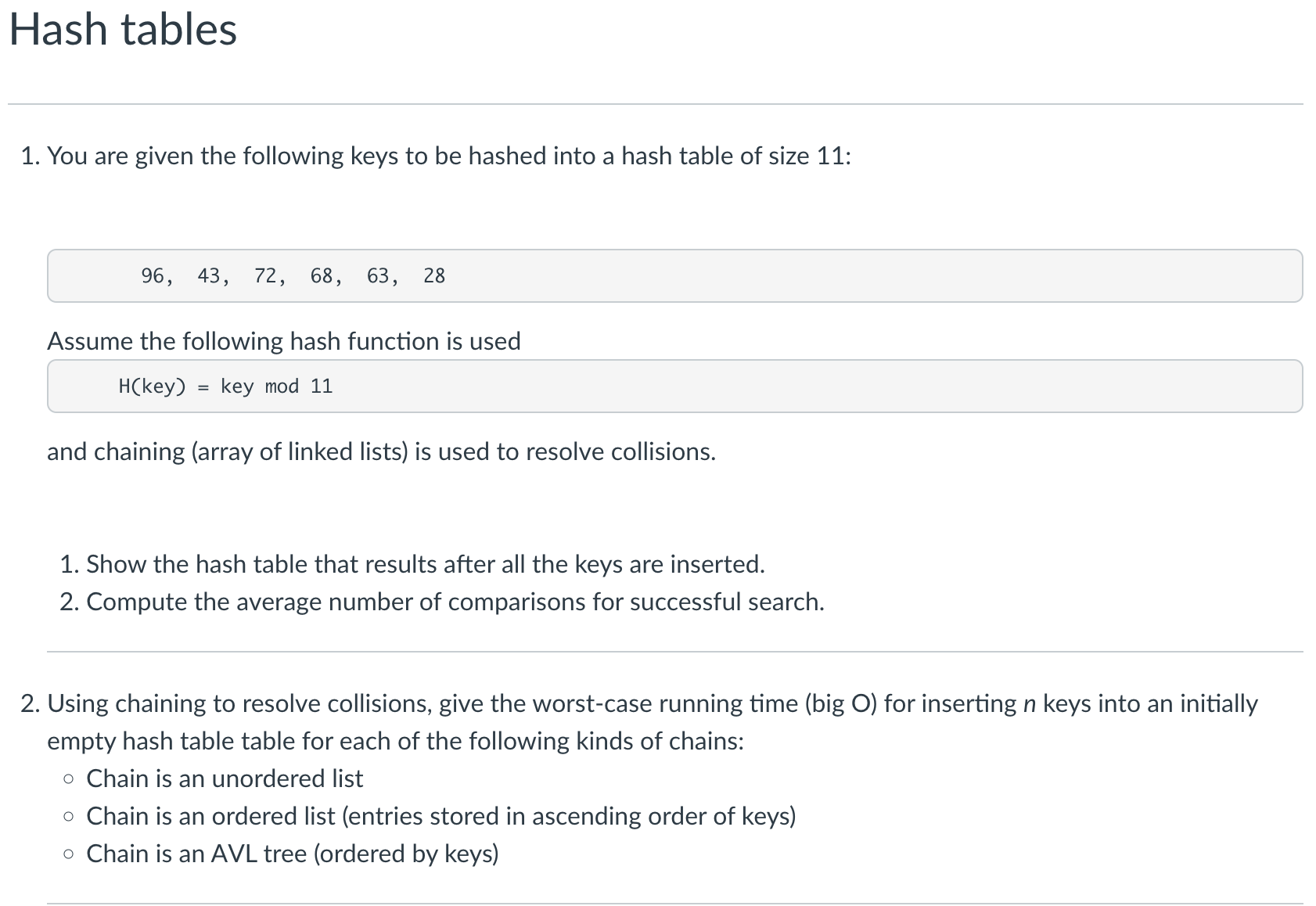
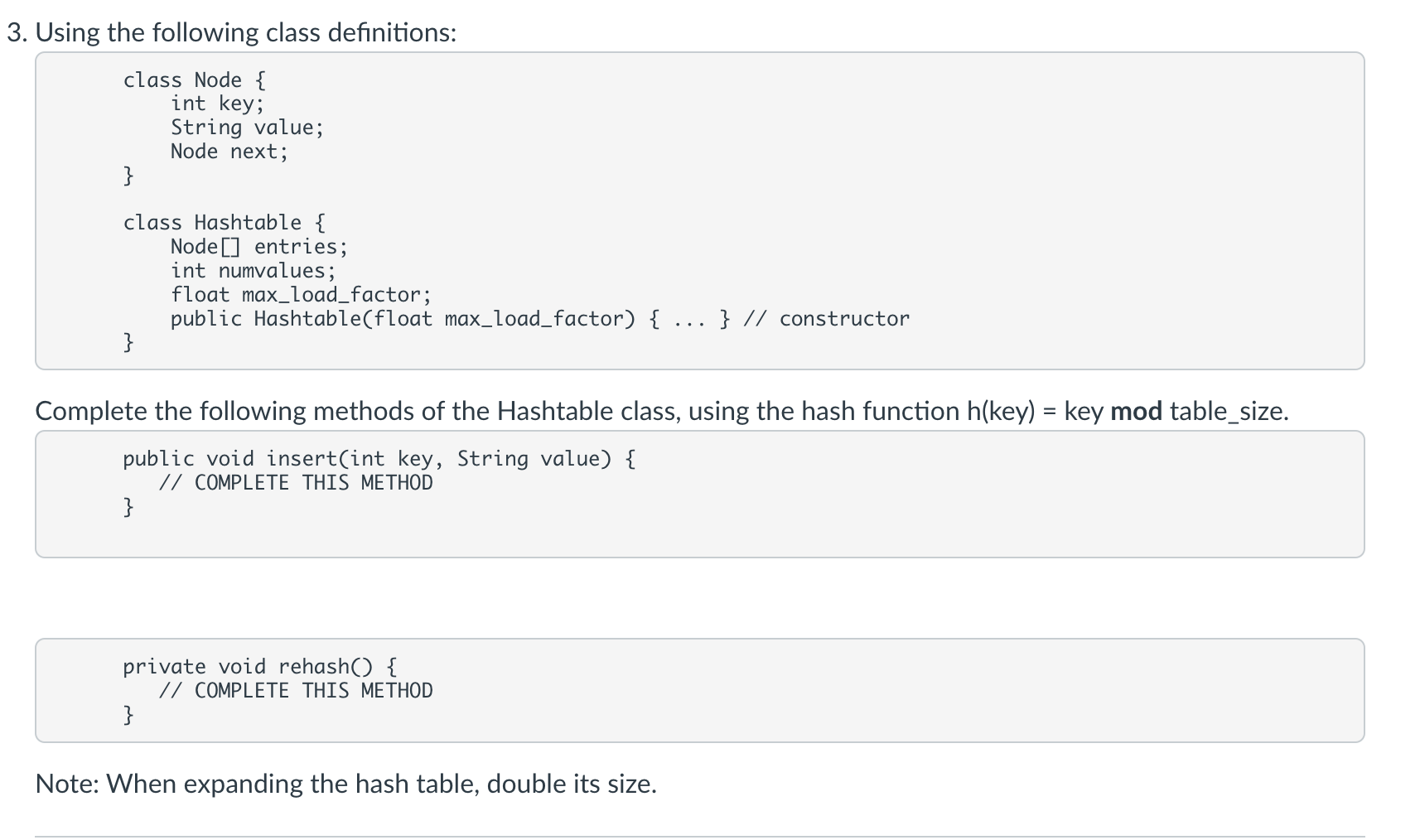
