Heap-Sort Algorithm

(a) Algorithms:

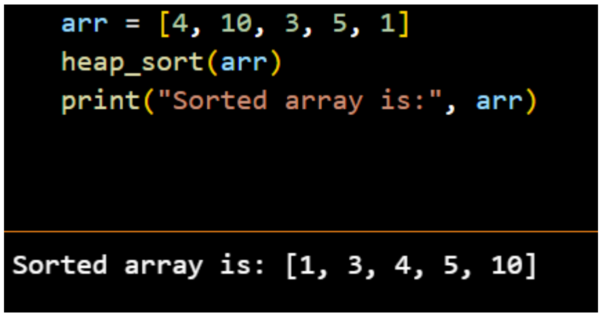
The Heap-Sort algorithm involves two main steps:

1. **Building the Max-Heap**:
   - Rearrange the elements of the array to satisfy the Max-Heap property.
   - Use the **`heapify`** function to ensure each **subtree** is a valid Max-Heap.
2. **Sorting**:
   - Repeatedly extract the largest element (root of the heap) by swapping it with the last element in the heap.
   - Reduce the size of the heap and reapply **`heapify`** to maintain the Max-Heap property.

(b) Analysis of Heap-Sort:

1. **Time Complexity**:
   - **Building the Heap**: O(n)
   - **Heapify during sorting**: For n elements, each heapify operation takes O(log n). Thus, sorting O(n log n).
   - Total Time Complexity: **O(n log n)**
2. **Space Complexity**:
   - Heap-Sort is an in-place algorithm, so it requires only O(1) extra space.
3. **Stability**:
   - Heap-Sort is not stable since element positions are swapped in a Max-Heap structure.

(c) Implementation Example:



```
arr = [4, 10, 3, 5, 1]
heap_sort(arr)
print("Sorted array is:", arr)
```

```
Sorted array is: [1, 3, 4, 5, 10]
```