**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)

index = hashCode % 10 // O(1)

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

index = index + 10 // O(1)

}

return index //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)

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

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

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

return entry.value // O(1)

}

}

}

return null // O(1)

}

**}**

**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 value** | **Quantity of atomic values** |
| --- | --- | --- | --- |
| **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 entry in table[index] { // O(n)

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

return entry.value // O(1)

}

}

}

return null // O(1)

}

**}**

| **Type** | **Variable** | **Size of 1 atomic value** | **Quantity of atomic values** |
| --- | --- | --- | --- |
| **Input** | **Key** | **64 bits** | **1** |
| **Auxiliary** | **table** | **32 bits** | **n** |
| **Output** | **entry** | **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)**