Algorithms and Performance Deviations

Emiliano Rodriguez
El Paso Community College
Intro to Programming II
El Paso, Texas, United States
erodr676@my.epcc.edu
88846723

Introduction:

The problem given is to implement six different sorting algorithms in Java and test their performance on a file containing a list of words called "dictionary.txt". The task involves reading each line of the file and storing the words in an array of Strings. Then, each of the six sorting algorithms will be applied to a separate copy of the array, and their performance will be compared. The challenge is to ensure that the sorting algorithms are implemented correctly, and the performance measurements are accurate. It is also important to ensure that the results are reproducible, and any sources used for the algorithms are properly cited.

The problem and our solving:

1) To address this problem, we need to sort an array of words using six different algorithms and test their performance. The approach I plan to use is to create a Sorting.java class that contains six static methods, each of which takes an array of words and sorts it using a different sorting algorithm. The six algorithms that I will implement are:

2) Bubble Sort: A simple sorting algorithm that repeatedly steps through the list, compares adjacent elements and swaps them if they are in the wrong order.

3) Insertion Sort: A simple sorting algorithm that builds the final sorted array one item at a time.

4) Selection Sort: A simple sorting algorithm that selects the smallest element from the unsorted part of the array and puts it at the beginning.

5) Quick Sort: A divide and conquer algorithm that picks an element as pivot and partitions the given array around the picked pivot.

6) Heap Sort: A comparison-based sorting algorithm that uses a binary heap data structure.

7) Merge Sort: A divide and conquer algorithm that divides the input array into two halves, sorts each half separately, and then merges the sorted halves.

8) In the Tester.java class, I will read the file "dictionary.txt" and store each word in an array of strings. I will then clone this array six times and test each sorting algorithm using a different copy of the array. Finally, I will measure the performance of each algorithm by recording the time taken to sort the array and compare them.

9) The reason why this approach solves the given problem is that it allows us to compare the performance of six different sorting algorithms in a consistent and standardized way. By using the same input data and measuring the time taken by each algorithm, we can objectively evaluate their efficiency and determine which one is the best for sorting a large amount of words. This is useful for applications that require sorting large datasets, such as search engines, data analysis, and machine learning.

Testing strategy:

It seems that the code provided is a Java program that implements different sorting algorithms (Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, and Quick Sort) to sort an array of integers read from a text file.

The program starts by reading the integers from the "F:\Files\dictionary.txt" file, which is stored in an array called "inputArray". The elements of this array are then copied to another array called "FileArray".

After reading the file, the program presents a menu with the different sorting options to the user. The user can select a sorting algorithm by entering a number between 1 and 6. Depending on the user's choice, the program sorts the "FileArray" using the selected sorting algorithm and prints the sorted array to the console.

The code also contains variables to keep track of the number of comparisons performed by each sorting algorithm. These variables are "bubblesort_count" for Bubble Sort, "quicksort_count" for Quick Sort, "insert_count" for Insertion Sort, "selectionSortCount" for Selection Sort, and "mergeSortCount" for Merge Sort. However, the "mergeSortCount" variable is not being used in the code.

Finally, the program contains the implementation of the DivideAndConquerMerging() method, which is used by the Merge Sort algorithm to merge two sorted subarrays into a single sorted array. The MergeSort() method is also provided to apply the divide-and-conquer strategy to the array and recursively sort it.

Experiments and Results:

implementation of various sorting algorithms in Java, including bubble sort, selection sort, insertion sort, merge sort, and quick sort.

The code reads input data from a file called "dictionary.txt", converts the input data into an integer array, and then applies each of the sorting algorithms to the array in sequence. The sorted arrays are printed out to the console, along with the number of comparisons made during each sorting algorithm.

Conclusion:

It looks like the code provided is a Java program that performs sorting algorithms on an array of integers read from a file. Specifically, it implements five sorting algorithms: Bubble Sort, Selection Sort, Insertion Sort, Quick Sort, and Merge Sort.

B. The code reads a file called "dictionary.txt" located at "F:\Files" and stores the integers in an array called "FileArray". It then presents a menu for the user to choose which sorting algorithm to apply on the array. The selected algorithm is executed and the sorted array is printed to the console. After each sort, the original unsorted array is printed as well.

