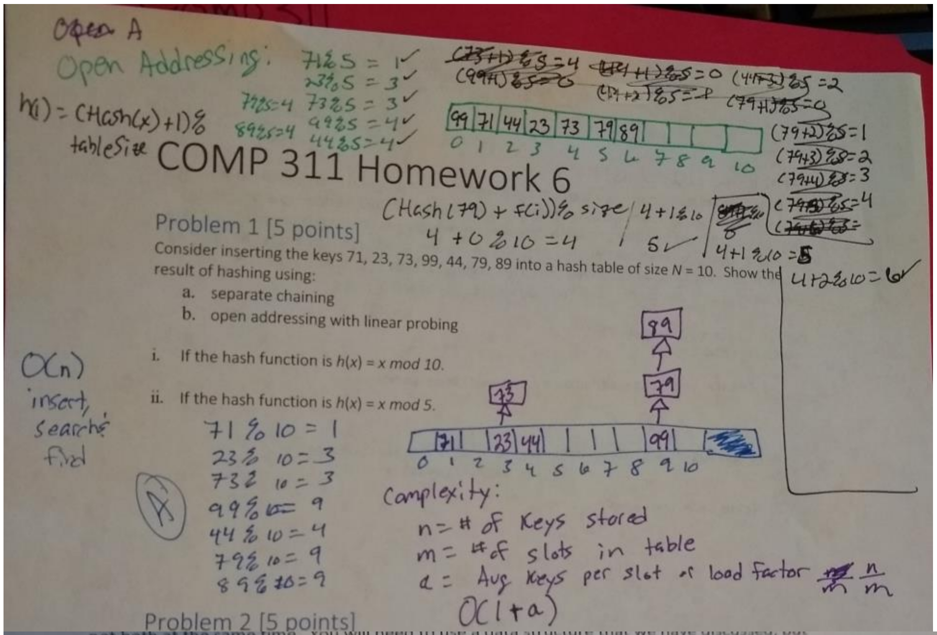
Holly Robertson
COMP 311 – W19
March 7, 2019
Homework 6

Diagrams made in Visio by me

Problem 1 [5 points]

Consider inserting the keys 71, 23, 73, 99, 44, 79, 89 into a hash table of size *N* = 10. Show the result of hashing using:

a. separate chaining
b. open addressing with linear probing



Open A

Open Addressing: 71%5 = 1✓
23%5 = 3✓
73%5=4 73%5 = 3✓
89%5=4 99%5 = 4✓
44%5=4✓

h(i) = (Hash(x)+1)% tableSize

~~(73+1)%5=4~~ ~~(44+1)%5=0~~ (44+3)%5 =2
~~(99+1)%5=0~~ ~~(44+2)%5=1~~ ~~(79+1)%5=0~~
(79+2)%5=1
(79+3)%5=2
(79+4)%5=3
(79+5)%5=4

99	71	44	23	73	79	89				
0	1	2	3	4	5	6	7	8	9	10

COMP 311 Homework 6

(Hash(79) + f(i))% size
4 + 0 % 10 = 4 | 4+1%10 5✓ | 4+1%10 = 5 | 4+2%10=6✓

Problem 1 [5 points]

Consider inserting the keys 71, 23, 73, 99, 44, 79, 89 into a hash table of size *N* = 10. Show the result of hashing using:

a. separate chaining
b. open addressing with linear probing

i. If the hash function is *h*(*x*) = *x mod 10*.

ii. If the hash function is *h*(*x*) = *x mod 5*.

