CO 322

SORTING ALGORITHMS

NAME : WIMALASIRI KPGP

REG NO : E/14/403

SEMESTER : 5th

DATE : 20/04/2018

Introduction

In this report, a comparison of three algorithms which are used to sort is discussed. Those algorithms are,

1. Bubble Sort
2. Selection Sort
3. Insertion Sort

Time taken for each sorting method execution is considered first and then the big 'oh' of each method is considered.

Experiment

Each algorithm is implemented in java and the runtime is measured in order to have an idea about the execution time of each method.

Click here to review the code.

1. Time measurements

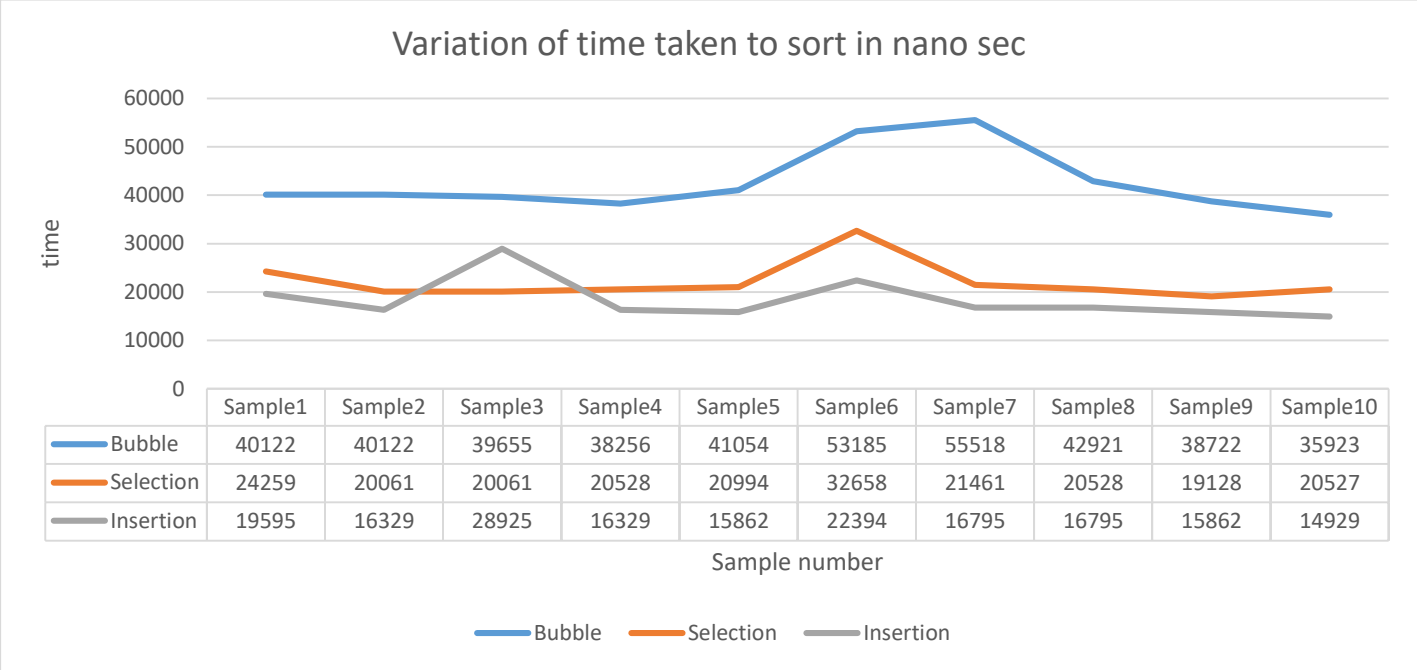
Sample 1	16	51	22	53	40	23	47	28	1	12	56	34	23	28	38	24	39	13	20	17	7	11	49	57	9	10	28	56	52	21
Sample 2	21	14	26	20	11	27	52	56	45	44	0	26	14	54	35	38	54	52	43	47	16	38	5	15	16	0	50	23	43	50
Sample 3	5	54	19	56	39	28	29	37	6	39	20	6	54	9	48	36	35	35	50	2	34	30	21	52	59	52	23	33	57	14
Sample 4	58	29	56	20	15	48	42	9	11	4	51	34	26	14	38	47	11	9	6	46	24	4	13	52	46	10	34	53	36	50
Sample 5	4	11	19	27	32	33	31	22	6	42	56	12	27	49	56	51	50	43	28	8	40	22	59	35	1	49	58	22	45	5
Sample 6	43	30	47	25	42	59	28	33	34	28	28	31	16	16	12	7	19	6	22	42	58	28	6	42	24	40	9	11	54	42
Sample 7	25	7	45	54	47	46	59	47	23	7	5	52	52	45	59	11	51	48	42	38	58	30	37	27	11	24	2	36	36	1
Sample 8	23	26	35	45	17	28	31	54	42	31	24	31	52	7	22	2	8	15	8	48	47	51	48	34	55	33	12	23	13	15
Sample 9	40	50	1	49	37	22	35	53	18	35	21	37	48	45	11	3	11	37	57	33	17	20	33	52	49	43	11	28	49	49
Sample 10	3	11	46	27	35	10	29	16	21	39	44	2	27	46	30	5	35	57	14	30	25	29	54	40	37	28	26	17	33	43

