HACETTEPE UNIVERSITY COMPUTER SCIENCE ENGINEERING DEPARTMENT

BBM204 SOFTWARE LABORATORY II

ASSIGNMENT-1

Tas : Selim Yılmaz, Levent Karacan, Merve Özdeş

Subject	**: Analyzing the Complexity Of Sorting Algorithtms**
Due Date	: 15/03/2018
Student ID	: 21228817
Name	: Ali Utku ÜNLÜ
E-Mail	: aliutkuunlu@gmail.com

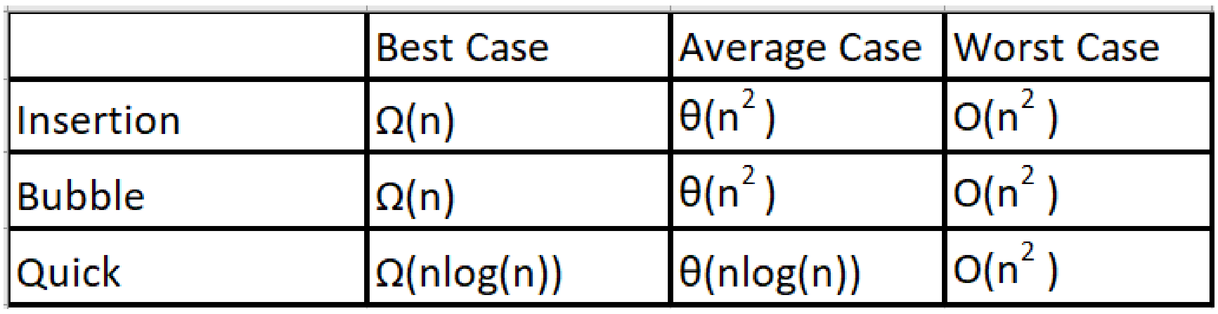
1- Problem Definition

We have an input file which contains traffic data flows.

In this assignment we need to sort the given input file according to given feature index which is given by command line argument. The feature index must be in (7< index<= 84) interval.

For sorting this file, we need to choose 3 different sorting algorithms. Which are; Selection Sort, Bubble Sort, Insertion Sort, Heap Sort, Quick Sort, Merge Sort, Radix Sort.

I chose Insertion, Bubble and Quick Sort algorithms. This sorting algorithms have their complexities.

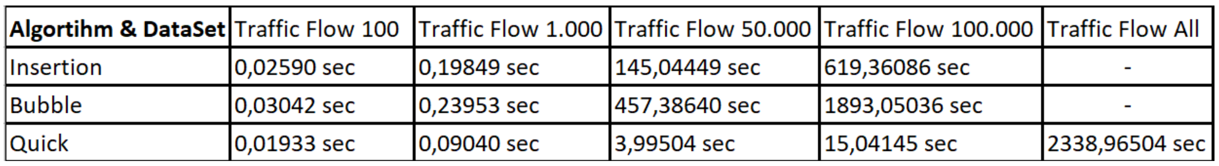


	Best Case	Average Case	Worst Case
Insertion	$\Omega(n)$	$\theta(n^2)$	$O(n^2)$
Bubble	$\Omega(n)$	$\theta(n^2)$	$O(n^2)$
Quick	$\Omega(n\log(n))$	$\theta(n\log(n))$	$O(n^2)$

While sorting I held the input file like this;

I read the input file line by line and I kept every single line in an arraylist(main). When I split the line corresponding to comma, I kept to tokens in arraylist also. When I sent the main arraylist to sorting method with the given feature index. Sorting method must sort all the lines corresponding to comparing element.

2- Finding Results



Algortihm & DataSet	Traffic Flow 100	Traffic Flow 1.000	Traffic Flow 50.000	Traffic Flow 100.000	Traffic Flow All
Insertion	0,02590 sec	0,19849 sec	145,04449 sec	619,36086 sec	-
Bubble	0,03042 sec	0,23953 sec	457,38640 sec	1893,05036 sec	-
Quick	0,01933 sec	0,09040 sec	3,99504 sec	15,04145 sec	2338,96504 sec

For insertion sort and bubble sort; due to its complexity is too high, when data is get bigger run time increased quadratically. But insertion sort has better performance against the bubble sort.

For quick sort; run time is not as high as the other algorithms. But we need extra memory to implement this algorithm.

