CSE 12 Week 4 Discussion

1-26-21

Focus: Counting Step & Runtime Analysis

Reminders

- PA3 due tomorrow at 11:59 PM
- PA4 is released Thursday, is an **open** assignment - collaborate!

- PA1 Resubmission due Friday, January 29th 11:59 PM
- PA2 Resubmission due Friday, February 5th 11:59 PM

Do you feel prepared for the Midterm?

A. Going to ace it!

B. Almost there

C. ¯_(ツ)_/¯

D. not ready at all

E. What midterm?

Counting Steps...

How many times does the following loop run?

```
for (int i = 1; i < 1000; i*=2) {
    System.out.println(i);
}
```

A. 100
B. 50
C. 25
D. 10

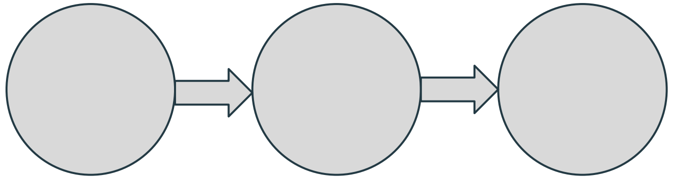
Answer - D

Loop index increments by a factor of 2 each iteration:

i = 1...2...4...8...16...32...64...128...256...512 **(10 times total)**

And fails to run when i = 1024 since i > 1000

Counting Steps...

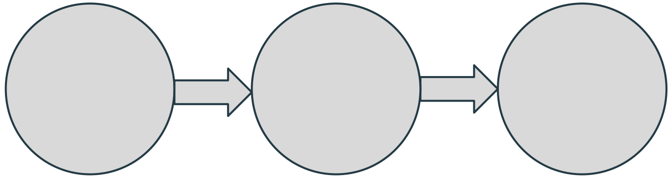


```
//LINKED LIST THAT HOLDS A STRING IN EACH NODE
boolean find(String toFind) {
  Node current = this.front.next;
  while(current != null) {
    if(current.value.equals(toFind)) {return true;}
    current = current.next;
  }
  return false;
}
```

We have a Linked List containing 10 nodes (including the dummy node). If we are searching for “Jerry” using find() but the Linked List does not contain “Jerry”, how many times will the while loop condition execute?

A. 9
B. 1
C. 3
D. 10

Answer - D



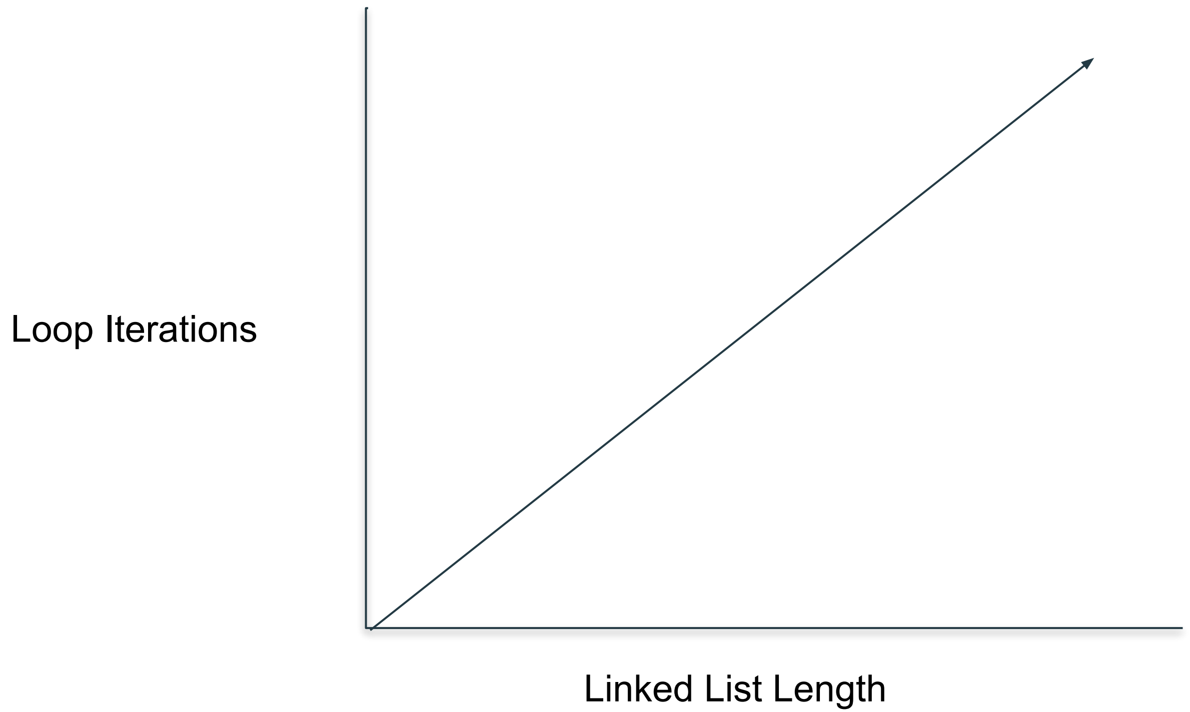
Considering the LinkedList has 10 nodes, we will loop through all Nodes until a match is found.

