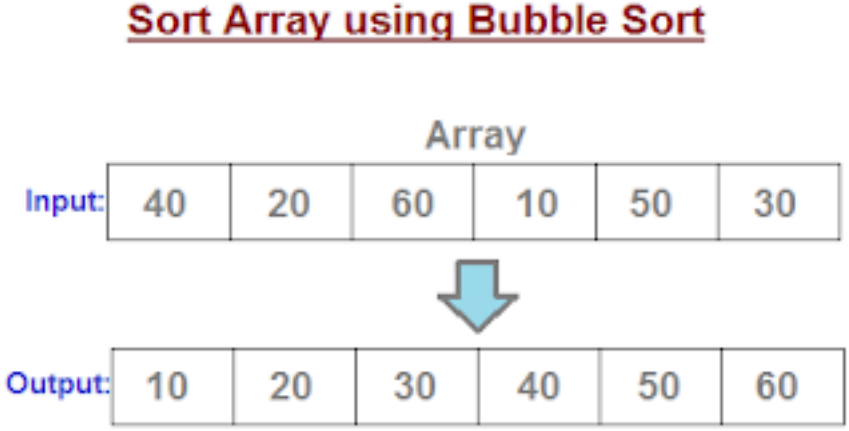
Lab 2 CSUN Comp 182

Seyed Alireza Alampour

May 7, 2018

1 Bubble Sort algorithm

In this lab we are looking at analysis of bubble, which iterates through an array, compares adjacent items and swap them if they are out of order. In each iteration, Largest item will be moved(bubble up) at its proper place.



The bubble sort algorithm has a worst case time complexity or BigOh 1, A best case time complexity or BigOmega 2, And an average time complexity of3.

