Heapsort Detailed Description

Heapsort is a comparison-based algorithm that uses binary heap data structure to sort elements in an array. It's an in-place algorithm which means it doesn't need any additional memory space for sorting, making it very efficient with a space complexity of O (1).

Algorithm Steps:

1. **Convert input array into a Max Heap using Heapify**. Max heap is a binary tree where the value of each parent node is greater than or equal to the children. This makes sure that the largest element is at the root (top) of the heap.
2. **Extract elements from the Heap.** Remove the largest element from the heap (should now be at the root) and place it at the end of the array. After this step repeat step 1 and heapify again to make sure the new root is the largest element
3. Repeat until array is fully sorted

Pseudo Code:

```
heapsort(array)                    //runs n-1 times
    n = length(array)              // 1 assignment operation
    // Build Max Heap
    for i = (n/2) - 1 downto 0:    //n/2 iterations
    heapify(array, i, n)

    //extract elements from the heap
    for i = n – 1 downto 0:        //n iterations
    swap array[0] and array[i]     //3 operations per swap. 1 for temp array, 1 to assign
                                   // array[i] to [0] and 1 more to assign temp to array[i]

    //heapify reduced heap
    heapify(array, 0, i)
```

```
heapify(array,  i , n)
    largest = i                    //1 assignment operation
    left = 2 * i +1                //3 operations. 1 addition, 1 multiplication, 1 assign
    right = 2*i+2                  //3 operations

    //compare left child with current largest
    If left < n                    //1 compare operation
        if array[left] > array[largest]   //1 compare operation
            largest = left                // 1 assignment operation

    //compare right child with current largest
    If right < n                   //1 compare operation
        if array[right] > array[largest]  //1 compare operation
            largest = right               // 1 assignment operation

    //if largest value is not at the root swap
    If largest !=i                 // 1 operation
    swap array[i] and array[largest]  //3 operations
    heapify(array, largest, n)
```

Counting Summary:

Building Max Heap:

- The loop runs for around n/2 iterations on average
- Each iteration calls heapify function (around n/2 calls)

- Each heapify call takes O(log n) worst case

Sorting elements:

- Swap the root (max) with last element. 3 operations **O(1)**
- Call heapify function
- Time complexity for this stage is **n x O(log n)** since its repeated for n elements giving it a time complexity of **O(n log n)**

Total Time Complexity

Step 1: building max heap: **O(n)**

Step 2: Extracting/Sorting: **O(n log n)**

Total Time Complexity: O(n log n)