

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

28 de mayo de 2024

HeapSort: $O(n \log n)$

```
1 public static void insertionHeap (int[] arr) {
2     int index = 1;
3     while (index < arr.length) {
4         int auxIndex = index;
5         while (auxIndex > 0) {
6             int parentIndex = (auxIndex - 1) / 2;
7             if (arr[auxIndex] > arr[parentIndex]) {
8                 int temp = arr[auxIndex];
9                 arr[auxIndex] = arr[parentIndex];
10                arr[parentIndex] = temp;
11                auxIndex = parentIndex;
12            } else {
13                break;
14            }
15        }
16        index++;
17    }
18 }
```

```
1 public static void heapify (int arr[], int maxRange) {
2     int index = 0;
3     int maxIndex = 0;
4     while (index <= maxRange) {
5         int childLeft = index*2 + 1;
6         int childRight = index*2 + 2;
7
8         if (childLeft > maxRange && childRight > maxRange) {
9             break;
10        }
11        if (childLeft > maxRange && childRight <= maxRange) {
12            maxIndex = childRight;
13        }
14        if (childRight > maxRange && childLeft <= maxRange) {
15            maxIndex = childLeft;
16        }
17        if (childLeft <= maxRange && childRight <= maxRange ) {
18            maxIndex = arr[childLeft] > arr[childRight] ? childLeft : childRight;
19        }
20
21        //change
22        if (arr[index] < arr[maxIndex]) {
23            int temp = arr[index];
24            arr[index] = arr[maxIndex];
25            arr[maxIndex] = temp;
26            index = maxIndex;
27        } else {
28            break;
29        }
30    }
31 }
```

```
1 public static void heapSort (int[] arr, int N) {
2     for (int i = 1; i <= N - 1; i++) {
3         int temp = arr[N - i];
4         arr[N - i] = arr[0];
5         arr[0] = temp;
6         if (i < N - 1) {
7             heapify(arr, N - i - 1);
8         }
9     }
10 }
```

Para ejecutar el HeapSort; primero se debe construir la estructura de un Heap en el arreglo. 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: $O(n^2)$

```

1 public static void bubbleSort (int[] arr, int n) {
2     for (int i = 0; i < n; i++) {
3         for (int j = 0; j < n - 1; j++) {
4             if (arr[j] > arr[j + 1]) {
5                 int temp = arr[j];
6                 arr[j] = arr[j + 1];
7                 arr[j + 1] = temp;
8             }
9         }
10    }
11 }

```

Comparativas

Tiempo de ejecución de los algoritmos de ordenamiento

	Cantidad de elementos	Tiempo de ejecución	Intercambios
Bubble Sort	1000	9326327 ns	248355
HeapSort		1411607 ns	7335
Bubble Sort	10000	126078414 ns	24994308
HeapSort		2968907 ns	106773
Bubble Sort	90000	11319206055 ns	2028683366
HeapSort		14850438 ns	1247077
Bubble Sort	100000	14806657855 ns	2487605285
HeapSort		17320553 ns	1400797
Bubble Sort	200000	59007670042 ns	9992809524
HeapSort		16532050 ns	3001937
Bubble Sort	500000	329669950060 ns	62514029981
HeapSort		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 tiempo 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.