Temporal and Spatial Complexity Analysis

Temporal Complexity Analysis:

Add method in a Hash Table

Statement	***Effort***
int index=hashFunction(key);	1
HashEntry<K,V> newEntry=new HashEntry< >(key,value);	1
HashEntry< K,V > current = table[index];	1
if (current==null){	1
table[index]=newEntry;	1

}else{
while(current.getNext!=null){
	n+1

current=current.getNext();
}
	n

current.setNext(newEntry);	1
newEntry.setPrev(current);	1

newEntry.setNext(null);
}
	1

this.existingNodes++;	1

$$T(A) = 1 + 1 + 1 + 1 + 1 + (n + 1) + n + 1 + 1 + 1 + 1$$

$$T(A) = 2n + 10$$

With this we can say that the time complexity of this algorithm in big O notation would be : $O(n)$

Insert element in a MinHeap

Statement	***Effort***
heap.add(element);	1
int index = heap.size()-1;	1
while(index > 0){	n+1
int parentIndex=(index-1)/2;	n
if(heap.get(index).compareTo(heap.get(parentIndex))<0){	n
T temp=heap.get(index);	n
Heap.set(index, heap.get(parentIndex));	n
Heap.set(parentIndex, temp);	n
Index=parentIndex;	n

}else{
break;
}
}
	n

$$T(A) = 1 + 1 + (n + 1) + n + n + n + n + n + n + n$$

$$T(A) = 8n + 3$$

With this we can say that the time complexity of this algorithm in big O notation would be : $O(n)$

Spatial Complexity Analysis:

Statement

```
public void addActivity(Integer id, String title, String description, LocalDate dueDate, String location,
boolean priority){

    Activity newActivity=new Activity(id, title, description, dueDate, location, priority);
    actionsStack.push(new Action(newActivity,1));
    activities.add(id, newActivity);

    if (priority)
        priorityActivities.insert(newActivity);
    else
        activitiesQueue.add(newActivity);

}
```

Type	***Variable***	***Length***	***Amount Values***
Input	*id*	-	0
	title	-	0
	description	-	0
	duedate	32	1
	location	-	0
	priority	16	1
Aux	*newactivity*		0
Output	*none*	-	

$$input + aux + output = 2 = O(1)$$

With this we can say that the spatial complexity of this algorithm in big O notation would be :

$$O(1)$$

Statement

```
public boolean ableToModify(){

    Activity modified=activities.findValue(id);
    if(modified!=null){
        boolean priority=modified.getPriority();
        if(priority && !priorityActivities.isEmpty()){
            if(priorityActivities.peekMax.getId.equals(id)){
                return true;
            }else{
                return false;
            }
    }else if(!priority && !activitiesQueue.isEmpty()){
        if(activitiesQueue.peek().getId().equals(id)){
            return true;
        }else{
            return false;
        }
    }

    }else{
        return false;
    }

    return false;

}
```

Type	***Variable***	***Length***	***Amount Values***
Input	*id*	-	0
	title	-	0
	description	-	0
	duedate	32	1
	location	-	0
	priority	16	1
Aux	*newactivity*		0
Output	*none*	-	

$$input + aux + output = 2 = O(1)$$

With this we can say that the spatial complexity of this algorithm in big O notation would be :

$$O(1)$$