Complexity Analysis of algorithms (Big O space and time complexity)

**HeapSort algorithm:**

**Time Complexity:**

1. **Build Heap (Max-Heapify):** O(n)
   * We first create a max-heap from the unsorted array (in our case, from the HashTable). This operation takes O(n) time because we start from the middle of the array and work our way up (sorting it following the form of a binary tree, in which each step of the heapify function takes linear time to be done), ensuring that the largest (for max-heap) elements "bubble up" to the top of the heap.
2. **Heapify (per element in the array):** O(n log n)
   * After the initial heap is built, we need to maintain the heap property for each element. This is done by "switching down" elements, which takes O(log n) time because we’re following the structure of a binary tree. Since we have to do this for all n elements, the total time complexity for this step is O(n log n).
3. **Heap Sort (per element in the array):** O(log n)
   * For each element in the array, we extract the root element from the heap (which is the maximum for max-heap or minimum for min-heap) and place it at the end of the sorted portion of the array. This operation takes O(log n) time because we need to restore the heap property. We repeat this process for all n elements, resulting in a total time complexity of O(n log n).

Overall, the time complexity of Heap Sort is **O(n log n)** in the worst, average, and best cases, because in all of these cases, we need to create and sort the heap for every operation we do).

**Space Complexity:** Heap Sort is an in-place sorting algorithm, which means it doesn't require extra memory proportional to the input size. The space complexity is constant:

1. **In-Place:** O(1)
   * Heap Sort operates using only a constant amount of additional memory to store variables and indices. It rearranges the elements in the input array without needing more memory as the input size grows.

**HashTable algorithm:**

**Time Complexity:**

In a Hash Table with chaining collision resolution, the time complexity of basic operations (insertion, retrieval, and deletion) depends on the number of elements in the Hash Table (n) and the number of collisions (k).

1. **Insertion (O(1) average, O(n) worst case):**
   * In the average case, when the hash function is well-distributed and collisions are minimal, inserting a key-value pair into the Hash Table is still considered an O(1) operation. However, in the worst case scenario, when many keys hash to the same location and we need to traverse the full linked list to insert, the time complexity becomes O(n). This worst-case scenario occurs when there are many collisions.
2. **GetElement (O(1) average, O(k) worst case):**
   * In the average case, getting a value by its key from the Hash Table with chaining is typically an O(1) operation. This is because we can directly access the linked list where the value is stored. However, in the worst case, when many keys hash to the same location and we need to search through a linked list, the time complexity becomes O(k), where k is the number of elements in the linked list at the hash location.
3. **Deletion (O(1) average, O(k) worst case):**
   * Deletion of a key-value pair in the Hash Table with chaining is usually an O(1) operation on average. However, in the worst case, when there are many collisions and we need to traverse a linked list, the time complexity becomes O(k), where k is the number of elements in the linked list at the hash location.

Here we can see the importance of an appropriate hash function, since it will lead us to less collisions, hence reducing the time complexity of our algorithm significantly.

**Space Complexity:**

The space complexity of a Hash Table using chaining for collision resolution depends on the number of key-value pairs stored in the table (n).

1. **Size of Stored Data (O(n)):**
   * The space complexity for a Hash Table with chaining grows with the number of key-value pairs stored in the table. If we have n key-value pairs, the space complexity is O(n) because we need to allocate memory for each pair and potentially for the linked lists used for collision resolution.