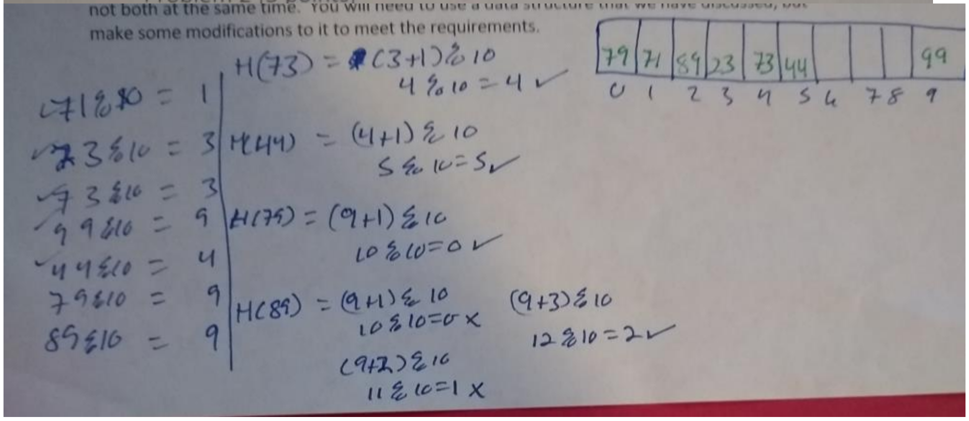
O(n) insert, search, find

71 % 10 = 1
23 % 10 = 3
73 % 10 = 3
99 % 10 = 9
44 % 10 = 4
79 % 10 = 9
89 % 10 = 9

	71		23 73	44					99 79 89	
0	1	2	3	4	5	6	7	8	9	10

Complexity:
n = # of Keys stored
m = # of slots in table
α = Avg keys per slot or load factor $\frac{n}{m}$
O(1+α)

Problem 2 [5 points]



not both at the same time. You will need to use a data structure that we have discussed, but make some modifications to it to meet the requirements.

79	71	89	23	73	44				99
0	1	2	3	4	5	6	7	8	9

71%10 = 1
73%10 = 3
73%10 = 3
99%10 = 9
44%10 = 4
79%10 = 9
89%10 = 9

H(73) = (3+1)%10
4%10 = 4✓

H(44) = (4+1)%10
5%10 = 5✓

H(79) = (9+1)%10
10%10 = 0✓

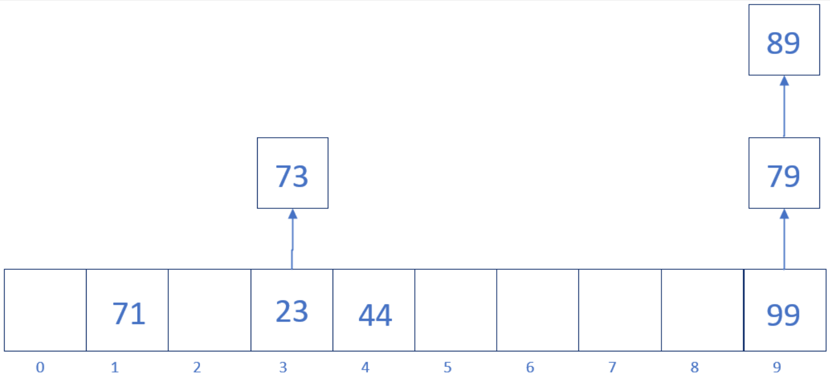
H(89) = (9+1)%10
10%10 = 0 ×
(9+2)%10
11%10 = 1 ×
(9+3)%10
12%10 = 2✓

Separate Chaining

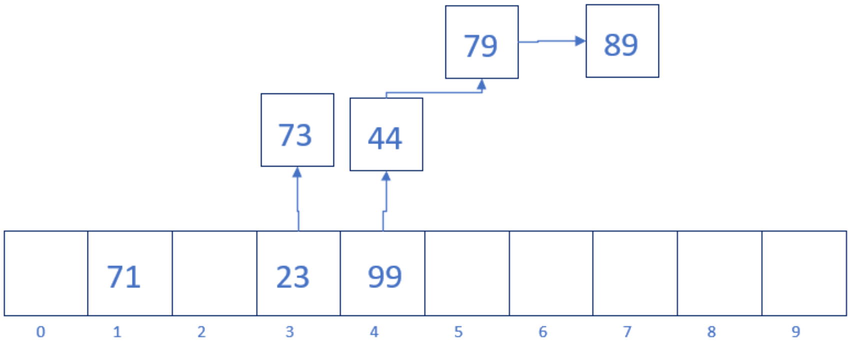
i. h(x) % 10

ii. h(x) % 5

71 % 10 = 1
23 % 10 = 3
73 % 10 = 3
99 % 10 = 9
44 % 10 = 4
79 % 10 = 9
89 % 10 = 9



71 % 5 = 1
23 % 5 = 3
73 % 5 = 3
99 % 5 = 4
44 % 5 = 4
79 % 5 = 4
89 % 5 = 4



Open Addressing – Linear Probing

i. h(x) % 10
ii. h(x) % 5

Collision: ([h(x) + f(i)] % table size)

