|  |  |  |  |
| --- | --- | --- | --- |
| **المملكة العربية السعودية السعودية**  **وزارة التعليم العالي**  **جامعة الإمام محمد بن سعود الإسلامية**  **كلية علوم الحاسب والمعلومات** | | A description...  **Second term 1441/2020** | **KINGDOM OF SAUDI ARABIA**  **Ministry of Higher Education**  **Al-Imam Mohammad University**  **College of Computer & Information Sciences** |
|  | **Design and analysis algorithms (CS- 310)**  **Section: 171**  **Course Project:**  **Sorting Algorithms**  **Submitted By**  Saad BinOnayq (439017145) – Coordinator  Fawzan alhantoshi (439014363)  Mohammed Alkhalifah (439011298)  Ibrahim Altwiejery(439xxxxxx)  **Supervisor**  Ganesh Kumar Perumal  **Date:** 2020\4\4 | | | |

## **Table of Content**

[**1. Introduction**](#_gw1ywyh1oe3e) **4**

[**2. HeapSort**](#_vepjznkj8xy7) **4**

[***2.1 pseudocode***](#_vy5v38ayu5kl) ***4***

[***2.2 Time complexity***](#_vdplvvy43cey) ***4***

[***2.3 Source code***](#_ufswtzqv8m13) ***4***

[***2.4 Output screenshots***](#_nybdhmgrar8o)**4**

[**3. InsertionSort**](#_j7t0w9kvo5bi) **5**

[***3.1 pseudocode***](#_dum5ynjjw1r7) ***5***

[***3.2 Time complexity***](#_cdj53jmxnz87) ***5***

[***3.3 Source code***](#_z46tywwza7rm) ***5***

[***3.4 Output screenshots***](#_kn3r99ktet8o) ***5***

[**4. Results of Heap and Insertion sorts**](#_j1qsty4vn5er) **6**

[***4.1 Result table***](#_p4j95py1h23q) ***6***

[***4.2 Result Chart***](#_dgvisvc019rk) ***6***

[**5. Descending HeapSort**](#_628e4uadzbcr) **7**

[***5.1 pseudocode***](#_tl411t3myfk0) ***7***

[***5.2 Source code***](#_qw2qfz1ow5as) ***7***

[***5.3 Output screenshots***](#_6wax8mupi4c0) ***7***

[**6. Conclusion**](#_fqp3ig5vjv8b) **8**

[**7. References**](#_hfjr5e6we17w) **8**

## 

## 

## 

## 

## **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)

## 

## 

## 

## 

## 

## 

## 

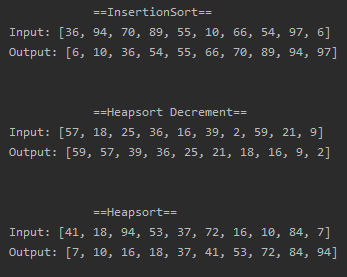
## 

## 

## **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**



## **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 (**N2**)

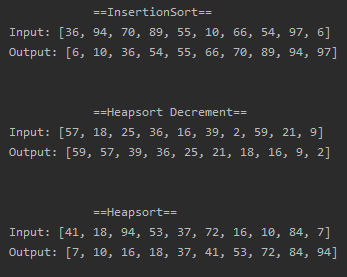
The best case time complexity of InsertionSort is (**N**)

## 

## **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**

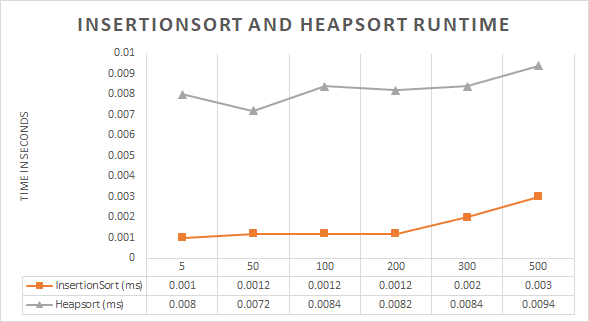


## **4. Results of Heap and Insertion sorts**

## **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**

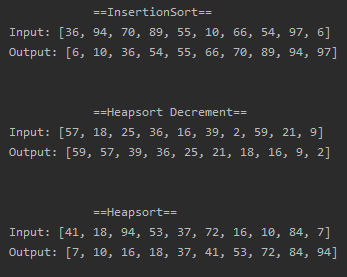
## **5.1 pseudocode**

|  |
| --- |
| **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**



## **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**