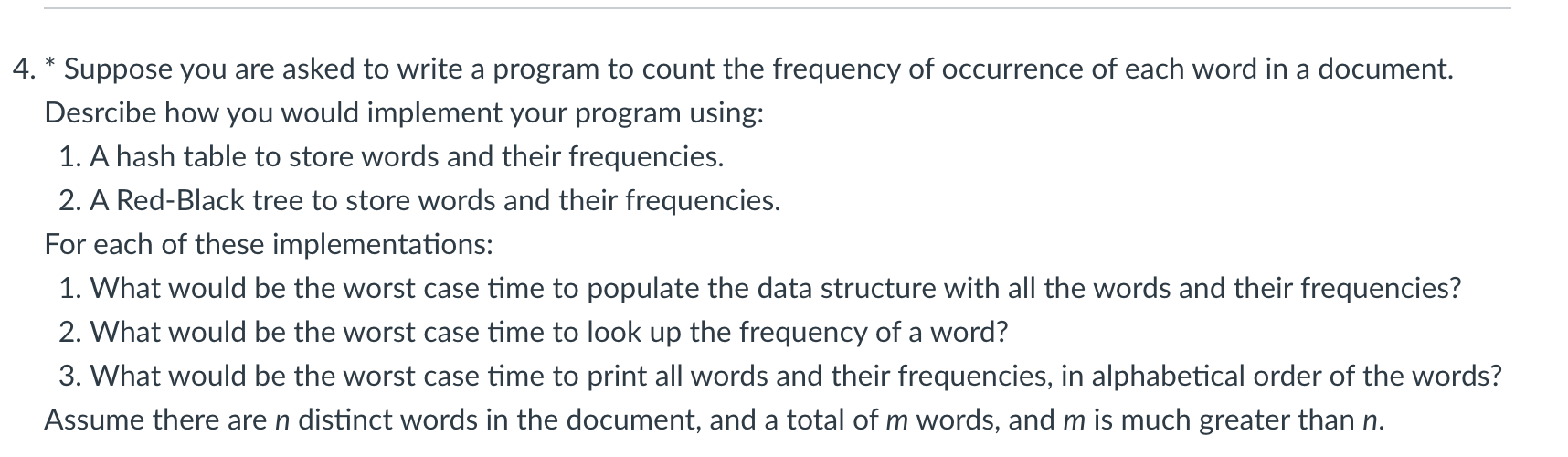
****