71 % 10 = 1
23 % 10 = 3
73 % 10 = 3
99 % 10 = 9
44 % 10 = 4
79 % 10 = 9
89 % 10 = 9

79	71	89	23	73	44				99
0	1	2	3	4	5	6	7	8	9

71 % 5 = 1
23 % 5 = 3
73 % 5 = 3
99 % 5 = 4
44 % 5 = 4
79 % 5 = 4
89 % 5 = 4

99	71	44	23	73	79	89			
0	1	2	3	4	5	6	7	8	9

Problem 2 [3/5 points]

Design a data structure that can perform the given operations in the stated worst-case running time:

Insert: $O(\lg n)$
Maximum: $O(1)$
Minimum: $O(1)$
Extract-max: $O(\lg n)$
Extract-min: $O(\lg n)$

This will be a single data structure, not a different data structure for each operation. Keep in mind that it must have this performance for all of the methods at once, so a heap does not satisfy the requirements – a heap has O(1) performance for extract minimum or extract maximum, but not both at the same time. You will need to use a data structure that we have discussed, but make some modifications to it to meet the requirements.
LinkedList Source Priority Queue Source Huffman Source BinarySearch Tree Source ArrayList Source

Data Structures:
HashSet / LinkedHashSet / TreeSet

- HashSet – not ordered
    - Add/remove/contains – O(1)
- TreeSet – Ordered
    - Add/remove/contains – O(log n)

  LinkedHashSet – Hash Table - Ordered
    - Add/remove/contains – O(1)

Recursive Data Structure (LinkedList, This means linear search is an **O**(n) algorithm.)

- LinkedList
    - Add– O(1)
    - Remove – O(n)
    - Access – O(n)
    - Search – O(n)

BinaryTree (Binary Search Tree)

- Binary Search Tree
    - Add/Remove – O(n)
    - Access – O(n)
    - Search – O(n)

Priority Queues

- PriorityQueues
    - Add/Delete – (O(log n))
    - Retrieve Max/Min – O(1)
    - Contains – O(n)

Huffman Trees

- HuffmanTree
    - Retrieveal – O(log n)

Stack

- Stack
    - Push – O(1)
    - Pop – O(1)
    - Peek – O(1)
    - Search – O(n)

ArrayList

- ArrayList
    - Add – O(1)
    - Remove – O(n)
    - Search – O(n)

Answer****

PriorityQueue

“Implementation note: this implementation provides O(log(n)) time for the enqueing and dequeing methods (offer, poll, remove() and add); linear time for the remove(Object) and contains(Object) methods; and constant time for the retrieval methods (peek, element, and size).”

PriorityQueues pull out info by the minimum value. The lower the value, the higher the priority. The compare method compares and returns the lowest value. When an item is removed, then the bottom leaf is replaced at the root and then compared again, this is why it is linear. The insert methods are logarithmic because it compares against the root node and then traverses whichever side of the queue it aligns with. Extract max and min is the logarithmic as well since it has evaluate the root node and then traverse the sides and compare each to find the max/min node.

- add()
- remove()
- peek()
- contains()

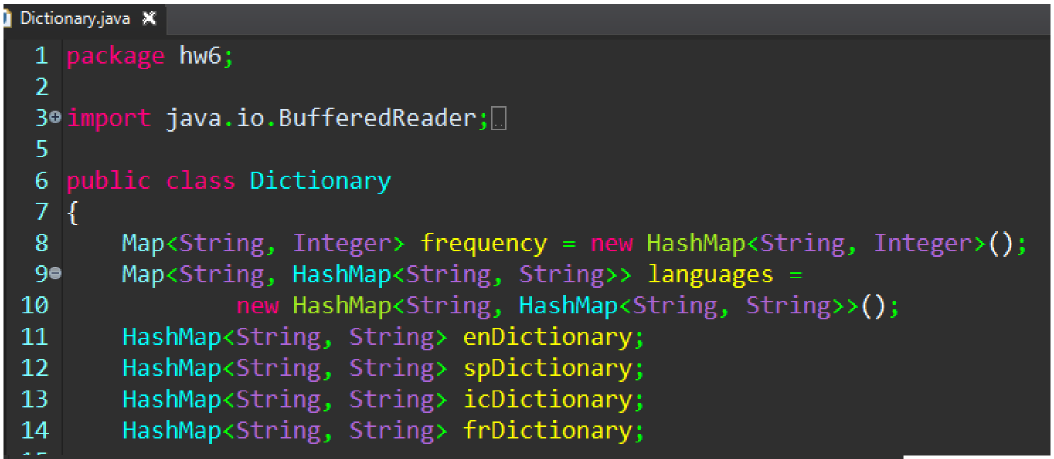
    This almost works, but a PQ can do either extractMin or extractMax fast, but not both.