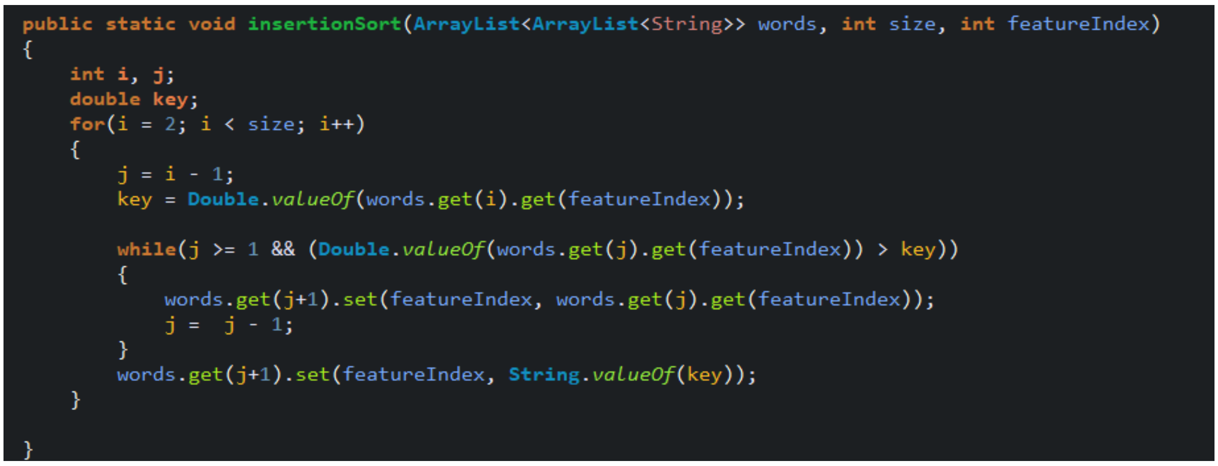
For small datasets it's not important that which dataset you will choose. But when dataset is getting bigger we need to choose Quick sort between these algorithms.

It's a general concept if you want to gain speed you need to loose memory. If you want to save memory you have to accept that low speed.

In the given table above when I run the program with bubble sort and insertion sort it took too much time and never terminate themselves for Traffic Flow All. I terminated myself. But my observation is, they need hours to terminate themselves.

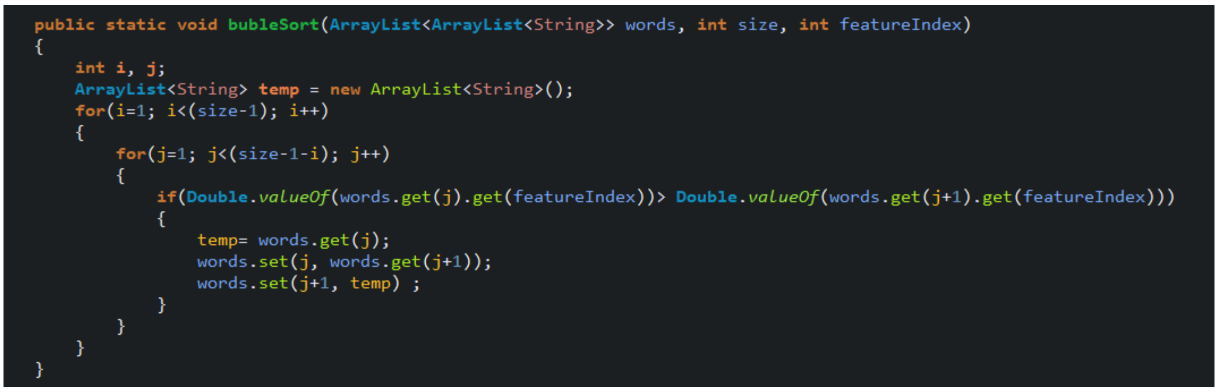
3- Algorithms

- Insertion Sort:



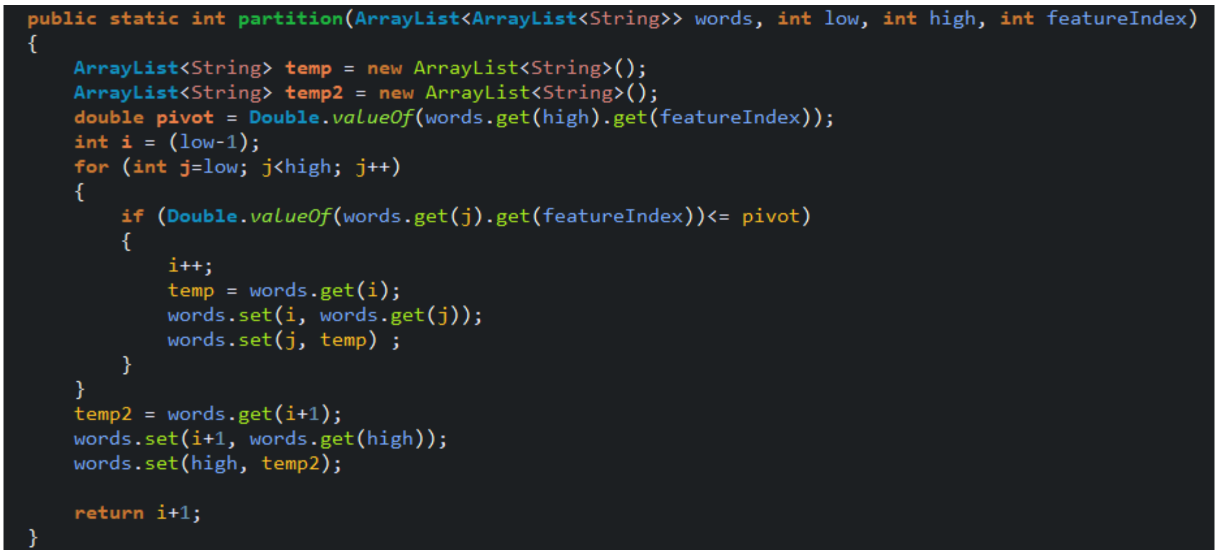
```java
public static void insertionSort(ArrayList<ArrayList<String>> words, int size, int featureIndex)
{
    int i, j;
    double key;
    for(i = 2; i < size; i++)
    {
        j = i - 1;
        key = Double.valueOf(words.get(i).get(featureIndex));

        while(j >= 1 && (Double.valueOf(words.get(j).get(featureIndex)) > key))
        {
            words.get(j+1).set(featureIndex, words.get(j).get(featureIndex));
            j =  j - 1;
        }
        words.get(j+1).set(featureIndex, String.valueOf(key));
    }

}
```

- Bubble Sort:



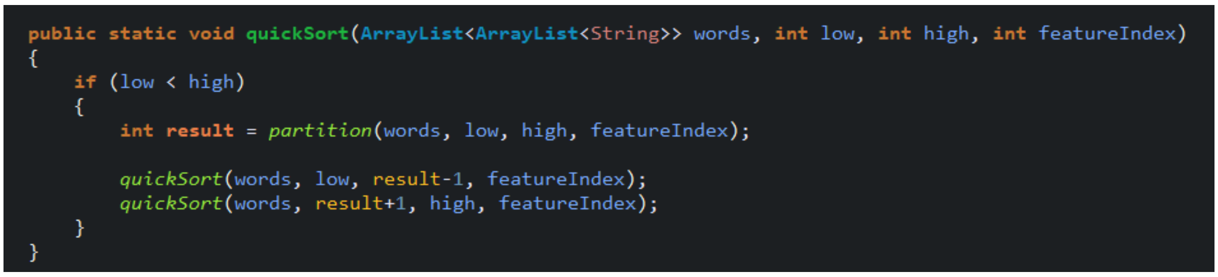
```java
public static void bubleSort(ArrayList<ArrayList<String>> words, int size, int featureIndex)
{
    int i, j;
    ArrayList<String> temp = new ArrayList<String>();
    for(i=1; i<(size-1); i++)
    {
        for(j=1; j<(size-1-i); j++)
        {
            if(Double.valueOf(words.get(j).get(featureIndex))> Double.valueOf(words.get(j+1).get(featureIndex)))
            {
                temp= words.get(j);
                words.set(j, words.get(j+1));
                words.set(j+1, temp) ;
            }
        }
    }
}
```

- Quick Sort:



```java
public static int partition(ArrayList<ArrayList<String>> words, int low, int high, int featureIndex)
{
    ArrayList<String> temp = new ArrayList<String>();
    ArrayList<String> temp2 = new ArrayList<String>();
    double pivot = Double.valueOf(words.get(high).get(featureIndex));
    int i = (low-1);
    for (int j=low; j<high; j++)
    {
        if (Double.valueOf(words.get(j).get(featureIndex))<= pivot)
        {
            i++;
            temp = words.get(i);
            words.set(i, words.get(j));
            words.set(j, temp) ;
        }
    }
    temp2 = words.get(i+1);
    words.set(i+1, words.get(high));
    words.set(high, temp2);

    return i+1;
}
```

In partition part we are allocating extra memory.



```java
public static void quickSort(ArrayList<ArrayList<String>> words, int low, int high, int featureIndex)
{
    if (low < high)
    {
        int result = partition(words, low, high, featureIndex);

        quickSort(words, low, result-1, featureIndex);
        quickSort(words, result+1, high, featureIndex);
    }
}
```