Sorting Algorithm of Choice

Heap Sort Overview:

Heap Sort is a comparison sorting technique that uses the Binary Heap data structure [2]. It's a combination algorithm of insertion sort and merge sort [1]. Heap sort is considered an optimization over the selection sort algorithm as we get a time complexity of O (n Log n) over selection sort's O(n^2) in the worst case but comes with its own pros and cons

Pros [2]

Guarantees O(n Log n) in the worst case (Time Complexity)

Memory Usage minimal if heapify() iterative not recursive

Cons [2]

Not Stable – Might change relative ordering of keys (but can be made stable)

Still slower than merge sort.

Implementation

```
public static void heapSort(int[] arr) { int n = arr.length;
// Step A: Build a max heap
 for (int i = n / 2 - 1; i >= 0; i--) {

heapify(arr, n, i);
}
 // Step B: Extract the maximum element and place it at the end
 for (int i = n - 1; i > 0; i--) {
swap(arr, 0, i);
heapify(arr, i, 0);
 }
}
private static void heapify(int[] arr, int n, int i) {
 int largest = i;   // Assume current index i is the largest
int left = 2 * i + 1;  // Left child index
int right = 2 * i + 2;   // Right child index
```

```
 // If left child is larger than the current largest
 if (left < n && arr[left] > arr[largest]) {
 largest = left;
 }
// If right child is larger than the current largest
 if (right < n && arr[right] > arr[largest]) {
largest = right;
 }
// If the largest is not the original root (i), swap
      if (largest != i) {
       swap(arr, i, largest);
      // Recursively heapify the affected subtree
      heapify(arr, n, largest);
      }
 }
```

Complexity Breakdown

Step A:

Loops runs for approx. n/2 times non(-1 irrelevant)

Call heapify = worst case O(log n)

O(n Log n)

However:

While looking at sources this algorithm is actually O(n)

This is due to the bottom up approach of this algorithm. 2 thing happen in this approach.

1. Nodes near the bottom of the binary tree do very little.
2. Nodes near top have large trees but not many of them.

So the amortized solution is O(n)

Step B + heapify:

loops runs approx n times(-1 irrelevant)

Swap root with the last element = O(1)

Heapify on size *i* = worst case of O(log n) due to height of reduced heap still order of log n

(n) * O(log n) = O(n Log n)

So, for the total time complexity we have.

Step A: O(n)

Step B: O(n Log n)

Total: O(n + n Log n) = **O(n log n)**

References:

[1] Heap Sort Algorithm

[2] Heap Sort - Data Structures and Algorithms Tutorials - GeeksforGeeks

[3] Binary Heap - GeeksforGeeks

[4]Time & Space Complexity of Heap Sort