Formatted: Indent: Left: 0.5", No bullets or numbering

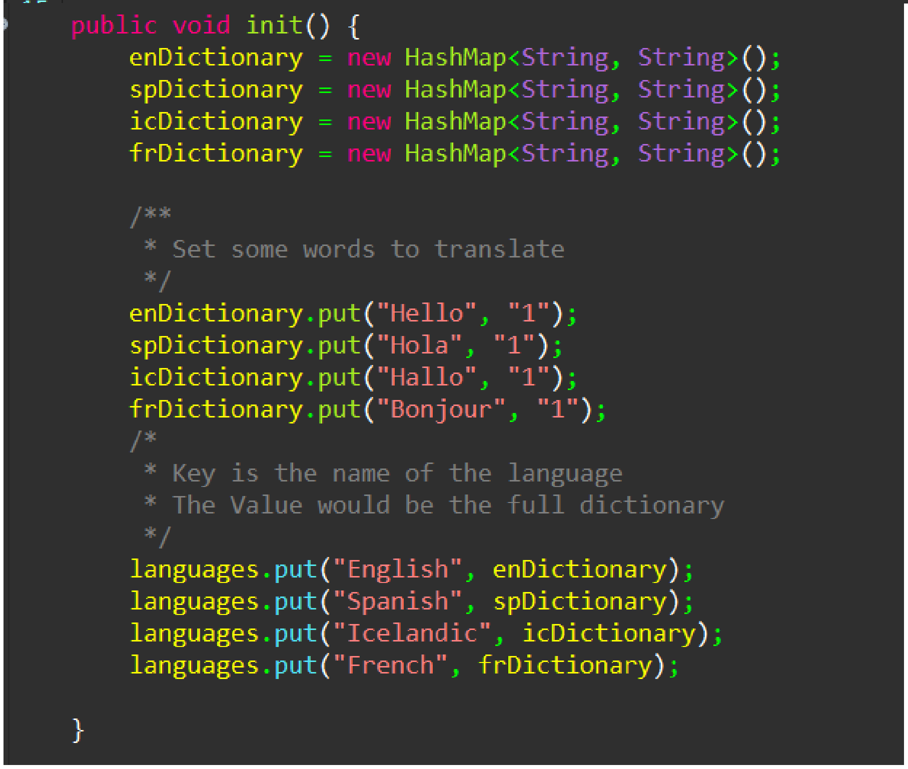
Problem 3 [8/10 points]

Write a program that simulates a translation dictionary. Your program must be able to support any number of words and multiple languages. All translations should be done as efficiently as possible, and it must be able to translate between any two languages. Your code should not care what languages exist in the dictionary – it should be possible to add more languages by changing the code in `init()`, and not touching the code in `translateWord()`. **Indicate the time complexity of your algorithm**. Hint: You may want to use a map of maps.

