**Time Complexity Analysis**

**hashSearch(N) Temporal Analysis**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Algorithm hashSearch(N) | |  | # times each statement is executed |
| 1 | if (key == null) { | | C1 | 1 |
| 2 | throw new IllegalArgumentException("Null key");} | | C2 | 1 |
| 3 | int tried = 0; | | C3 | 1 |
| 4 | int j = 0; | | C4 | 1 |
| 5 | Node<V, K> temporal = null; | | C5 | 1 |
| 6 | while (tried != capacity){ | | C6 | n+1 |
| 7 | j = hash(key, tried); | | C7 | 1 |
| 8 | temporary = table.get(j); | | C8 | 1 |
| 9 | if (temporal != null && temporal.getKey().equals(key)){ | | C9 | n |
| 10 | return temporal.getValue();} | | C10 | n |
| 11 | tried += 1; } | | C11 | 1 |
| 12 | return null; | | C12 | 1 |
| HashSearch(n) | | 3n + 10 | | |

**buildMaxHeap(N) Temporal Analysis**

|  |  |  |  |
| --- | --- | --- | --- |
| # | Algorithm hashSearch(N) |  | # times each statement is executed |
| 1 | this.heapSize = array.size()-1; | C1 | 1 |
| 2 | for (int i = (int)Math.floor(array.size()/2); i>=1; i--){ | C2 | n |
| 3 | maxHeapify(i-1); } // Recursive Algorithm | C3 |  |
| buildMaxHeap(n) | | n log n | |

**heapSort(N) Temporal Analysis**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Algorithm heapSort(N) | |  | # times each statement is executed |
| 1 | buildMaxHeap() | | C1 | n |
| 2 | for (int i = array.size(); i>=1; i--) { | | C2 | n |
| 3 | Collections.swap(array, 0, i - 1); | | C3 | 1 |
| 4 | reduceHeapSize(); | | C4 | 1 |
| 5 | maxHeapify(0); } | | C5 | Log n |
| HashSearch(n) | | n log n | | |

**hashSearch(N) Spatial Analysis**

|  |  |  |  |
| --- | --- | --- | --- |
| Guy | Variable | Size of 1 atomic value | Number of atomic values |
| Entrance | Key | 32-bit (Si key == int) | n |
| Auxiliary | tried | 32-bit | 1 |
| J | 32-bit | 1 |
| temporary | 32-bit (Si value == int) | 1 |
| Exit | Null | Null / 32 bits (Si value == int) | 0/1 |

Total Spatial Complexity = Input + Auxiliary + Output = n + 4 = θ(n)

Auxiliary Spatial Complexity = 1 + 1 + 1 + 1 = θ(1)

Auxiliary + Output Spatial Complexity = 1 + 1 + 1 + 1 + 1 = θ(1)

**buildMaxHeap(N) Spatial Analysis**

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Variable | Size of 1 atomic value | Number of atomic values |
| Auxiliary | heapSize | 32-bit | 1 |
| Array | 32-bit | n |
| i | 32-bit | 1 |
| Exit | - | - | - |

Total Spatial Complexity = Input + Auxiliary + Output = n + 2 = θ(n)

Auxiliary Spatial Complexity = n + 1 + 1 = θ(n)

Auxiliary Spatial Complexity + Output = 0 = θ(1)

Auxiliary Spatial Complexity + Output = 0 = θ(1)