CSE12 Discussion 3

Max Jiao

iClicker Participation

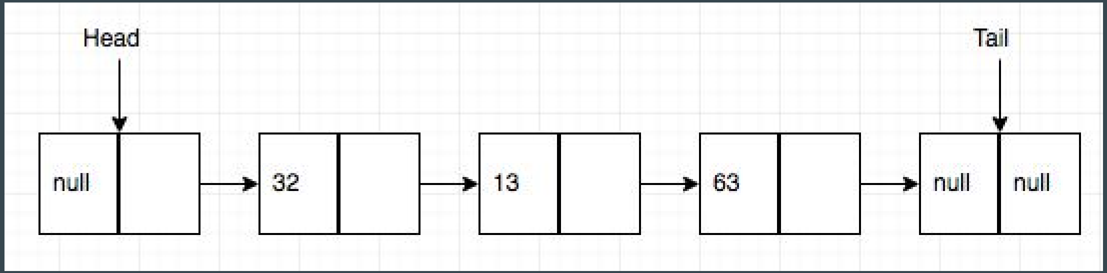
- There will be questions sprinkled over the 50 mins time slot.
- You must click at least 75% of the times to get any credit.
- You must attend the discussion the same week as that of the review quiz.

HW1 Grades are out!

- There is a pinned post on Piazza for regrades. If you have any VALID regrade request, make a follow-up post with the name of your Grader.
- Submission rules: If submission.txt says failed submission, then your submission is not graded.
    - **<u>Only for HW1</u>**: we will allow you to get a regrade, with a 10 point penalty.
- Please triple-check submission report for HW2 and all future homeworks.

Review Quiz

Consider the Linked List:

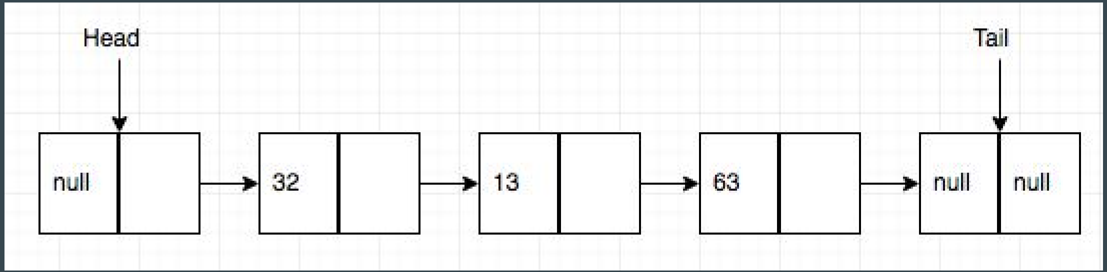


What would myList.get(3); return?

A. null
B. Tail
C. Runtime error
D. 63
E. Compile error

Review Quiz

Consider the Linked List:



What would myList.get(3); return?

A. null

B. Tail

C. Runtime error

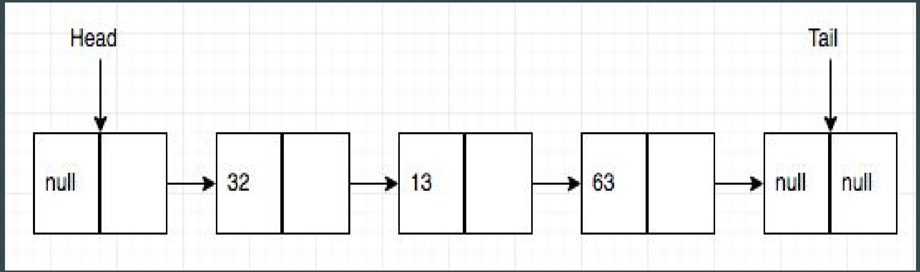
D. 63

E. Compile error

Review Quiz

Looking at the same linked list:

The following code would successfully insert the int 5 at index 0:




```
Node inserted = new Node(5);
this.head.setNext(inserted);
inserted.setNext(head.getNext());
```