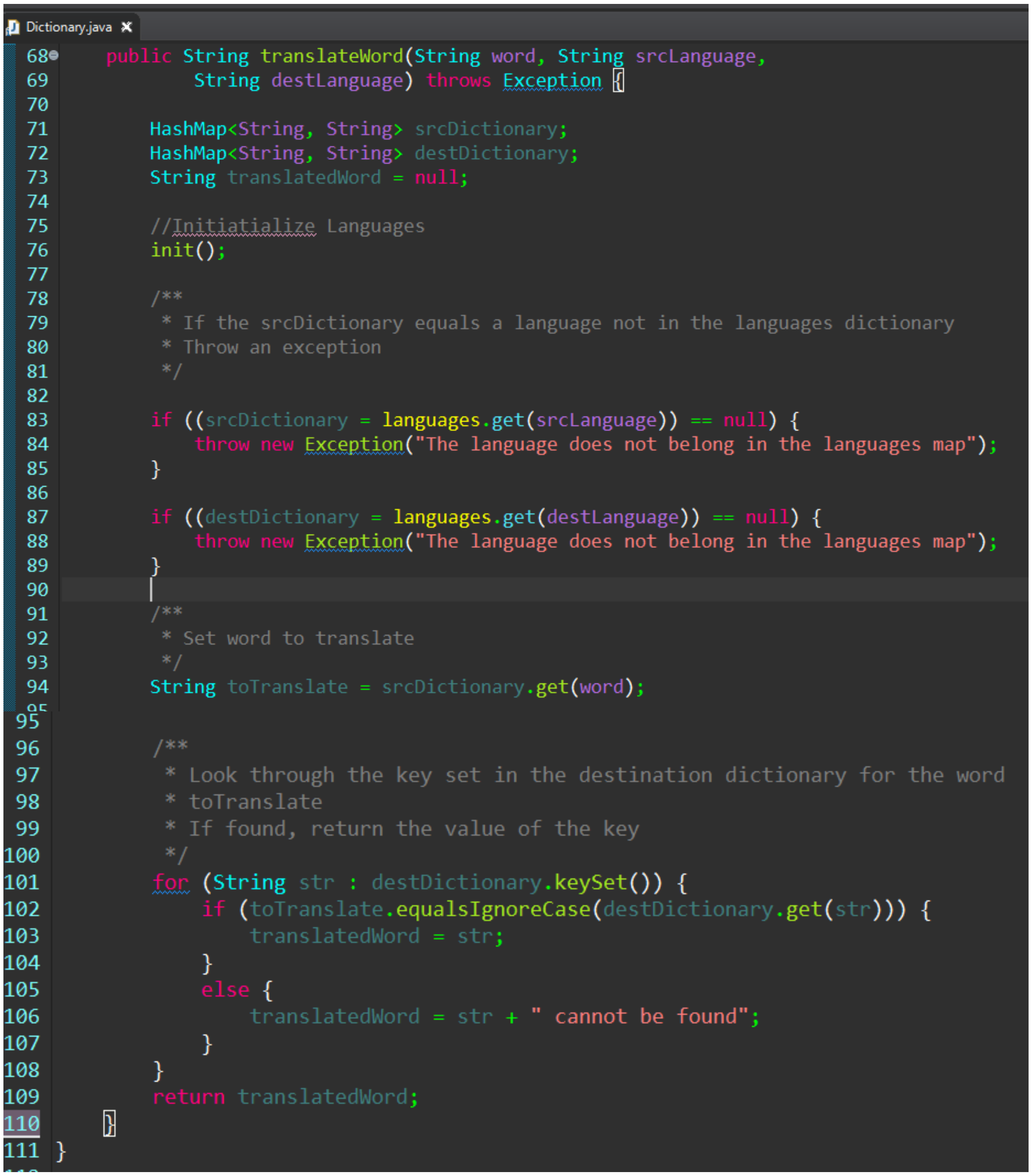
Formatted: Font: Bold



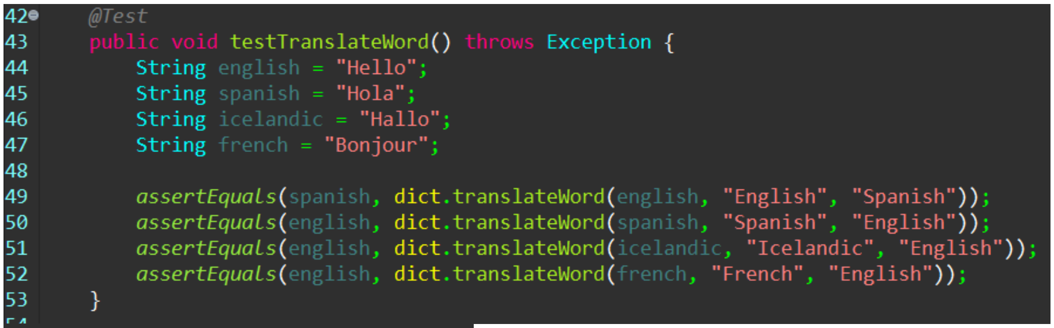
```java
package hw6;

import java.io.BufferedReader;

public class Dictionary
{
    Map<String, Integer> frequency = new HashMap<String, Integer>();
    Map<String, HashMap<String, String>> languages =
            new HashMap<String, HashMap<String, String>>();
    HashMap<String, String> enDictionary;
    HashMap<String, String> spDictionary;
    HashMap<String, String> icDictionary;
    HashMap<String, String> frDictionary;
```