Table 1 Samples used in algorithms

	Sample1	Sample2	Sample3	Sample4	Sample5	Sample6	Sample7	Sample8	Sample9	Sample10
Bubble	40122	40122	39655	38256	41054	53185	55518	42921	38722	35923
Selection	24259	20061	20061	20528	20994	32658	21461	20528	19128	20527
Insertion	19595	16329	28925	16329	15862	22394	16795	16795	15862	14929

Table 2 Time taken to sort in nano seconds

Figure 1



Average time taken by each sorting methods is as follows.

Sorting method	Avg. time in nano sec.
Bubble	42547.8
Selection	22020.5
Insertion	18381.5

Table 3 Average time of performing

Therefore according to the time measurements best algorithm is Insertion sort. Then Selection sort algorithm performs well. Worst performance is by Bubble sort.

2. Big ‘O’ comparison

Bubble sort

```
static void bubble_sort(int [] data) {
    for(int j=0;j<data.length;j++){
        count=0;
        for(int i=1;i<data.length;i++){
          if(data[i]<data[i-1]){
          data = swap(i-1,i,data);
          count++;
            }
        }
        if(count==0)
            break;
    }
}
```

If the number of items is considered as N,

	Best Case	Worst Case
Comparisons	N(N-1)/2	N(N-1)/2
Swaps	0	N(N-1)/2
Total	N(N-1)/2	N(N-1)
Big ‘O’	N^2	N^2

Table 4 Big 'O' of Bubble sort

Selection sort

```
static void selection_sort(int [] data) {
    int maxIndex=0;
    int i;
    for(int j=0; j<data.length; j++){
        i=data.length-1-j;
        maxIndex = getMaxIndex(data,0,i);
        data = swap(data.length-1-j,maxIndex,data);
    }
}
```

```
  private static int getMaxIndex(int[] data, int from, int to) {
    int max=0;
    int maxIndex=0;
    for(int i=from;i<=to;i++){
        if(max<data[i]){
            max =data[i];
            maxIndex = i;
        }
    }
    return maxIndex;
}
```

If the number of items is considered as N,

	Best Case	Worst Case
Comparisons	N(N-1)/2	N(N-1)/2
Swaps	0	N-1
Total	N(N-1)/2	(N(N-1)/2)+(N-1)
Big 'O'	N^2	N^2

Table 5 Big 'O' of Selection sort

Insertion sort

```
static void insertion_sort(int [] data) {
    for(int i=0; i<data.length-1; i++){
        if(data[i]>data[i+1]){
            int tmp = data[i+1];
            for(int j=i;j>=0;j--){
                if(tmp<data[j]){
                    data[j+1] = data[j];
                    data[j] = tmp;
                }else{
                    break;
                }
            }
        }
    }}
```

If the number of items is considered as N,

	Best Case	Worst Case
Comparisons	N-1	N(N-1)/2
Swaps	0	N(N-1)/2
Total	N-1	N(N-1)
Big 'O'	N	N^2

Table 6 Big 'O' of Insertion sort

According to the Big 'O' calculation, Insertion shows the best, 'best performance'. However, worst performance is almost the same in every other algorithm for large N.

❖ Considering the both operations done on this paper, according to that insertion sorting algorithm shows better performance for large N than the others.