$$O(n^2) \tag{1}$$

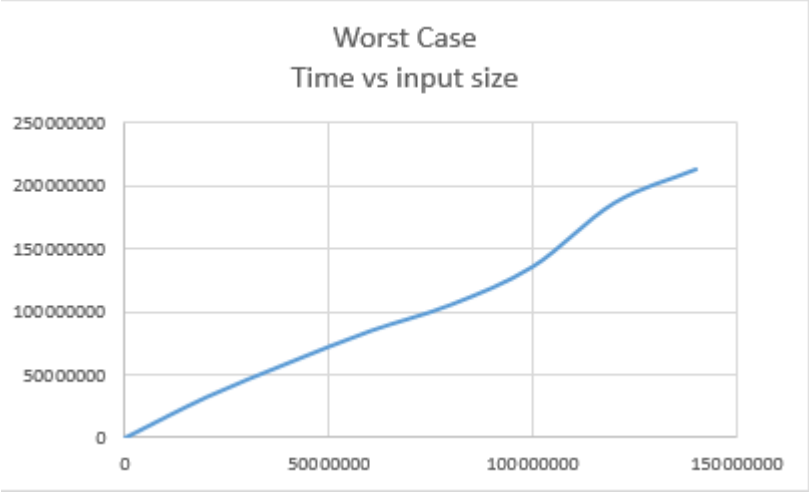
$$\Omega(n) \tag{2}$$

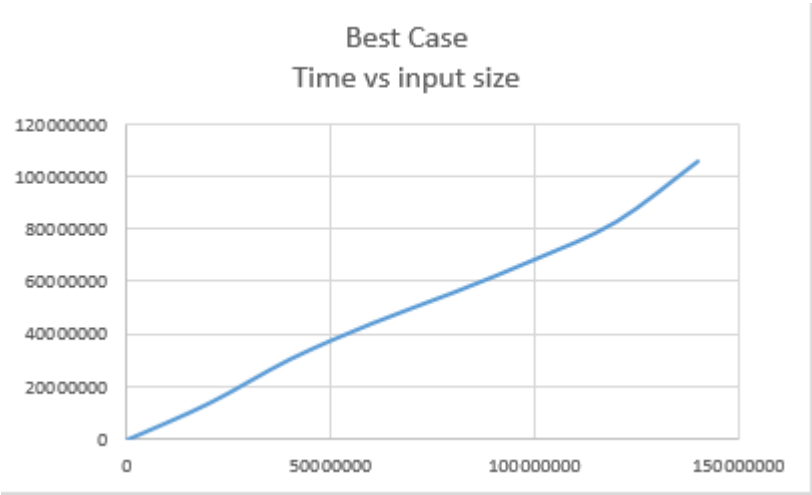
$$\theta(n^2) \tag{3}$$

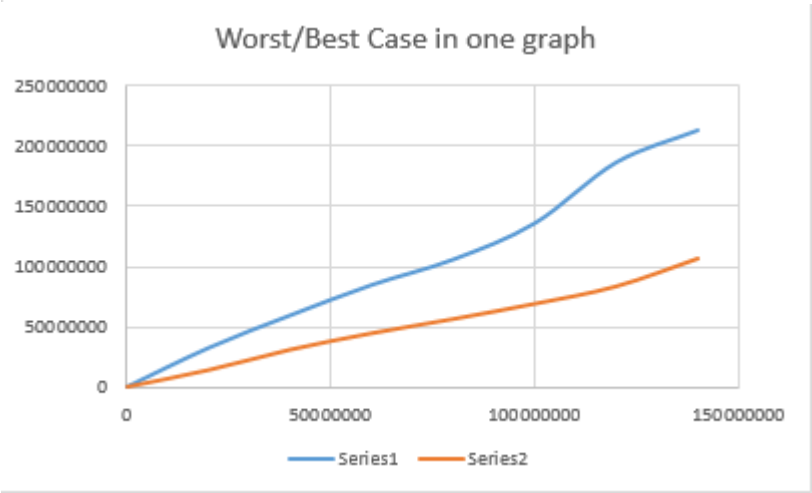
2 Source Code and Analysis

To Analyze The Bubble Sort, I have set up the bubbleSort method which takes in a comparable type and does the sorting. Because bubbleSort method takes in a comparable type, I have another method turning the int to Integer. After doing so I take the time for the bubbleSort method to finish, and by adjusting the inputs, I can get the best and the worst case, which are shown in the graphs below.

```
1         Integer[] A = convertInt(input);//Turn int into Integer
2         //bubbleSort(A);
3         long start = System.nanoTime();
4         bubbleSort(A); // the actual bubble sort
5         long finish = System.nanoTime();
6         // Solve for the actual execution time
7         long algTime =  finish - start;
8         System.out.println(algTime +"---"+A.length);
```







3 Conclusion

when the given data set is sorted in the opposite way(Worst Case) therefore $O(n^2)$, which makes it quite inefficient for sorting large data volumes. Also When the given data set is already sorted(Best Case), in that case bubble sort can identify it in one single iteration hence Ω(n).

4 Source Code

Here is the source code I used.

```
1       package lab2;
2 import java.util.Arrays;
3 import java.util.Random;
4 import java.util.stream.IntStream;
5
6
7 public class LAB2 {
8
9
10     public static void main(String[] args)
11     {
12         benchmark (8, 10);
13     }
14
15
16     public static void benchmark(int x, int y)
17     {
18         // Then you call  the method
19         int [][] myTests =  getArray(x,y);
20         for (int []vector : myTests )
21         {
22             Arrays.sort(vector);
23             myMethod(vector);
24         }
25
26     }
27
28     public static int[][] getArray(int x, int y)
29     {
30         return IntStream.range(0, x).parallel().mapToObj(i-> new Random().ints(1*i*(int)Math
31     }
32
33
34     public static void myMethod (int [] input)
35     {
36
37         // Then you time when the method finishes
38         // First you count the start time
39         Integer[] A = convertInt(input);
40         Arrays.sort(A);
```

```
41          long start = System.nanoTime();
42          bubbleSort(A);
43          long finish = System.nanoTime();
44          // Then you solve for the actual execution time
45          long algTime =  finish - start;
46          System.out.println(algTime);//input.length algTime
47      }
48          public static Integer[] convertInt(int[] input) {
49          Integer[] result = new Integer[input.length];
50          for(int i = 0; i < input.length; i+=1){
51              result[i] = input[i];
52
53          }
54          bubbleSort(result);
55          return result;
56      }
57          public static <E extends Comparable<? super E>> void bubbleSort(E[] comparable) {
58      boolean changed = false;
59      do {
60          changed = false;
61          for (int a = 0; a < comparable.length - 1; a++) {
62              if (comparable[a].compareTo(comparable[a + 1]) > 0) {
63                  E tmp = comparable[a];
64                  comparable[a] = comparable[a + 1];
65                  comparable[a + 1] = tmp;
66                  changed = true;
67              }
68          }
69      } while (changed);
70  }
71
72  }
```