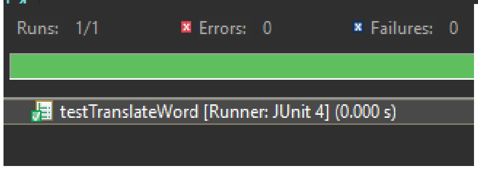
```java
    public void init() {
        enDictionary = new HashMap<String, String>();
        spDictionary = new HashMap<String, String>();
        icDictionary = new HashMap<String, String>();
        frDictionary = new HashMap<String, String>();

        /**
         * Set some words to translate
         */
        enDictionary.put("Hello", "1");
        spDictionary.put("Hola", "1");
        icDictionary.put("Hallo", "1");
        frDictionary.put("Bonjour", "1");
        /*
         * Key is the name of the language
         * The Value would be the full dictionary
         */
        languages.put("English", enDictionary);
        languages.put("Spanish", spDictionary);
        languages.put("Icelandic", icDictionary);
        languages.put("French", frDictionary);

    }
```



```java
    public String translateWord(String word, String srcLanguage,
            String destLanguage) throws Exception {

        HashMap<String, String> srcDictionary;
        HashMap<String, String> destDictionary;
        String translatedWord = null;

        //Initiatialize Languages
        init();

        /**
         * If the srcDictionary equals a language not in the languages dictionary
         * Throw an exception
         */

        if ((srcDictionary = languages.get(srcLanguage)) == null) {
            throw new Exception("The language does not belong in the languages map");
        }

        if ((destDictionary = languages.get(destLanguage)) == null) {
            throw new Exception("The language does not belong in the languages map");
        }

        /**
         * Set word to translate
         */
        String toTranslate = srcDictionary.get(word);

        /**
         * Look through the key set in the destination dictionary for the word
         * toTranslate
         * If found, return the value of the key
         */
        for (String str : destDictionary.keySet()) {
            if (toTranslate.equalsIgnoreCase(destDictionary.get(str))) {
                translatedWord = str;
            }
            else {
                translatedWord = str + " cannot be found";
            }
        }
        return translatedWord;
    }
}
```



```
    @Test
    public void testTranslateWord() throws Exception {
        String english = "Hello";
        String spanish = "Hola";
        String icelandic = "Hallo";
        String french = "Bonjour";

        assertEquals(spanish, dict.translateWord(english, "English", "Spanish"));
        assertEquals(english, dict.translateWord(spanish, "Spanish", "English"));
        assertEquals(english, dict.translateWord(icelandic, "Icelandic", "English"));
        assertEquals(english, dict.translateWord(french, "French", "English"));
    }
```



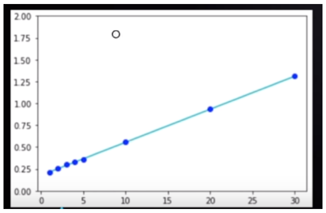
```
Runs:  1/1        Errors:  0        Failures:  0

testTranslateWord [Runner: JUnit 4] (0.000 s)
```

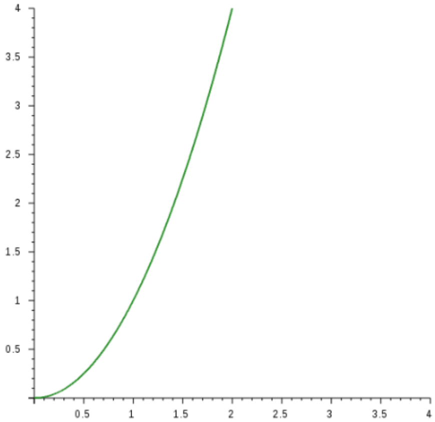
Reflection [5 points]

In two to three paragraphs of prose (i.e. sentences, not bullet lists) using APA style citations if needed, summarize and interact with the content that was covered in the class session this week. In your summary, you should highlight the major topics, theories, practices, and knowledge that were covered. Your summary should also interact with the material through personal observations, reflections, and applications to the field of study. In particular, highlight what surprised, enlightened, or otherwise engaged you. Make sure to include at least one thing that you're still confused about. In other words, you should think and write critically not just about what was presented but also what you have learned through the session. Feel free to ask questions in this as well since it will be returned to you with answers.

