Análisis del funcionamiento de HeapSort

Este documento analiza el funcionamiento de HeapSort según la ejecución mostrada en la salida del programa `programaHeap`. Explicaré el proceso paso a paso, desde la construcción del heap hasta la ordenación final mediante el algoritmo HeapSort.

1. Visión general de HeapSort

HeapSort es un algoritmo de ordenación basado en una estructura de datos llamada heap (montículo). Tiene dos fases principales:

- Construcción del heap máximo (donde el mayor elemento está en la raíz)
- Extracción ordenada de elementos

El algoritmo tiene una complejidad de tiempo de O(n log n), lo que lo hace eficiente para grandes conjuntos de datos.

2. Construcción del Heap

El programa comienza con un arreglo desordenado: `45 36 54 27 63 18 72 9 81 90`

Durante la fase de construcción, el algoritmo convierte este arreglo en un heap máximo (donde cada nodo padre es mayor que sus hijos).

Los intercambios realizados durante la construcción son:

```
Intercambio (construir): 63 <-> 90
Intercambio (construir): 27 <-> 81
Intercambio (construir): 54 <-> 72
Intercambio (construir): 36 <-> 90
Intercambio (construir): 36 <-> 63
Intercambio (construir): 45 <-> 90
Intercambio (construir): 45 <-> 81
```

Estos intercambios siguen el proceso de "heapify", donde cada nodo se compara con sus hijos y se intercambia cuando es necesario, comenzando desde los nodos más bajos y trabajando hacia arriba.

Resultado del heap construido: `90 81 72 45 63 18 54 9 27 36`

En este heap máximo:

- 90 es la raíz (mayor elemento)
- Cada padre es mayor que sus hijos

3. Proceso de HeapSort

Una vez construido el heap, HeapSort realiza las siguientes operaciones repetidamente:

1. Intercambia la raíz (elemento máximo) con el último elemento no ordenado
2. Reduce el tamaño del heap en uno
3. Reorganiza el heap para mantener la propiedad de heap máximo

Iteración 1:

- Intercambio: 90 <-> 36
- Reorganización para restaurar heap
- Estado actual: `81 63 72 45 36 18 54 9 27 90`
- Nota: 90 ya está en su posición final (última)

Iteración 2:

- Intercambio: 81 <-> 27
- Reorganización para restaurar heap
- Estado actual: `72 63 54 45 36 18 27 9 81 90`
- Nota: 81 y 90 ya están en sus posiciones finales

Iteración 3:

- Intercambio: 72 <-> 9
- Reorganización para restaurar heap
- Estado actual: `63 45 54 9 36 18 27 72 81 90`

Iteración 4:

- Intercambio: 63 <-> 27
- Reorganización para restaurar heap
- Estado actual: `54 45 27 9 36 18 63 72 81 90`

Iteración 5:

- Intercambio: 54 <-> 18
- Reorganización para restaurar heap
- Estado actual: `45 36 27 9 18 54 63 72 81 90`

Iteración 6:

- Intercambio: 45 <-> 18
- Reorganización para restaurar heap
- Estado actual: `36 18 27 9 45 54 63 72 81 90`

Iteración 7:

- Intercambio: 36 <-> 9
- Reorganización para restaurar heap
- Estado actual: `27 18 9 36 45 54 63 72 81 90`

Iteración 8:

- Intercambio: 27 <-> 9
- Reorganización para restaurar heap
- Estado actual: `18 9 27 36 45 54 63 72 81 90`

Iteración 9:

- Intercambio: 18 <-> 9
- Estado actual: `9 18 27 36 45 54 63 72 81 90`

4. Resultado final

Arreglo ordenado: `9 18 27 36 45 54 63 72 81 90`