KINGDOM OF SAUDI ARABIA
Ministry of Higher Educatio
Al-Imam Mohammad University
College of Computer & Information Sciences

المملكة العربية السعودية السعودية
وزارة التعليم العالي
جامعة الإمام محمد بن سعود الإسلامية
كلية علوم الحاسب والمعلومات

<u>**Second term 1441/2020**</u>

Design and analysis algorithms (CS- 310)
Section: 171

Course Project:
Sorting Algorithms

Submitted By
Saad BinOnayq (439017145) – Coordinator
Fawzan alhantoshi (439014363)
Mohammed Alkhalifah (439011298)
Ibrahem altuwayjiri (439013074)

Supervisor
Ganesh Kumar Perumal

Date: 2020\4\4

Table of Content

1. Introduction

Initially, algorithms are one of the foundations of programming and development, and many distinguished programs are one of their strengths in choosing a fast and practical algorithm.

For this in this project, we will create an algorithm for the sortings (Heapsort - insertion Sort) and we will apply them to the Java language and also to the pseudocode, and we will test it with several variables and show the time complexity for each algorithm.

2. HeapSort

2.1 pseudocode

```
Heapsort(A as array)
   n = elements_in(A)
   for i = floor(n/2) to 1
      Heap(A,i,n)

   for i = n to 1
      swap(A[1], A[i])
      n = n - 1
      Heap(A, 1)

Heap(A as array, i as int, n as int)
   max=i
   left = 2i
   right = 2i+1

   if (left <= n) and (A[left] > A[i])
      max = left

   if (right<=n) and (A[right] > A[max])
      max = right

   if (max != i)
      swap(A[i], A[max])
      Heap(A, max)
```

2.2 Time complexity

The worst and best case time complexity of HeapSort is (N Log N)

2.3 Source code

```
public class Heapsort {
   public static void heapsort(int[] array) {

      // Turns the array to heap
      int arrSize = array.length;
      for (int i = arrSize / 2 - 1; i >= 0; i--) {
         heap(array, arrSize, i);
      }

      // Extract all elements from the heap.
      for (int i = arrSize - 1; i >= 0; i--) {
         int temp = array[0];
         array[0] = array[i];
         array[i] = temp;

         heap(array, i, 0);
      }
   }

   private static void heap(int[] array, int arrSize, int i) {
      int max = i;           // Set max as the root
      int left = 2 * i + 1;   // Set left child of i
      int right = 2 * i + 2;  // Set right child of i

      if (left < arrSize && array[left] > array[max])
         max = left;

      if (right < arrSize && array[right] > array[max])
         max = right;

      if (max != i) {
         int swap = array[i];
         array[i] = array[max];
         array[max] = swap;

         heap(array, arrSize, max);
      }
   }
}
```

2.4 Output screenshots

```
          ==Heapsort==
Input: [41, 18, 94, 53, 37, 72, 16, 10, 84, 7]
Output: [7, 10, 16, 18, 37, 41, 53, 72, 84, 94]
```

3. InsertionSort

3.1 pseudocode

```
for i from 1 to N
  key = a[i]
  j = i - 1
  while j >= 0 and a[j] > key
    a[j+1] = a[j]
    j = j - 1
  a[j+1] = key
next i
```

3.2 Time complexity

The worst case time complexity of InsertionSort is ($\mathbf{N^2}$)
The best case time complexity of InsertionSort is (**N**)

3.3 Source code

```
public class InsertionSort {
    public static int[] insertionSort(int[] arr) {
        for (int i = 1; i < arr.length; i++) {
            int key = arr[i];
            int j = i - 1;
            while (j >= 0 && arr[j] > key) {
                arr[j + 1] = arr[j];
                j--;
                arr[j + 1] = key;
            }
        }
        return arr;
    }
}
```