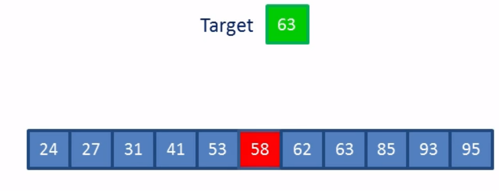
My definition and explanation of Time Complexity Source 2nd Source 3rd Source 4th Source

- The time it takes to run your function grows with the size of the function.
- Important
    - If you have two steps in your function, you add those steps
        - touch each element in Array A = O(n)
        - touch each element in Array B = O(x)
            - T = O(n + x)
        - Since each array is a singular "list" of *n/x* items, they're separate functions that are running in the same function.
    - If you have two nested steps in your function, you multiply those steps
        - touch each element in Array A = O(n)
        - compare each element in Array A against Array B
        - touch each element in Array B = O(n)
            - T = O(a * b)
- Simple Array
    - Sum of array
        - Have to touch each element in the array and then add the elements together (this step would affect space, not time though – in a meaningful BigO way – O(1))
        - The runtime of this function grows as the size of the array grows
        - If there are 3 elements, it would take 3 touches – N Time
        - This means the time complexity runs linearly -
            - T = *(an + b)*

        - Take the fastest growing time = *(an)*
        - Take the coefficient out = *O(n)*
            - 

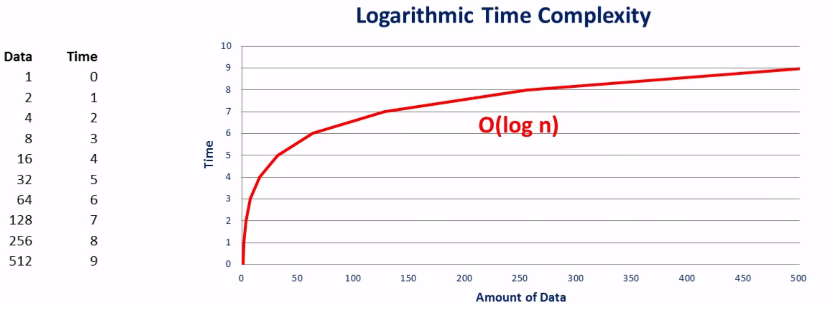
- Two Dimensional Array
    - Sum of array
        - Have to **touch** row and then **touch** each element in that row and then add the elements in the array
            - set total -> O(1)
            - **touch** each row -> O(n)
            - **touch** each element in the row -> O(n)
            - total += total -> O(1)
            - return total -> O(1)
        - n^2 * *O(1) + O(1) + O(1)* = O(n^2)
        - 

- Binary Search
    - Used to search an ordered list for a particular value (**like a Binary Search Tree**)
    - Target is compared with root, then traverses the subtree that is compatible with the value being searched, the other half of the search tree is discarded. Very efficient for large sorted list



- The left half is discarded and so on.
- When you double the amount of data in tree, you only have to “chop” the array one more time.

- The algorithm is
    - $n * (1/2)^k = 1$
        - n = number of elements in array
        - k = worst cast # of times needed to “split” array
    - To convert it to a logarithmic expression
        - $n * (1/2^k) = 1$
        - $n^k * (n/2^k) = 2^k$
            - $2^k(n^k) * (2^k(n/2^k)) = 2^k(2^k)$
        - $n = 2^k$
        - $\log_2 n = k$
- So the distance traversed is double, but the time needed is only one extra “tick”



Data	Time
1	0
2	1
4	2
8	3
16	4
32	5
64	6
128	7
256	8
512	9

My question this week is not related to the class at all, because my brain is a little fried. I took and passed the AWS Solutions Architect + the lab + the homework + my other courses + I have the flu = brain and body fried. My question this week is – are you excited for Game of Thrones in a month?