Part (a): All Required Algorithms

Heapsort involves the following steps:

- **Building a Max-Heap**: Convert the array into a max-heap where the largest value is at the root. This is done by heapifying all non-leaf nodes.

  Algorithm:

  def build_max_heap(arr):
- n = len(arr)
- for i in range(n // 2 - 1, -1, -1):
- max_heapify(arr, n, i)

1.
- **Heapify Process**: Ensures the heap property is maintained for the subtree rooted at a given node.

  Algorithm:

  def max_heapify(arr, n, i):
- largest = i
- left = 2 * i + 1
- right = 2 * i + 2
-
- if left < n and arr[left] > arr[largest]:
- largest = left
- if right < n and arr[right] > arr[largest]:
- largest = right
-
- if largest != i:
- arr[i], arr[largest] = arr[largest], arr[i]
- max_heapify(arr, n, largest)

2.
- **Sorting**: Repeatedly extracts the maximum element and rebuilds the heap.

  Algorithm:

  def heap_sort(arr):
- n = len(arr)
- build_max_heap(arr)
- for i in range(n - 1, 0, -1):
- arr[0], arr[i] = arr[i], arr[0]
- max_heapify(arr, i, 0)

3.

Part (b): Algorithm Analysis

Time Complexity

1. **Max-Heapify**:
    - For a single node: O(logn).
    - As it is called for each extraction and during heap construction: O(nlogn).
2. **Build-Max-Heap**:
    - Iterates over all non-leaf nodes from bottom to top.
    - Total time complexity: O(n).
3. **Heapsort**:
    - Extraction phase requires O(nlogn) as it processes nn elements.

Overall Time Complexity: O(nlogn).

Space Complexity

- Heapsort is an in-place algorithm.
- **Space Complexity**: O(1)