**Algorithm Analysis**

**First Algorithm Analysis (Cloning of elements):**

public TaskManagementController clone() {  
 TaskManagementController clon = new TaskManagementController(10);  
 clon.hashTableChaining = this.hashTableChaining.clone();  
 clon.keys = new ArrayList<>(this.keys);  
  
 HashNode<String, Activity>[] array = clon.hashTableChaining.getArray();  
 for (int i = 0; i < array.length; i++) {  
 if (array[i] != null) {  
 HashNode<String, Activity> current = array[i];  
 while (current!=null) {  
 current.setValue(current.getValue().clone());  
 Activity act = current.getValue();  
 if(act instanceof Task){  
 if(((Task) act).getPriorityLevel()==PriorityLevel.*NON\_PRIORITY*){  
 clon.taskQueue.offer(act);  
 } else if(((Task) act).getPriorityLevel()==PriorityLevel.*LOW*){  
 clon.priorityQueueLow.enqueue(act, calculatePriority(act.getDate()));  
 } else if(((Task) act).getPriorityLevel()==PriorityLevel.*MEDIUM*){  
 clon.priorityQueueMedium.enqueue(act, calculatePriority(act.getDate()));  
 } else {  
 clon.priorityQueueHigh.enqueue(act, calculatePriority(act.getDate()));  
 }  
 } else {  
 clon.reminderQueue.offer(act);  
 }  
 current = current.getNext();  
 }  
 }  
 }  
 return clon;  
}

**Temporal analysis:**

Let's analyze the code from the beginning, the for loop iterates through the cloned hash table structure, which results in a time complexity of O(n), where "n" is the size of the hash table.

Inside this loop, the while loop iterates through the linked lists at each position of the hash table. In the worst case, this loop has a time complexity of O(m), where "m" is the length of the longest linked list.

The cloning and copying operations performed within the while loop generally have a constant time complexity O(1).

So, the overall time complexity of the algorithm depends on the size of the hash table (n) and the length of the linked lists (m). In the worst case, the complexity is O(n \* m).

**Spatial analysis**

Cloning the task controller requires constant additional space O(1) for its core components. Cloning the hash table may require additional space proportional to the original size (O(n)), the same goes for the keys. Copying the activities consumes constant space for each activity O(1). Then, the overall space complexity is O(n) in the worst case, where "n" is the size of the original hash table.

**Second Algorithm Analysis (Remove an element from the hash table):**

public void remove(K key) {  
 int index = hash(key);  
 HashNode<K, V> currentNode = array[index];  
 HashNode<K, V> previousNode = null;  
  
 while (currentNode != null) {  
 if (currentNode.getKey().equals(key)) {  
 if (previousNode == null) {  
 array[index] = currentNode.getNext();  
 } else {  
 previousNode.setNext(currentNode.getNext());  
 }  
 return;  
 }  
 previousNode = currentNode;  
 currentNode = currentNode.getNext();  
 }  
}

**Temporal analysis:**

To fully analyze this algorithm, we first need to establish some initial context. Initially, we need to determine the hash index we are searching for, which is a variable assignment, so for now, we have a time complexity of O(1). Similarly, currentNode and previousNode are also variable assignments, so the complexity remains O(1) at this stage. Then, we encounter a while loop that iterates 'n' times until a node with the same key as the one being searched for is found. If the key is not found, the search continues through the entire collision chain. We also have variable assignments and if evaluations within the loop, each with a time complexity of O(1).

Therefore, the time complexity of this algorithm is O(n), as the worst-case scenario is when the key is in the last position of the collision chain within a specific index.

**Spatial analysis**

The space used by this algorithm is primarily for local variables such as index, currentNode, and previousNode, which occupy a constant space of O(1). The additional space used by these variables does not depend on the size of the hash table or the collision list, as they are temporary variables. No new data structures are created, and no additional data copies are stored in memory, resulting in a space complexity of O(1). So, the space complexity is O(1), meaning that the additional space used is independent of the hash table's size or the number of elements in the collision list.