Heap-Sort algorithm

a)

Build a Max-Heap:

- **Start from the last non-leaf node in the array and apply the heapify process (also called sift-down or sift-heap) to ensure that the subtree rooted at that node satisfies the max-heap property.**
- **Repeat the heapify operation for all nodes up to the root node.**

Extract Maximum:

- **After building the max-heap, the root node (the first element) will contain the largest element.**
- **Swap the root with the last element of the heap (the current last element in the array).**
- **Decrease the heap size by 1.**
- **Reapply the heapify operation on the root node to restore the max-heap property.**

Repeat the Extraction:

- **Continue extracting the maximum element and rebuilding the heap until the size of the heap is reduced to 1. At this point, the array will be sorted.**

b)

Heapsort has two major phases: building the max-heap and extracting the maximum element.
Building the Max-Heap:

- **For each node in the heap, the heapify operation takes O(log n) time.**
- **In the worst case, the build process involves calling heapify on all nodes, so the overall time complexity for building the heap is O(n).**

Extracting Elements:

- **After building the max-heap, each extraction of the root (maximum) element involves swapping the root with the last element, followed by a heapify operation.**
- **Each heapify operation takes O(log n) time, and since we need to perform this extraction for each of the n elements, the time complexity for the extraction phase is O(n log n).**

Thus, the overall time complexity of the Heapsort algorithm is:

- **Time Complexity: O(n log n) for both average and worst-case scenarios.**
- **Space Complexity: O(1) because Heapsort is an in-place sorting algorithm (it doesn't require extra space aside from the input array).**

heap-sort .cs

c)