

```
1  import java.util.HashMap;
2  import java.util.Random;
3
4  class Node {
5      int value;
6      Node next;
7
8      Node(int value) {
9          this.value = value;
10         this.next = null;
11     }
12 }
13
14 class Hashing {
15
16     static int getRandomStudentId() {
17         Random rand = new Random();
18         return rand.nextInt(1000, 9999);
19     }
20
21     static int hashFunction(int num, int size) {
22         return num % size;
23     }
24
25     static int searchIdUsingHashFunction(int studentId, int size, Node[] hashTable) {
26         int counter = 0;
27         boolean idFound = false;
28
29         int hashIndex = hashFunction(studentId, size);
30
31         Node traversalNode = hashTable[hashIndex];
32
33         if (traversalNode != null) {
34             counter++;
35             while(traversalNode != null) {
36                 if (traversalNode.value == studentId) {
37                     idFound = true;
38                     break;
39                 }
40
41                 traversalNode = traversalNode.next;
42                 counter++;
43             }
44
45             if (idFound) {
46                 return counter;
47             } else {
48                 return 0;
49             }
50 }
```

```
51     }
52     return 0;
53 }
54
55
56 public static void main(String[] args) {
57     int[] studentArray = new int[500];
58
59     HashMap<Integer, Integer> hashSearchCounter = new HashMap<>();
60     HashMap<Integer, Integer> linearSearchCounter = new HashMap<>();
61
62     // Generate an array of students with random roll numbers
63     for(int i = 0; i < 500; i++) {
64         studentArray[i] = Hashing.getRandomStudentId();
65     }
66
67     System.out.print("Printing roll numbers of all students: ");
68     for(int i = 0; i < 500; i++) {
69         System.out.print(studentArray[i] + ", ");
70     }
71
72     // Prime number between 128 and 256
73     int size = 139;
74     Node[] hashTable = new Node[size];
75
76     // Filling the hashtable
77     int hashIndex;
78     for(int i = 0; i < 500; i++) {
79         hashIndex = hashFunction(studentArray[i], size);
80         Node tempNode = new Node(studentArray[i]);
81
82         // If the hash index is not found set the tempNode as the first element
83         if(hashTable[hashIndex] == null) {
84             hashTable[hashIndex] = tempNode;
85         } else {
86             // If the hash index is found, traverse till the end of the list and
then add the new node
87             Node traversalNode = hashTable[hashIndex];
88             while(traversalNode.next != null) {
89                 traversalNode = traversalNode.next;
90             }
91
92             traversalNode.next = tempNode;
93         }
94     }
95
96     System.out.println("\n");
97     System.out.println("Displaying the Hashtable Index with all the contents");
98
99     for(int i = 0; i < size; i++) {
```

```
100         Node temp = hashTable[i];
101
102         if (temp != null) {
103             System.out.print(i + "-> \t");
104
105             while(temp.next != null) {
106                 System.out.print(temp.value + ", ");
107                 temp = temp.next;
108             }
109             System.out.println(temp.value);
110         }
111     }
112
113     int foundCounter = 0, notFoundCounter = 0;
114
115     int[] searchArray = new int[20];
116     int searchArrayIndex = 0;
117
118
119     System.out.println("\nSearching the elements using Hash Function");
120     while (foundCounter < 17 || notFoundCounter < 3) {
121         int searchId = getRandomStudentId();
122
123         int counter = searchIdUsingHashFunction(searchId, size, hashTable);
124
125         if(counter == 0 && notFoundCounter < 3) {
126             System.out.println("ID: " + searchId + " was not found in hash table
with counter: " + counter);
127             hashSearchCounter.put(searchId, counter);
128             notFoundCounter++;
129
130             searchArray[searchArrayIndex] = searchId;
131             searchArrayIndex++;
132
133         } else if (counter > 0 && foundCounter < 17) {
134             System.out.println("ID: " + searchId + " was found in hash table with
counter: " + counter);
135             hashSearchCounter.put(searchId, counter);
136             foundCounter++;
137
138             searchArray[searchArrayIndex] = searchId;
139             searchArrayIndex++;
140         }
141     }
142
143
144     System.out.println("\nSearching the elements using Sequential Search
Function");
145
146     for(int i = 0; i < searchArray.length; i++) {
147         int searchId = searchArray[i];
```

```
148         int counter = 0;
149         boolean idFound = false;
150
151         for (int j = 0; j < size; j++) {
152             Node traversalNode = hashTable[j];
153
154             while(traversalNode != null) {
155
156                 if(traversalNode.value == searchId) {
157                     idFound = true;
158                     break;
159                 }
160
161                 traversalNode = traversalNode.next;
162                 counter++;
163             }
164
165             if (idFound) {
166                 break;
167             }
168         }
169
170         if (idFound) {
171             System.out.println("ID: " + searchId + " was found in hash table with
counter: " + counter);
172             linearSearchCounter.put(searchId, counter);
173         } else {
174             System.out.println("ID: " + searchId + " was not found in hash table
with counter: " + counter);
175             linearSearchCounter.put(searchId, counter);
176         }
177     }
178
179
180     System.out.println("\n\nComputational complexity of Linear and Sequential
Search");
181     System.out.println("\tNumber | Sequential | \tHash ");
182
183     for(int i = 0; i < searchArray.length; i++) {
184         System.out.println("\t"+ searchArray[i] + " | \t" +
linearSearchCounter.get(searchArray[i]) + " | \t" +
hashSearchCounter.get(searchArray[i]));
185     }
186 }
187 }
```

```

ID: 5856 was not found in hash table with counter: 0
ID: 4468 was found in hash table with counter: 1
ID: 2717 was found in hash table with counter: 6
ID: 4685 was found in hash table with counter: 2
ID: 7569 was found in hash table with counter: 1
ID: 4685 was found in hash table with counter: 2
ID: 9213 was found in hash table with counter: 1
ID: 8408 was found in hash table with counter: 2
ID: 4872 was found in hash table with counter: 2
ID: 2083 was found in hash table with counter: 5
ID: 9735 was found in hash table with counter: 2
ID: 8907 was found in hash table with counter: 3
ID: 7948 was found in hash table with counter: 4
ID: 9717 was found in hash table with counter: 2
ID: 2507 was found in hash table with counter: 9
ID: 4317 was found in hash table with counter: 5
ID: 2969 was found in hash table with counter: 3
ID: 6984 was found in hash table with counter: 2