A) True

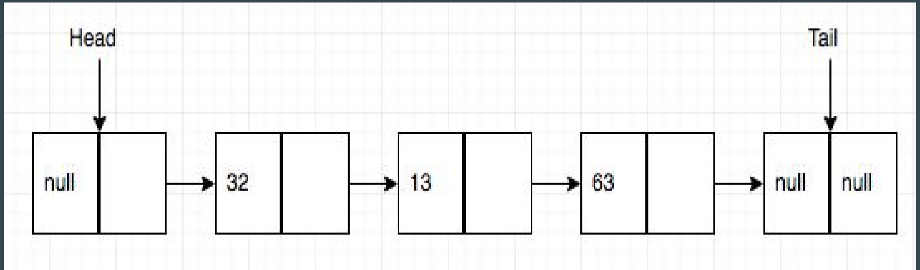
B) False

Review Quiz

Looking at the same linked list:

The following code would successfully insert the int 5 at index 0:

Node inserted = new Node(5);
this.head.setNext(inserted);
inserted.setNext(head.getNext());



A) True

B) False

Review Quiz

```java
interface QuizQuestion {

    void AskAQuestion(Object o);

    String importantMethod(String CorrectAnswer);
}

class Quiz implements QuizQuestion{

    public void AskAQuestion(Object o) {

        //do some stuff
    }

    private String importantMethod(String CorrectAnswer) {

        return CorrectAnswer;
    }
}
```

Considering the Code:

```java
Quiz q2 = new Quiz();
Object o = new Integer(4);
q2.AskAQuestion(o);
String x =
q2.importantMethod("helloworld")
;
System.out.println(x);
```

<u>what will happen?</u>

What will happen?

A) Code will not compile because o is an integer

B) Code will not compile because interface was implemented wrong

C) helloworld will be printed

D) runtime error NullPointerException

E) can't tell

Review Quiz

```java
interface QuizQuestion {

    void AskAQuestion(Object o);

    String importantMethod(String CorrectAnswer);
}

class Quiz implements QuizQuestion{

    public void AskAQuestion(Object o) {

        //do some stuff
    }

    private String importantMethod(String CorrectAnswer) {

        return CorrectAnswer;
    }
}

```

Considering the Code:

```java
Quiz q2 = new Quiz();
Object o = new Integer(4);
q2.AskAQuestion(o);
String x =
q2.importantMethod("helloworld")
;
System.out.println(x);
```

<u>what will happen?</u>

What will happen?

A) Code will not compile because o is an integer
B) Code will not compile because interface was implemented wrong
C) helloworld will be printed
D) runtime error NullPointerException
E) can't tell

Review Quiz

Which operations is a Singly LinkedList (with a head and a tail pointers) better at (less operations/faster) compared to an array (which has enough room to add elements). You do not have a direct access to the elements of the linked list, except the head, and the tail.

- deleting (at any given index)
- insertion (at the front)
- deletion (from the front)
- accessing data (at any index)
- searching (sorted list)
- searching (unsorted list)
- replacing an element (at any index)

Review Quiz

Which operations is a Singly LinkedList (with a head and a tail pointers) better at (less operations/faster) compared to an array (which has enough room to add elements). You do not have a direct access to the elements of the linked list, except the head, and the tail.

- deleting (at any given index)
- insertion (at the front)
- deletion (from the front)
- accessing data (at any index)
- searching (sorted list)
- searching (unsorted list)
- replacing an element (at any index)

Review Quiz

A typical Node in a (Singly) LinkedList class will have which fields?
(choose all that apply)

-Note that the exact names of the fields may actually be different, so assume (or some equivalent) for each one.

- Prev
- Next
- Nelems
- Index
- Data
- Head
- Tail
- null

Review Quiz

A typical **Node** in a (Singly) LinkedList class will have which fields?
(choose all that apply)

-Note that the exact names of the fields may actually be different, so assume (or some equivalent) for each one.

- Prev
- Next
- Nelems
- Index
- Data
- Head
- Tail
- null

Runtime Analysis!

Motivation: Why analyze algorithms?

- Classify problems and algorithms by difficulty
- **Predict** performance, compare algorithms, tune parameters.
- Better understand and improve implementations and algorithms

Trivia: Analysis of algorithms is one of the subfields of computer science. People dedicate themselves to study the different ways of analyzing and improving algorithms.