3.4 Output screenshots

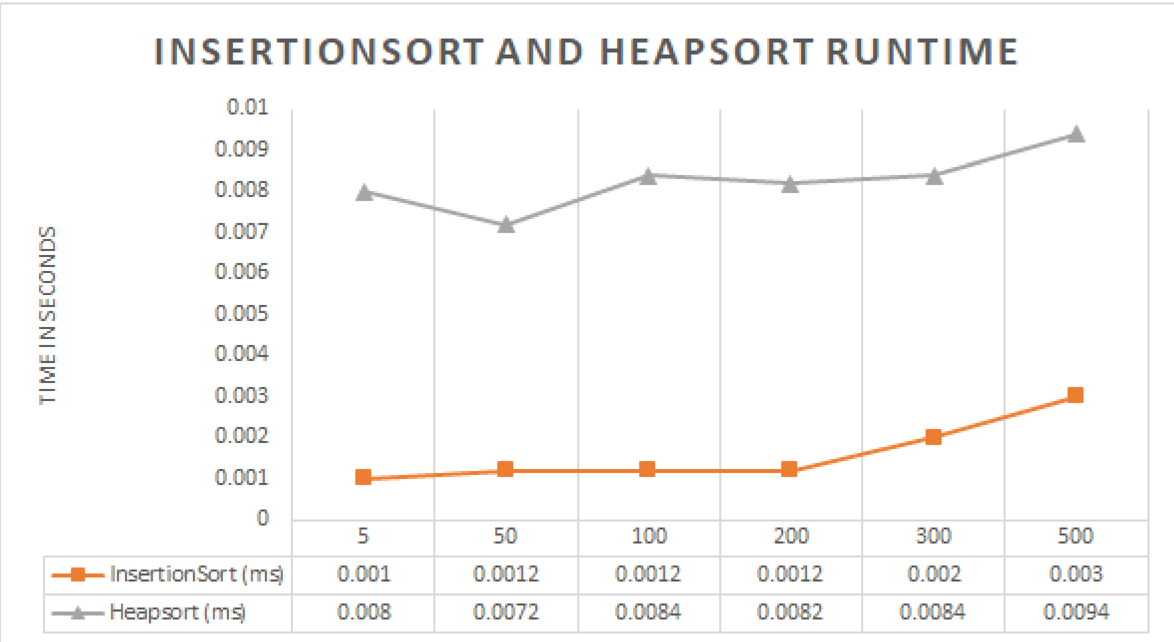
```
             ==InsertionSort==
Input: [36, 94, 70, 89, 55, 10, 66, 54, 97, 6]
Output: [6, 10, 36, 54, 55, 66, 70, 89, 94, 97]
```

4. Results of Heap and Insertion sorts

4.1 Result table

Input \ algorithm	Heap Sort(ms)	Insertion Sort(ms)
5	0.008	0.001
50	0.0072	0.0012
100	0.0084	0.0012
200	0.0082	0.0012
300	0.0084	0.002
500	0.0094	0.003

4.2 Result Chart



Based on the results and the time complexity, we see that the Insertion Sort in the small inputs is better than the Heap Sort , but with the large inputs the Heap Sort is better and faster.

5. Descending HeapSort

5.1 pseudocode

```
Heapsort(A as array)
   n = elements_in(A)
   for i = floor(n/2) to 1
      Heap(A,i,n)

   for i = n to 1
      swap(A[1], A[i])
      n = n - 1
      Heap(A, 1)

Heap(A as array, i as int, n as int)
   max=i
   left = 2i
   right = 2i+1

   if (left <= n) and (A[left] < A[i])
      max = left

   if (right<=n) and (A[right] < A[max])
      max = right

   if (max != i)
      swap(A[i], A[max])
      Heap(A, max)
```

5.2 Source code

```
public class HeapsortDecrement {
    public static void heapsort(int[] array) {

        // Turns the array to heap
        int arrSize = array.length;
        for (int i = arrSize / 2 - 1; i >= 0; i--) {
            heap(array, arrSize, i);
        }

        // Extract all elements from the heap.
        for (int i = arrSize - 1; i >= 0; i--) {
            int temp = array[0];
            array[0] = array[i];
            array[i] = temp;

            heap(array, i, 0);
        }
    }

    private static void heap(int[] array, int arrSize, int i) {
        int max = i;            // Set max as the root
        int left = 2 * i + 1;   // Set left child of i
        int right = 2 * i + 2;  // Set right child of i

        if (left < arrSize && array[left] < array[max])
            max = left;

        if (right < arrSize && array[right] < array[max])
            max = right;

        if (max != i) {
            int swap = array[i];
            array[i] = array[max];
            array[max] = swap;

            heap(array, arrSize, max);
        }
    }
}
```

5.3 Output screenshots

```
            ==Heapsort Decrement==
Input: [57, 18, 25, 36, 16, 39, 2, 59, 21, 9]
Output: [59, 57, 39, 36, 25, 21, 18, 16, 9, 2]
```

6. Conclusion

In conclusion, I hope that we have clarified the sorting algorithms and their codes, their time complexity, and the outputs for each algorithm and the preference between them.

7. References

1- Intellij de
2- Maven
3- Maven-compiler-plugin
4- Maven-surefire-plugin
5- Juint