Part (a): Heapsort Algorithms

To perform sorting on a sequence of numbers using the Heapsort algorithm, we need to implement three core operations:

1. **Max Heap Construction: Transform the unsorted array into a max heap.**
2. **Heapify: Maintain the max heap property after an element is modified or removed.**
3. **Heapsort: Sort the array by repeatedly extracting the maximum element from the heap and placing it in its correct position.**

Let's discuss each of these operations:

1. Heapify Algorithm

Heapify ensures that a subtree rooted at a specific index follows the max heap property, meaning that each parent node's value must be greater than or equal to that of its children.

2. Max Heap Construction

The build_max_heap function ensures that all elements are arranged to satisfy the max heap property by applying heapify from the last non-leaf node to the root.

3. Heapsort Algorithm

Heapsort uses a max heap to sort the array. After constructing the max heap, it swaps the root (the maximum value) with the last element, shrinks the heap, and then calls heapify to restore the heap property.

Part (b): Detailed Analysis of the Algorithms

1. **Heapify Algorithm**:
    - **Time Complexity**: O(log n)
        - The time complexity of heapify comes from the fact that it may need to compare a node with its children and perform swaps down the tree. The number of comparisons and swaps needed is proportional to the height of the tree, which is log(n).
    - **Space Complexity**: O(1)
        - Heapify is an in-place operation, requiring only a constant amount of space for variables like largest, left, and right.
2. **Max Heap Construction**:
    - **Time Complexity**: O(n)
        - Despite invoking heapify on each non-leaf node, the total time complexity for building the heap is linear. This is because the majority of nodes are leaves

(which don't require heapification), and the rest require progressively fewer operations as we move up the tree.

- **Space Complexity**: O(1)
  - The construction of the max heap is done in-place, so only a constant amount of extra space is required.

3. **Heapsort Algorithm**:
   - **Time Complexity**: O(n log n)
     - The build_max_heap function runs in O(n) time, and each iteration in the sorting loop performs a swap followed by a call to heapify, which takes O(log n) time. Hence, the total time complexity of Heapsort is O(n log n).
   - **Space Complexity**: O(1)
     - Heapsort operates in-place, meaning it only requires a constant amount of extra space, aside from the input array.