Measuring time

- The absolute running time of an algorithm cannot be predicted, since this depends on
    - the programming language used to implement the algorithm,
    - the computer machine the program runs on,
    - other programs running at the same time,
    - the quality of the operating system,

  and many other factors. We need a machine-independent notion of an algorithm's running time.

- The current state-of-the-art in analysis is finding a measure of an algorithm's relative running time, as a function of how many items there are in the input, i.e., the number of symbols required to reasonably encode the input, which we call n.
- We count the number of abstract operations as a function of n.

Example

```
for (int i = 0; i < a.length; i++) {
    System.out.println(a[i]);
}
```

Here n=a.length (provided we know that all of the items in the array have a fixed size, which is often the case).

- 1 initialization of i
- n comparisons of i against a.length
- n increments of i
- n array indexing operations (to compute a[i])
- n invocations of System.out.println

Time Complexity

- Different programming languages will yield different factors when we count their instructions.
- For example, Pascal requires 3 instructions for each array access instead of the 1 instruction Java requires.
- Dropping this factor goes along the lines of ignoring the differences between particular programming languages and compilers and only analyzing the idea of the algorithm itself.
- This filter of "dropping all factors" and of "keeping the largest growing term" as described above is what we call asymptotic behavior.
- So the asymptotic behavior of $f(n) = 4n + 1$ is described by the function $f(n) = n$

- Since we are really measuring growth rates, we usually ignore:
    - all but the “largest” term, and
    - any constant multipliers

Examples:

1. $4n^5 + 3n^3 + 255$
2. $8n\log n + 5n + 2$
3. $5n^2 + 10n + 1$

Big-O: Asymptotic Upper Bounds

A function f is in O(g) whenever there exist constants c and n0 such that for every n>n0, f(n) is bounded above by a constant times g(n).

i.e. **f(n) <= c*g(n) for all n > n0**

Big O of a function gives us 'rate of growth' of the step count function f(n), in terms of a simple function g(n), which is easy to compare.

Examples:

$f(n) = 2n+1 \quad g(n) = n$

$f(n) = n^2 + 2n + 1 \quad g(n) = n^2$

Determining C and n_0

- C is any positive constant greater than 0 ($C > 0$)
  - Once you choose a C, it will not change
- N_0 is any positive **integer** ($n_0 \geq 0$)
  - Think of n_0 as a starting point, and we want to see that our equation:
    $f(n) <= c*g(n)$
    will always hold when n is greater than n_0

Clicker Question!

Let’s see if you get the gist of the previous slide:

What would we consider to be g(n), or the growth rate of f(n) = 14 n log(4) + n

A) n
B) nlogn
C) 14n
D) Nlog4
E) 2n

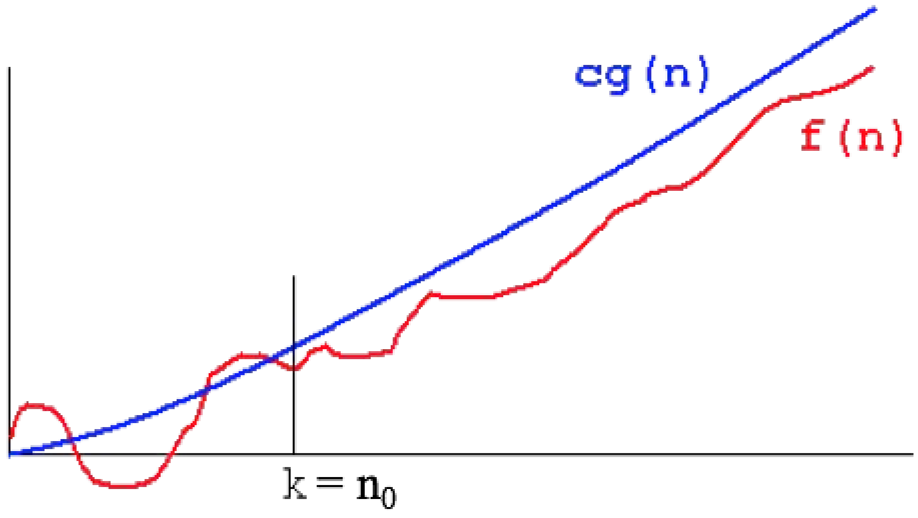
Clicker Question!