There will be no match in this case since “Jerry” is not in the LinkedList.

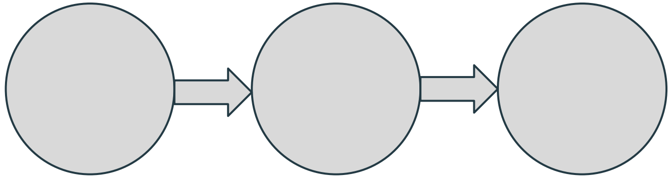
We will therefore go through the loop condition 10 times.

```
//LINKED LIST THAT HOLDS A STRING IN EACH NODE
boolean find(String toFind) {
  Node current = this.front.next;
  while(current != null) {
    if(current.value.equals(toFind)) {return true;}
    current = current.next;
  }
  return false;
}
```

Loop Iterations vs Linked List Length (when String to find is not in list)



Counting Steps...

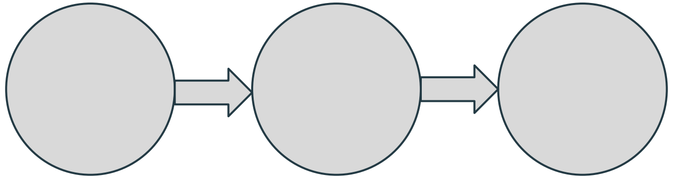


```
//LINKED LIST THAT HOLDS A STRING IN EACH NODE
boolean find(String toFind) {
  Node current = this.front.next;
  while(current != null) {
    if(current.value.equals(toFind)) {return true;}
    current = current.next;
  }
  return false;
}
```

If we are still searching for “Jerry” and it is found in the LinkedList, how many times will the while loop run?

A. 9
B. 1
C. 10
D. Can not be determined

Answer - D



```
//LINKED LIST THAT HOLDS A STRING IN EACH NODE
boolean find(String toFind) {
  Node current = this.front.next;
  while(current != null) {
    if(current.value.equals(toFind)) {return true;}
    current = current.next;
  }
  return false;
}
```

If “Jerry” is in the LinkedList, we do not know exactly how many Nodes from the start it is located.

It could be located at start of list, end, middle, anywhere...

Counting Steps...

```
//FIND FIRST MATCHING VALUE ACROSS BOTH LISTS

String findMatch(LinkedList a, LinkedList b) {
  Node aNode = a.first.next;
  for (int i = 0; i < a.size; i++) {
    Node bNode = b.first.next;
    for (int j = 0; j < b.size; j++) {
      if (aNode.value.equals(bNode.value) {
        return aNode.value;
      }
      bNode = bNode.next;
    }
    aNode = aNode.next;
  }
  return null;
}
```

Assuming that there are no matches between the two Linked Lists a and b, how many times would the inner if statement be executed?

A. a.size
B. b.size
C. a.size * b.size
D. a.size + b.size
E. None of the above

Answer - C

```
//FIND FIRST MATCHING VALUE ACROSS BOTH LISTS

String findMatch(LinkedList a, LinkedList b) {
  Node aNode = a.first.next;
  for (int i = 0; i < a.size; i++) {
    Node bNode = b.first.next;
    for (int j = 0; j < b.size; j++) {
      if (aNode.value.equals(bNode.value) {
        return aNode.value;
      }
      bNode = bNode.next;
    }
    aNode = aNode.next;
  }
  return null;
}
```

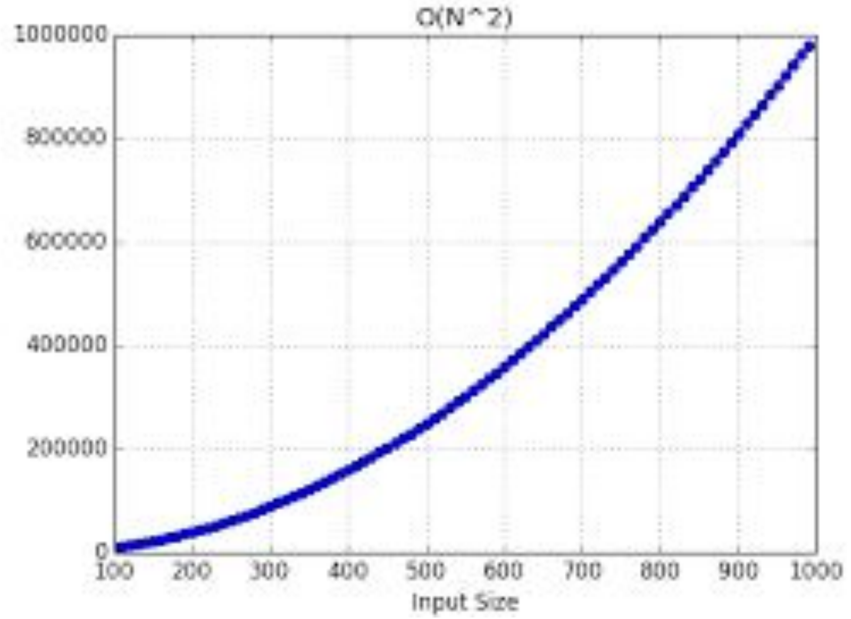
The inner if statement is inside both for loops.

