Algorithms Required for Heapsort

1. **Heapify**: This function ensures that a subtree rooted at index **i** satisfies the heap property. It compares the node with its children and swaps it with the largest child if necessary, then recursively heapifies the affected subtree.
2. **Build Heap**: This is done by calling **heapify** on all non-leaf nodes, starting from the last non-leaf node down to the root. This rearranges the array into a max-heap.
3. **Sort**: After building the heap, the largest element (root of the heap) is swapped with the last element of the array. The size of the heap is reduced, and **heapify** is called on the root to maintain the heap property. This process is repeated until the heap is empty.

Analysis of the Heapsort Algorithm

1. **Time Complexity**:
    - **Building the Heap**: The **heapify** function is called **n/2** times, and each call takes O(log n) time in the worst case. Thus, building the heap takes O(n).
    - **Sorting**: The sorting process involves **n** calls to **heapify**, each taking O(log n) time. Therefore, the sorting step takes O(n log n).
    - Overall, the time complexity of Heapsort is O(n log n).