Let's see if you get the gist of the previous slide:

What would we consider to be the growth rate of $f(n) = 14\ n \log(4) + n$

A) n
B) nlogn
C) 14n
D) Nlog4
E) 2n

Big O Notation



Big-Omega and Big Theta?

A function f(n) is in Ω(g) whenever there exist constants c and n0 such that for every n>n0, f(n) is bounded below by a constant times g(n)

When the lower and upper bounds are the same, we can use Big-Theta notation

Analysing simple algorithms

1. Given an array “arr” of size n, what is the runtime of getting the 5th value of arr?
   arr[5]

2. Given the same array “arr”, what is the runtime of finding the value 5 in arr?
   if (arr[i] == 5)

3. Given a DoublyLinkedList with head and tail, and with nelems=n, what is the runtime complexity of add(data)?

Analysing simple algorithms

1. Given an array “arr” of size n, what is the runtime of getting the 5th value of arr?
   Arr[5]
   a. O(1)
2. Given the same array “arr”, what is the runtime of finding the value 5 in arr?
   if (arr[i] == 5)
   a. O(n)
3. Given a DoublyLinkedList with head and tail, and with nelems=n, what is the runtime complexity of add(data)?
   a. O(1)

Big-O Hierarchy

- O(1)
- O(log n)
- O(n)
- O(n log n)
- O(n^2) and beyond, n^3, n^4...
- O(2^n) and beyond, 3^n, 4^n...
- There are other “weird” ones but are not usually used

O(logn)

Clicker Question:

- Is $\log_{10}(n)$ a different runtime than $\log_2(n)$?

A. Yes
B. No

O(logn)

Clicker Question:

- Is $\log_{10}(n)$ a different runtime than $\log_2(n)$?

A. Yes
B. No

$$\begin{aligned}\mathrm{Log}_c n &= \log n / \log c \\ &= (1/\log c) * \log n \\ &= c * \log n \\ &= \log n\end{aligned}$$

How can we get O(log n)?

Binary search

- Given a sorted array, find the value n in the array
- Start at the middle, is the value you want smaller or greater?
- Get rid of the unneeded half of the array
- Try again until you find it

The reason this is O(log n) is because it gets rid of half of the possibilities per iteration

Things to memorize

- Array accesses and abstract operations are O(1)
- Traversing an array is O(n)
- Sorting algorithms are O(n log n) (if you use good ones)
- If it contains the word “binary” it’s most likely O(log n)
- HashTables (haven’t been covered) are considered O(1)

Review: How to calculate Big O

- Drop constants
  - $O(n + 3)$ is $O(n)$
  - $O(n / 2)$ is $O(n)$

- Adding is really just picking which one is bigger
  - $O(1) + O(n)$ is $O(n)$
  - $O(n^2) + O(2^n)$ is $O(2^n)$

- Multiplying gets a little complex...

How to multiply Big O's

- In the same sense that x * 1 is x, O(1) * x is x
  - $O(1) * O(n^2) = O(n^2)$

- What about $O(n) * O(n^2)$?
    - Is it $O(n^3)$?

- How would you get O(n log n)?

Analyzing runtime

```
for(int i=0; i<n; i++) {
    //do stuff here
}
```

Runtime? O(n)

Analyzing runtime

```
for(int i=0; i<n; i++) {
	//do stuff here
}
for(int i=0; i<n; i++) {
	//do stuff here
}
```

Runtime: $O(n) + O(n) =$

Analyzing runtime

```
for(int i = 0; i < n; i++)
{
        for(int j = 0; j < n; j++)
        {
                // do more stuff here
        }
}
```

Runtime?

n + n + n + ...n = n*n = $O(n^2)$

Analyzing runtime

```
for(int i = 1; i < =n; i++)
{
        for(int j = 0; j <i; j++)
        {
                // do more stuff here
        }
}
```

Runtime?