****

****

1)

a)

| 0 |  |  |
| --- | --- | --- |
| 1 |  |  |
| 2 | 68 |  |
| 3 |  |  |
| 4 |  |  |
| 5 |  |  |
| 6 | 28 | 72 |
| 7 |  |  |
| 8 | 63 | 96 |
| 9 |  |  |
| 10 | 43 |  |
| 11 |  |  |

b)

68 - 1 comparisons

72 - 1 comps

28 - 2 comps

96 - 1 comps

63 - 2 comps

43 - 1 comps

(1+1+2+1+2+1) / 6 = 1 1/3

Average of 1 ⅓ comparisons per successful search.

2) When Chain is:

* Unordered List insertion time: **O(n)**
  + Entry is added to front of list in its index.
* Ordered Linked List insertion time: **O(n^2)** 
  + Entry has to iterate through the list in its index to get to its position.
* AVL (balanced) Tree insertion time: **O(n lg(n))** 
  + Entry has to iterate through the height of the tree in its index to get to its position.

3)

public void insert(int key, String value) {

int hash = key % entries.length;

int i = hash & (entries.length - 1);

Node n = entries[i];

while (n != null) {

if (key.equals(n.key)) {

n.value = value;

return;

}

n = n.next;

}

n = entries[i];

Node newNode = new Node();

newNode.key = key;

newNode.value = value;

newNode.next = n;

entries[i] = newNode;

numvalues++;

if (numvalues > entries.length) {

rehash();

}

return;

}

private void rehash() {

Node[] prevEntries = entries;

Node[] newEntries = new Node[entries.length \* 2];

for (int x = 0; x < entries.length; x++) {

Node n = prevEntries[x];

while (n != null) {

Node next = n.next;

int hash = n.key % newEntries.length;

int i = hash & (newEntries.length - 1);

n.next = newEntries[i];

newEntries[i] = n;

n = next;

}

}

this.entries = newEntries;

return;

}

4)

1. Hash Table: Use the characters in the words as the keys for where they go in the table. The frequencies are stored inside each word’s node.
2. RB Tree: Every word would be added alphabetically to the tree. For each non-unique word, do not add the same word, but increase the frequency by 1 for the existing word in the tree.
3. Hash Table
   1. Populate time: O(m\*n) - Iterates through total words and table.
   2. Lookup time: O(n) - Iterates through table.
   3. Print time: O(n) - Iterates through table.
4. RB Tree:
   1. Populate time: O(m\*lg(n)) - Iterates through total words and height of tree.
   2. Lookup time: O(lg(n)) - Iterates through single branch of the tree.
   3. Print time: O(n) - Iterates through every element of RB Tree.