Appendix

```
/**************************************************
 * Simple sorting algorithms
 * Modified by E/14/403
 **************************************************/
import java.util.Random;
class CompareSorting {
   static int count = 0;         //count to check whether how many swaps done
   static void bubble_sort(int [] data) {
      for(int j=0;j<data.length;j++){
         count=0;                // for each start of round make count=0
         for(int i=1;i<data.length;i++){
            if(data[i]<data[i-1]){     // checking whether the two considered items are sorted
               data = swap(i-1,i,data);
               count++;             // count incremented as swap called
            }
         }
         if(count==0)       // no swaps done
            break;        // stop sorting
      }
   }
```

```
static void selection_sort(int [] data) {
   int maxIndex=0;
   int i;
   for(int j=0; j<data.length; j++){                // j is counting backwards as the sorted items are kept in the end of the array
      i=data.length-1-j;                        // i is the counter which iterates through unsorted items
      maxIndex = getMaxIndex(data,0,i);          // getting the index of maximum item in the unsorted list
      data = swap(data.length-1-j,maxIndex,data); // swapping the maximum item with next sorted item
   }
}
static void insertion_sort(int [] data) {
   for(int i=0; i<data.length-1; i++){        // iterating through the whole list
      if(data[i]>data[i+1]){             // check whether the next item is less than the current
         int tmp = data[i+1];           // copying next item to a tmp
         for(int j=i;j>=0;j--){         // iterating through the sorted items
            if(tmp<data[j]){
               data[j+1] = data[j];    // shifting items by one step
               data[j] = tmp;         // inserting the considered value
            }else{
               break;             // if the item is kept in the right place break the loop
            }
         }
      }
   }
}
// Helper functions
static int [] generate_data(int sizeOfData) {
          /* create an array of sizeOfData and
           * populate with random integers betweem 0-1000  */
          int [] tmp = new int[sizeOfData];
          Random rand = new Random();
          for(int i=0; i < sizeOfData; i++)
             tmp[i] = rand.nextInt(2*sizeOfData);
          return tmp;
}
static int [] duplicate_array(int [] data) {
          /* create a duplicate array of the given
           * useful when sending the same array to different
```

```
         * algorithms. */
        int [] tmp = new int[data.length];
        for(int i=0; i< data.length; i++)
           tmp[i] = data[i];
        return tmp;
}
static void show(int [] data) {
        System.out.printf("\n");
        for(int i=0; i < data.length; i++)
           System.out.printf("%d %c", data[i],
         i == (data.length - 1) ? ' ' : ',');
        System.out.printf("\n");
}
static void postCondition(int [] data) {
        /* if sorted, for any i data[i] > data[i-1]
         * Need to run this with java -ea CompareSorting*/
        int i;
        for(i=1; i < data.length; i++)
           if(data[i] > data[i-1]) break;
        assert i == data.length : "Sorting algorithm used is broken";
}
private static int [] swap(int index2, int index1, int[] data) {
   int tmp;
   tmp = data[index2];
   data[index2]=data[index1];
   data[index1]= tmp;
   return data;
}
private static int getMax(int[] data) {       // getting the maximum item from the given array
   int max=0;
   for(int i=0;i<data.length;i++){
      if(max<data[i]){
         max =data[i];
      }
   }
   return max;
}
```

```
private static int getMaxIndex(int[] data, int from, int to) {      // getting the maximum item's index from the given array
    int max=0;
    int maxIndex=0;
    for(int i=from;i<=to;i++){
        if(max<data[i]){
            max =data[i];
            maxIndex = i;
        }
    }
    return maxIndex;
}
public static void main(String [] ar) {
    int [] t = generate_data(30);
    int [] t1 = duplicate_array(t);
    int [] t2 = duplicate_array(t);
    show(t);
    System.out.println();
    System.out.print("Bubble Sort");
    long timeBubble1 = System.nanoTime();
    bubble_sort(t);
    long timeBubble2 = System.nanoTime();
    long timeBubble = timeBubble2 - timeBubble1;
            show(t);
    System.out.println();
            postCondition(t);
    System.out.println();
    System.out.print("Selection Sort");
    long timeSelection1 = System.nanoTime();
    selection_sort(t1);
    long timeSelection2 = System.nanoTime();
    long timeSelection = timeSelection2 - timeSelection1;
    show(t1);
    System.out.println();
    postCondition(t1);
    System.out.println();
    System.out.print("Insertion Sort");
    long timeInsertion1 = System.nanoTime();
```

```
        insertion_sort(t2);
        long timeInsertion2 = System.nanoTime();
        long timeInsertion = timeInsertion2 - timeInsertion1;
        show(t2);
        System.out.println();
        postCondition(t2);
        System.out.println("Bubble sort\t----> "+timeBubble+" ns");
        System.out.println("Selection sort\t----> "+timeSelection+" ns");
        System.out.println("Insertion sort\t----> "+timeInsertion+" ns");
    }

}
```