Outer for loop runs a.size times.

Inner for loop runs b.size times.

Anything inside inner for loop is called a.size * b.size times.

If Statement Calls vs List Sizes (when no items match across two lists)



Categorizing Runtimes

Let $f(n) = 100$

Which of the following is true?

A. $f(n)$ is $O(2^n)$
B. $f(n)$ is $O(n^2)$
C. $f(n)$ is $O(n)$
D. All of these
E. None of these

Let $f(n) = 100$

Which of the following is true?

A. $f(n)$ is $O(2^n)$
B. $f(n)$ is $O(n^2)$
C. $f(n)$ is $O(n)$
[✓ D. All of these]
E. None of these

f_2 is $\Omega(f_1)$

$f(n) = \mathbf{O}(g(n))$, if there are positive constants c and n_0 such that $f(n) \leq c * g(n)$ for all $n \geq n_0$.

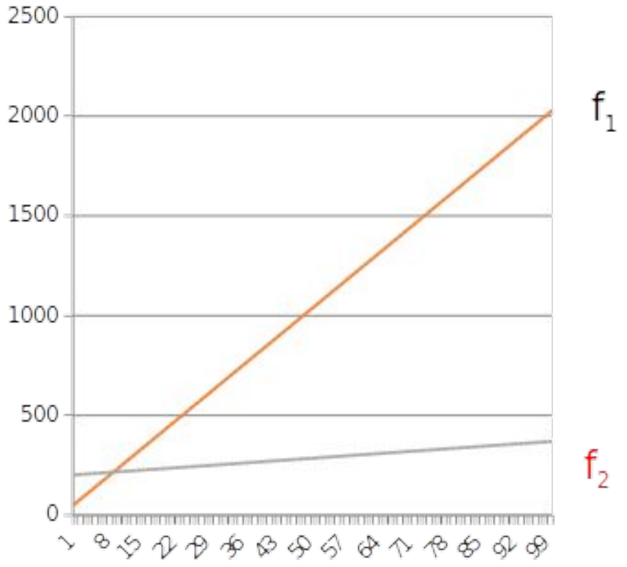
$f(n) = \mathbf{\Omega}(g(n))$, if there are positive constants c and n_0 such that $f(n) \geq c * g(n)$ for all $n \geq n_0$.

A. TRUE

B. FALSE

Why or why not?

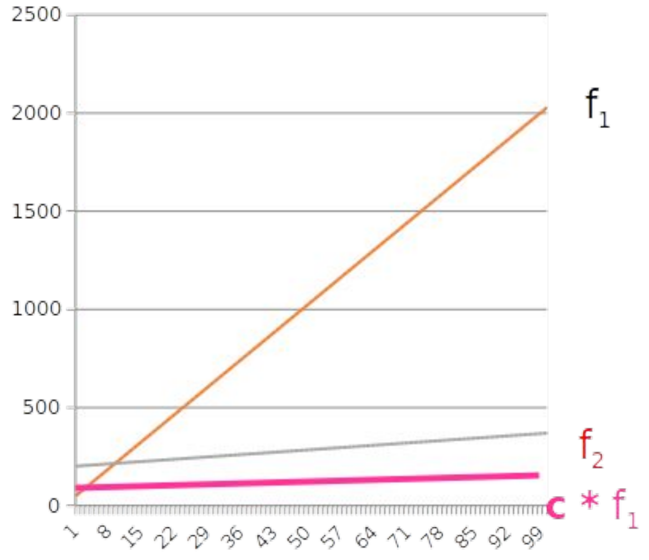
In other words, for large n, can you multiply f_1 by a positive constant and have it always be smaller than f_2



$f(n) = \mathbf{O}(g(n))$, if there are positive constants c and n_0 such that $f(n) \leq c * g(n)$ for all $n \geq n_0$.

$f(n) = \mathbf{\Omega}(g(n))$, if there are positive constants c and n_0 such that $f(n) \geq c * g(n)$ for all $n \geq n_0$.

- f_1 is $\Omega(f_2)$ because $f_1 > f_2$ (after about n=10, so we set $n_0 = 10$)
  - f_2 is clearly a ***lower bound*** on f_1 and that's what big-Ω is all about
- But f_2 is $\Omega(f_1)$ as well!
  - We just have to use the "**c**" to adjust so f_1 that it moves below f_2



Runtime Applications

- Functions:

  $f(n) = 0.23 * n + 6n^2 - \log(n)$

- Code:

```
boolean find(String toFind) {
  Node current = this.front.next;
  while(current != null) {
    if(current.value.equals(toFind)) {return true;}
    current = current.next;
  }
  return false;
}
```

- General:

  Data structure operations