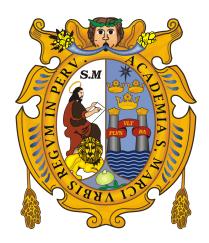
Universidad Nacional Mayor de San Marcos

FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

E.P. INGENIERÍA DE SOFTWARE



ASIGNATURA:

Análisis y Diseño de Algoritmos

DOCENTE:

Chavez Soto, Jose Luis

Tarea: Comparación entre HeapSort y BubbleSort Estudiante:

Gomez Olivas, Deyvi Pedro

HeapSort: O(nlogn)

```
public static void insertionHeap (int[] arr) {
      int index = 1;
      while (index < arr.length) {</pre>
        int auxIndex = index;
4
        while (auxIndex > 0) {
          int parentIndex = (auxIndex - 1) / 2;
         if (arr[auxIndex] > arr[parentIndex]) {
            int temp = arr[auxIndex];
            arr[auxIndex] = arr[parentIndex];
9
            arr[parentIndex] = temp;
10
            auxIndex = parentIndex;
11
          } else {
13
             break;
14
        }
15
        index++;
16
      }
18 }
public static void heapify (int arr[], int maxRange) {
      int index = 0;
      int maxIndex = 0;
      while (index <= maxRange) {</pre>
        int childLeft = index*2 + 1;
        int childRight = index*2 + 2;
6
       if (childLeft > maxRange && childRight > maxRange) {
          break;
10
       if (childLeft > maxRange && childRight <= maxRange) {</pre>
11
          maxIndex = childRight;
12
13
       if (childRight > maxRange && childLeft <= maxRange) {</pre>
14
15
         maxIndex = childLeft;
16
17
        if (childLeft <= maxRange && childRight <= maxRange ) {</pre>
          maxIndex = arr[childLeft] > arr[childRight] ? childLeft : childRight;
18
19
20
        //change
21
        if (arr[index] < arr[maxIndex]) {</pre>
22
         int temp = arr[index];
23
         arr[index] = arr[maxIndex];
24
          arr[maxIndex] = temp;
25
          index = maxIndex;
26
        } else {
27
          break;
28
29
      }
30
31 }
    public static void heapSort (int[] arr, int N) {
1
     for (int i = 1; i <= N - 1; i++) {
2
       int temp = arr[N - i];
3
        arr[N - i] = arr[0];
        arr[0] = temp;
5
        if (i < N - 1) {</pre>
6
         heapify(arr, N - i - 1);
9
      }
10 }
```

Para ejecutar el HeapSort; primero se debe construir la estructura de un Heap en el arrego. Para esto se usa la función *insertionHeap()*, luego se intercambia el último elemento por el primero del arreglo y se vuelve a ordenar el arreglo restante. Esto se hace mediante las funciones *heapSort()* y *heapify()*.

BubbleSort: $O(n^2)$

Comparativas

TD *	1	•	• •	1	1	1	. • .	1	ordenamiento
Tiomno	an	$\alpha_{1}\alpha_{C11}$	CION	α	Inc	2	monthmod	α	ordonomionto
TIGHIDO	ue	elecu		ue	105	aı	igorrinios.	ue	of defiatifiento

	Cantidad de elementos	Tiempo de ejecución	${\bf Intercambios}$
Bubble Sort	1000	$9326327~\mathrm{ns}$	248355
${\it HeapSort}$	1000	1411607 ns	7335
Bubble Sort	10000	126078414 ns	24994308
HeapSort	10000	2968907 ns	106773
Bubble Sort	90000	11319206055 ns	2028683366
HeapSort	90000	14850438 ns	1247077
Bubble Sort	100000	14806657855 ns	2487605285
HeapSort	100000	17320553 ns	1400797
Bubble Sort	200000	59007670042 ns	9992809524
HeapSort	200000	16532050 ns	3001937
Bubble Sort	500000	329669950060 ns	62514029981
HeapSort	500000	48574623 ns	8152661

Cuadro 1: Cuadro de Comparativas

Observaciones

Se puede observar en la tabla que con el tamaño de los arreglos el *HeapSort* logra ordenarlos en menos tiempo y con menos intercambios. Además, se puede observar también que el timepo de *BubbleSort* crece de manera exponencial según se aumenta el tamaño del arreglo.

No se agregó más elementos porque el algoritmo de BubbleSort no lograba ordenar el arreglo, causando que el editor de código se cierre.