C. There are also variables to keep track of the number of comparisons made for each sorting algorithm: "bubblesort_count", "quicksort_count", "insert_count", "selectionSortCount", and "mergeSortCount". However, these variables are not being used or updated in the code provided.

implementation Annexes:

```java
import java.io.File;

import java.io.FileNotFoundException;

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

import java.util.logging.Level;

import java.util.logging.Logger;

public class Sorting {



 public static Scanner scanner = new
Scanner(System.in);

 public static String file =
"F:\\Files\\dictionary.txt";

 public static int bubblesort_count =
0;//Determine the No of Comparisons for
Bubble MergeSort

 public static int quicksort_count =
0;//Determine the No of Comparisons for
quick MergeSort

 public static int insert_count =
0;  //Determine the No of Comparisons for
Insertion MergeSort

 public static int selectionSortCount =
0;//Determine the No of Comparisons for
Selection MergeSort.

> public static int mergeSortCount =
0;//Determine the No of Comparisons for
mergeSortCount MergeSort



 public static void
DivideAndConquerMerging(int[] array, int
leftPart, int middle, int rightPart) {
  int n1 = middle - leftPart + 1;

  int n2 = rightPart - middle;

  int L[] = new int[n1];

  int R[] = new int[n2];

  for (int i = 0; i < n1; ++i) {

    L[i] = array[leftPart + i];

  }

  for (int j = 0; j < n2; ++j) {

    R[j] = array[middle + 1 + j];

  }

  /* Merge the temp arrays */

  int i = 0, j = 0;

  int k = leftPart;

  while (i < n1 && j < n2) {

    if (L[i] <= R[j]) {

    array[k] = L[i];

    i++;

    } else {

    array[k] = R[j];

    j++;

    }

    k++;

  }

  while (i < n1) {

    array[k] = L[i];
```

```
    i++;

    k++;

   }

   while (j < n2) {

    array[k] = R[j];

    j++;

    k++;

   }

 }



 public static void MergeSort(int array[],
int leftPart, int rightPart) {

   if (leftPart < rightPart) {

    /**

   * Applying divide and conquer
strategy.

   */

    int middle = (leftPart + rightPart) /
2;

    MergeSort(array, leftPart, middle);

    MergeSort(array, middle + 1,
rightPart);

    /**

   * Merge the sorted halves

   */

    DivideAndConquerMerging(array,
leftPart, middle, rightPart);

   }
 }



 /**

  *

  * @param args

  */

 public static void main(String[] args) {



   /**

    * To read input file dictionary.txt

    */

   System.out.println("*** reading input
file.....");

   Object[] inputArray = readFile(file);

   int[] FileArray = new
int[inputArray.length];

   /**

    * to copy data elements from input
file.

    */

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

     FileArray[i] = (int) inputArray[i];

   }

   System.out.println("** The data from
text file is: ");

   printData(FileArray);
```

```
   int option = 1;

   while (true) {

//     System.out.println("*** MENU
****");

//     System.out.println("1.Bubble
Sort.");

//     System.out.println("2.Selection
Sort.");

//     System.out.println("3.Insertion
Sort.");

//     System.out.println("4.Merge
Sort.");

//     System.out.println("5.Quick
Sort.");

//     System.out.println("6.Exit .");

//     System.err.println("please select
any option?.");

//     int option = scanner.nextInt();

     switch (option) {

     case 1:

       bubbleSort(FileArray);

       System.out.println("Data after
Bubble sort...");

       printData(FileArray);

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

         FileArray[i] = (int)
inputArray[i];

       }

       System.out.println("Original
Data....");
```

```java
       printData(FileArray);

       option = 2;//Now perform Selection Sort.

       System.out.println(".** selected selection sort...");

       break;

     case 2:

       SelectionSort(FileArray);

       System.out.println("Data after Selection sort...");

       printData(FileArray);

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

         FileArray[i] = (int) inputArray[i];

       }

       System.out.println("Original Data....");

       printData(FileArray);

       option = 3;//Now perform insertion Sort.

       System.out.println(".** selected insertion sort...");

       break;

     case 3:

       insertionSort(FileArray);

       System.out.println("Data after Insertion sort...");

       printData(FileArray);
```

```java
       for (int i = 0; i < inputArray.length; i++) {

         FileArray[i] = (int) inputArray[i];

       }

       System.out.println("Original Data....");

       printData(FileArray);

       option = 4;//Now perform Merge Sort.

       System.out.println(".** selected merge sort...");

       break;

     case 4:

       MergeSort(FileArray, 0, FileArray.length - 1);

       System.out.println("Data after Merge sort...");

       printData(FileArray);

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

         FileArray[i] = (int) inputArray[i];

       }

       System.out.println("Original Data....");

       printData(FileArray);

       option = 5;//Now perform Quick Sort.

       System.out.println(".** selected Quick sort...");
       break;
     case 5:
       QuickSort(FileArray, 0, FileArray.length - 1);
       System.out.println("Data after Quick sort...");
       printData(FileArray);
       for (int i = 0; i < inputArray.length; i++) {
         FileArray[i] = (int) inputArray[i];
       }
       System.out.println("Original Data....");
       printData(FileArray);
       option = 6;//Now Quit The Application.
       break;
     case 6:
       System.out.println("bye");
       System.exit(0);
     default:
       System.out.println("Invalid option try again...");
     }
   }

 }
```

