Experimento sobre ordenamiento de un arreglo.

1. **Planeación y realización.**

El objeto de estudio será sobre las diferentes maneras que se puede ordenar un arreglo, se definirán dos tipos de algoritmos con la misma complejidad y sobre estos se trabajarán los diferentes factores que pueden afectar su estado.

La variable a considerar será el tiempo que demore el correr el algoritmo este tiempo será tomado con la herramienta dada por el lenguaje C#.

Se experimentará con varios factores además de la diferencia de algoritmo y repetiremos el tratamiento n veces.

Los factores a estudiar serán:

* Algoritmo de ordenamiento.
* Tamaño del arreglo.
* Estado de los valores en el arreglo (en orden aleatorio, ordenado ascendente, ordenado descendente)
* \*\*\*\*\*\*\*

Para este estudio utilizaremos dos algoritmos; el heapsort y el quicksort ambos con una complejidad de 𝜣(n log n), se varía el tamaño del arreglo en 20, 450, y 1000, se harán en todos los órdenes(ascendente, ordenado y descendente),

|  |  |  |
| --- | --- | --- |
| Algoritmo de ordenamiento | Tamaño del arreglo | Estado de los valores del arreglo. |
| Heapsort | 20 | Ascendente |
| Heapsort | 450 | Ascendente |
| Heapsort | 1000 | Ascendente |
| Heapsort | 20 | Ordenado |
| Heapsort | 450 | Ordenado |
| Heapsort | 1000 | Ordenado |
| Heapsort | 20 | Descendente |
| Heapsort | 450 | Descendente |
| Heapsort | 1000 | Descendente |
| Quicksort | 20 | Ascendente |
| Quicksort | 450 | Ascendente |
| Quicksort | 1000 | Ascendente |
| Quicksort | 20 | Ordenado |
| Quicksort | 450 | Ordenado |
| Quicksort | 1000 | Ordenado |
| Quicksort | 20 | Descendente |
| Quicksort | 450 | Descendente |
| Quicksort | 1000 | Descendente |

Complejidad temporal HeapSort.

MAX-HEAPIFY(A, i)

1 l = LEFT.(i)

2 r = RIGHT.(i )

3 if l <= A:heap-size and A[l] > A[i]

4 largest = l

5 else largest = i

6 if r <= A:heap-size and A[r] > A[largest]

7 largest = r

8 if largest != i

9 exchange A[i] with A[largest]

10 MAX-HEAPIFY(A, largest)

La función de este algoritmo es T(n) ≤ T(2n/3) + Θ(1)1

Por teorema del maestro la complejidad es de O(lg n)

BUILD-MAX-HEAP(A)

1 A.heap-size = A.length

2 for i = [A.length/2] downto 1

3 MAX-HEAPIFY(A, i )

La complejidad del BUILD-MAX-HEAP es Θ(n)[[1]](#footnote-1)

HEAPSORT(A)

1 BUILD-MAX-HEAP(A)

2 for i = A:length downto 2

3 exchange A[1] with A[i]

4 A.heap-size = A.heap-size - 1

5 MAX-HEAPIFY(A 1)

El HeapSort hace uso del BUILD-MAX-HEAP y el MAX-HEAPIFY por tanto la complejidad final es O(n log n).

1. Cormen, T.,Leiserson C.,Rivest, R., & Stein, C.. (2009). Introduction to algorithms. Massachusetts: Mit Pr [↑](#footnote-ref-1)