**Complexity analiysis**

**Algorithm 1**

public String showTable(){

String print=""; O(1)

for (int i = 0; i < hashTable.length; i++) { O(n) = n

print+="Subdivision "+(i+1)+"\n"; O(n-1) = n-1

print+="\n"+hashTable[i].print()+"\n\n"; O(n-1) = n-1

}

return print; O(1)

}

The algorithm has a time complexity of O(n) since the first line is executed once. After that, we have a 'for' loop which is executed 'n' times. The next two lines are executed 'n-1' times because they are inside the 'for' loop. This is because before it enters the 'for' loop, it verifies the continue condition, and if it is false, the 'for' loop does not start again, and it goes to the next line. The last line is executed only once. Therefore, we have 1 + n + 2(n-1) + 1 = n + 2(n-1) + 2, and this complexity depends on 'n,' so it's O(n).

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Variable** | **Size of 1 atomic passengerdata** | **Quantity of atomic passengerdatas** |
| **Input** | **hashTable** | **64 bits** | **n** |
| **Auxiliary** | **i** | **32 bits** | **1** |
| **Output** | **Print** | **16 bits** | **n** |

**Total spatial complexity = Input + Auxiliary + Output = n+1+n = θ(1)**

**Auxiliary spatial complexity = 1 = θ(1)**

**Auxiliary + Output spatial complexity = 1 + n = θ(n)**

**Algorithm 2**

**public Boolean searchItemToItem(int hashPointer, int linekdListPointer){**

**Boolean out = true; O(1)**

**if(undoStack.isEmpty()){ O(1)**

**itemHashTable.search(hashPointer,linekdListPointer); o(1)**

**}else{**

**out = false; o(1)**

**}**

**return out; o(1)**

**}**

The algorithm has a time complexity of O(1) as we can observe that all the lines are executed only once, and there are no loops in the code. The sum is 1+1+1+1+1 = 5, which means the complexity is O(1).

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Variable** | **Size of 1 atomic passengerdata** | **Quantity of atomic passengerdatas** |
| **Input** | **hashPointer** | **32 bits** | **1** |
| **Input** | **linkedList** | **32 bits** | **1** |
| **Output** | **Out** | **8 bits** | **1** |

**Total spatial complexity = Input + Auxiliary + Output = 1+1+1 = θ(1)**

**Auxiliary spatial complexity = 0 = θ(0)**

**Auxiliary + Output spatial complexity = 1 + 1+1 = θ(1)**