseconds.\n");

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

start = System.nanoTime();

int ind2 = str5.linearSearch(userChoice);

end = System.nanoTime();

timeTaken = end - start;

System.out.println("Linear search: "+ind2 + " took " + timeTaken + " nanoseconds.\n");

}

}