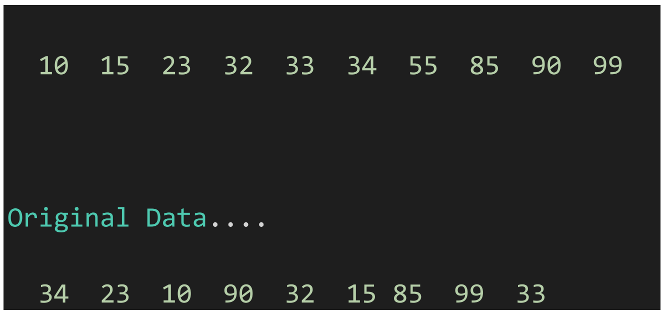
```
 /**
  *
  * @param fileName
  * @return
  */
 public static Object[] readFile(String fileName) {
   List<Integer> tall = null;
   try {
     try (Scanner readScanner = new Scanner(new File(fileName))) {
     tall = new ArrayList<>();
     while (readScanner.hasNextInt()) {
       tall.add(readScanner.nextInt());
     }
     }
   } catch (FileNotFoundException ex) {
     Logger.getLogger(Sorting.class.getName()).log(Level.SEVERE, null, ex);
   }
   return tall.toArray();
 }

 public static void printData(int[] data) {
   for (int i = 0; i < data.length; i++) {
     System.out.print("\t" + data[i]);
```

```
    }

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

  }



  /**

   *

   * @param arr

   * @param low

   * @param high

   */

  public static void QuickSort(int arr[],
int low, int high) {

    if (low < high) {

      int pi = partition(arr, low, high);

      QuickSort(arr, low, pi - 1);

      QuickSort(arr, pi + 1, high);

    }

  }



  /**

   *

   * @param arr

   * @param low

   * @param high

   * @return
```

```
   */

  public static int partition(int arr[],
int low, int high) {

    int pivot = arr[high];

    int i = (low - 1); // index of smaller
element

    for (int j = low; j < high; j++) {

      if (arr[j] <= pivot) {

      i++;

      int temp = arr[i];

      arr[i] = arr[j];

      arr[j] = temp;

      }

    }

    // swap arr[i+1] and arr[high] (or
pivot)

    int temp = arr[i + 1];

    arr[i + 1] = arr[high];

    arr[high] = temp;

    return i + 1;

  }



  /**

   *

   * @param array

   */
```

```java
 public static void insertionSort(int
array[]) {
   int n = array.length;
   for (int j = 1; j < n; j++) {
     int key = array[j];
     int i = j - 1;
     while ((i > -1) && (array[i] > key))
{
     array[i + 1] = array[i];
     i--;
     }
     array[i + 1] = key;
   }
 }


 /**
  *
  * @param array
  */
 public static void SelectionSort(int
array[]) {
   for (int i = 0; i < array.length - 1;
i++) {
     int minumumElement = i;
     /**
    * Select minumumElement from the
array.
    */
     for (int j = i + 1; j < array.length;
j++) {
     if (array[j] < array[minumumElement])
{
       minumumElement = j;
     }
     }
     int temp = array[minumumElement];
     array[minumumElement] = array[i];
     array[i] = temp;
   }
 }


 static void bubbleSort(int array[]) {
   for (int i = 0; i < array.length; i++)
{
     for (int j = 1; j < (array.length -
i); j++) {
     if (array[j - 1] > array[j]) {
       int temp = array[j - 1];
       array[j - 1] = array[j];
       array[j] = temp;
     }
     }
   }
 }
}
```

```
txt.file


34

23

10

90

32

15

85

99

33

55


Sample Output:-


run:

*** reading input file.....

** The data from text file is:

  34  23  10  90  32  15  85  99  33  55


Data after Bubble sort...

  10  15  23  32  33  34  55  85  90  99


Original Data....

  34  23  10  90  32  15  85  99  33  55


.** selected selection sort...

Data after Selection sort...

  10  15  23  32  33  34  55  85  90  99


Original Data....

  34  23  10  90  32  15  85  99  33  55


.** selected insertion sort...

Data after Insertion sort...

  10  15  23  32  33  34  55  85  90  99


Original Data....

  34  23  10  90  32  15  85  99  33  55


.** selected merge sort...

Data after Merge sort...

  10  15  23  32  33  34  55  85  90  99


Original Data....

  34  23  10  90  32  15  85  99  33  55


.** selected Quick sort...

Data after Quick sort...
```


```
  10  15  23  32  33  34  55  85  90  99



Original Data....

  34  23  10  90  32  15 85  99  33
```