**complexity analysis**

1. **the hash function hash division has a complexity of O(1) for the following reason:**

**function hashDivision(key) {**

hashCode = key.hashCode() // O(1) = 1

index = hashCode % 10 // O(1) = 1

if (index < 0) { // O(1) = 1

index = index + 10 // O(1) = 1

}

return index //O(1) = 1

**}**

**This is because each of the lines takes 1 time to execute, all the 5 lines in the code execute 1 time and its summatory is 1+1+1+1+1 = 5. And that means it takes constant time to execute the algorithm, and that means the time complexity is O(1)**

2)**the hash function get has a complexity of O(n) for the following reason:**

**function get(key) {**

index = hashDivision(key) // O(1) = 1

if (table[index] != null) { // O(1) = 1

for each passenger in table[index] { // O(n) = n

if (passenger.key.equals(key)) { // O(n-1) = n-1

return passenger.passengerdata // O(1) = 1

}

}

}

return null // O(1)

}

**}**

**This is because each of the 2 first lines take each one 1 time to execute, then for the next line we got a for each statement, a for statement executes n times, in this case the number of elements in that position with the same key. Then, the next line executes n-1 times, this because the lines inside the for executes 1 time less than the for statement, this because before it enters the for it verifies the continue condition and if it is false the for do not start again and it goes for the next line. At last we have the both return statement, each of them takes 1 time to execute. Finally we add every line, like this, n + n - 1 + 1 + 1 + 1 + 1 = 2n+3, and that complexity is dependent of n so the complexity is O(n)**

**Spatial complexity analysis**

**function hashDivision(long key) {**

hashCode = key.hashCode() // O(1)

index = hashCode % 10 // O(1)

if (index < 0) { // O(1)

index = index + 10 // O(1)

}

return index //O(1)

**}**

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Variable** | **Size of 1 atomic passengerdata** | **Quantity of atomic passengerdatas** |
| **Input** | **Key** | **64 bits** | **1** |
| **Auxiliary** | **hashCode** | **32 bits** | **1** |
| **Output** | **index** | **32 bits** | **1** |

**Total spatial complexity = Input + Auxiliary + Output = 3 = θ(1)**

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

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

2)

**function get(key) {**

index = hashDivision(key) // O(1)

if (table[index] != null) { // O(1)

for each passenger in table[index] { // O(n)

if (passenger.key.equals(key)) { // O(n)

return passenger.passengerdata // O(1)

}

}

}

return null // O(1)

}

**}**

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Variable** | **Size of 1 atomic passengerdata** | **Quantity of atomic passengerdatas** |
| **Input** | **Key** | **64 bits** | **1** |
| **Auxiliary** | **table** | **32 bits** | **n** |
| **Output** | **passenger** | **32 bits** | **1** |

**Total spatial complexity = Input + Auxiliary + Output = n + 2 = θ(n)**

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

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