```

Searching the elements using Sequential Search Function

```

ID: 6018 was not found in hash table with counter: 500
ID: 1189 was not found in hash table with counter: 500
ID: 5856 was not found in hash table with counter: 500
ID: 4468 was found in hash table with counter: 71
ID: 2717 was found in hash table with counter: 290
ID: 4685 was found in hash table with counter: 364
ID: 7569 was found in hash table with counter: 233
ID: 4685 was found in hash table with counter: 364
ID: 9213 was found in hash table with counter: 140
ID: 8408 was found in hash table with counter: 253
ID: 4872 was found in hash table with counter: 27
ID: 2083 was found in hash table with counter: 498
ID: 9735 was found in hash table with counter: 17
ID: 8907 was found in hash table with counter: 44
ID: 7948 was found in hash table with counter: 83
ID: 9717 was found in hash table with counter: 460
ID: 2507 was found in hash table with counter: 24
ID: 4317 was found in hash table with counter: 33
ID: 2969 was found in hash table with counter: 185
ID: 6984 was found in hash table with counter: 126

```

Computational complexity of Linear and Sequential Search

Number	Sequential	Hash
6018	500	0
1189	500	0
5856	500	0
4468	71	1
2717	290	6
4685	364	2
7569	233	1
4685	364	2
9213	140	1
8408	253	2
4872	27	2
2083	498	5
9735	17	2
8907	44	3
7948	83	4
9717	460	2
2507	24	9
4317	33	5
2969	185	3
6984	126	2

rohitkrishnanvidyasagar@Rohits-MacBook-Air assignment-6-hashing %