CSCI 136:
Data Structures
and
Advanced Programming

Lecture 11

Lists

Instructor: Dan Barowy

Williams

Topics

- Mathematical induction
  Vectors—why `add` is “always O(1)”
- Linked lists

Your to-dos

1. Read **before Wed**: Bailey, Ch 9.4–9.5.
2. Lab 3, **due Tuesday 10/4 by 10pm**.
3. Quiz 4, **due Saturday by noon**.

Announcements

- Colloquium: **What I Did Last Summer (Industry)**, 2:35pm in Wege Auditorium with **cookies**.

Quiz 3

Which of the following completions of the expression calculating midIndex results in the recursive function correctly calculating the smallest element of the array?

```
(a) int midIndex = (startIndex + endIndex) / 2;
(b) int midIndex = (endIndex - startIndex) / 2;
(c) int midIndex = endIndex / 2;
(d) int midIndex = (endIndex + 1) / 2;
```

c	2 respondents	3 %
d		0 %
a	54 respondents	93 %
b	2 respondents	3 %

Well done!

Quiz 3

Assuming this Java method is completed correctly, what is the Big-O running time of this algorithm assuming the length of the array is n?

```
public static int findSmallest(int [] array, int startIndex, int endIndex)
{
    if(startIndex == endIndex){
        return array[startIndex];
    }
    else{
        int midIndex = ________________________;    // Select code to complete

        int firstHalf = findSmallest(array, startIndex, midIndex);
        int secondHalf = findSmallest(array, 1 + midIndex, endIndex);

        return Math.min(firstHalf, secondHalf);
    }
}
```

O(1)		0 %
O(n^2)	18 respondents	31 %
O(log n)	14 respondents	24 %
O(n)	26 respondents	45 %

Let's discuss this problem…

Quiz 3

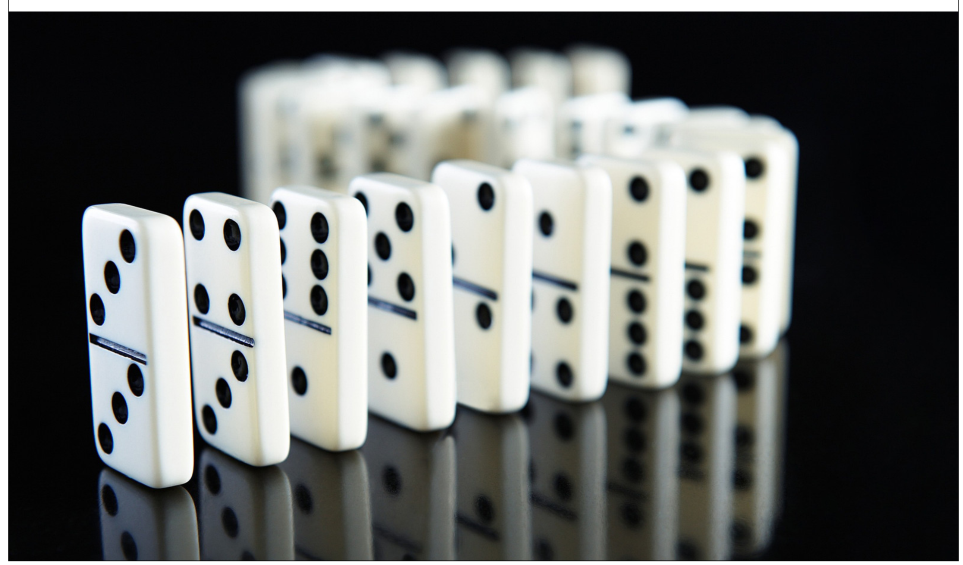
Assuming this Java method is completed correctly, what is the Big-O running time of this algorithm assuming the length of the array is n?

```
public static int findSmallest(int [] array, int startIndex, int endIndex)
{
C1    if(startIndex == endIndex){
          return array[startIndex];
      }
      else{
C6  C2    int midIndex = ________________________;    // Select code to complete

    C4    int firstHalf = findSmallest(array, startIndex, midIndex);
    C5    int secondHalf = findSmallest(array, 1 + midIndex, endIndex);

    C3    return Math.min(firstHalf, secondHalf);
      }
}
```

```
(a) int midIndex = (startIndex + endIndex) / 2;
(b) int midIndex = (endIndex - startIndex) / 2;
(c) int midIndex = endIndex / 2;
(d) int midIndex = (endIndex + 1) / 2;
```

$$T(n) = (c_1 \times n) + (c_6 \times (\log_2(n) - 1)) = O(n)$$

Mathematical Induction



Principle of Mathematical Induction

Hypothesis: **P(n)** is **true** for all integers **n ≥ a,**

1. Base case: **P(a)** is **true**.

2. Inductive step:

   For all integers **k ≥ a**, **if** **P(k)** is **true** **then** **P(k+1)** is **true**.

To be clear:

If you want to prove that **P(n)** is **true** for all integers **n ≥ a,**

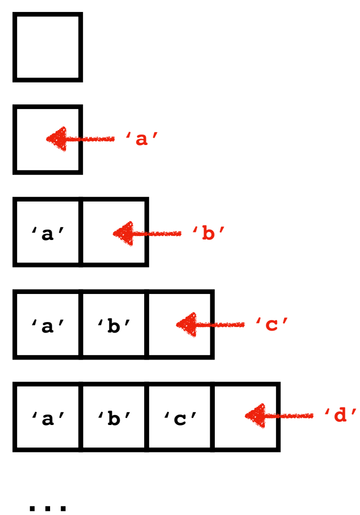
1. You must first prove that **P(a)** is **true**.

2. Then **suppose** **P(k)** is **true** and prove that **P(k+1)** is **true**.

Expanding vectors: why double?

Why is the **array doubling** strategy for Vector **better** than expanding the array **one element at a time**?

One-at-a-time expansion



Initial array.

Insert element.

New array; **copy** previous; **insert** element.

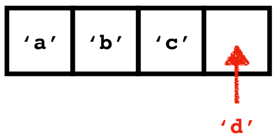
New array; **copy** previous; **insert** element.

New array; **copy** previous; **insert** element.

. . .

Insertion into an array

How much does **array insertion** cost?



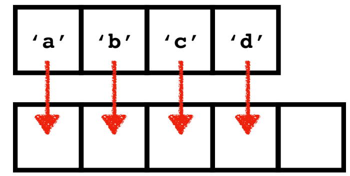
It costs **O(1)**.

In fact, lookup and insertion both cost **O(1)**.

Tradeoff: arrays are fixed size.

Copying an array

How much does an **array copy** cost?



It costs **O(1)** × **m**, where **m** is the size of the original array.

≈ **O(m)**

How many copies?

of copies for one-at-a-time expansion:

	1	**+**	**2**	**+**	**3**	**+**	**…**	**+(n-1)**
`add()`	2nd elem.		3rd elem.		4th elem.			nth elem.

Recall theorem: $1 + 2 + 3 + \ldots + k = k(k+1)/2$

Sub n-1 for k: $(n-1)((n-1)+1)/2 = n(n-1)/2$

$= (n^2-n)/2$

One-at-a-time expansion costs ≈ **O(n²)**

How many copies?

of copies for doubling expansion:

	1	**+**	**2**	**+**	**4**	**+**	**…**	**+(n/2)**
`add()`	up to 2nd elem.		up to 4th elem.		up to 8th elem.			up to nth elem.

Neat theorem: $1 + 2 + 4 + \ldots + 2^{k-1} = 2^k - 1$

Suppose $n = 2^k$.

Then $1 + \ldots + n/2 = 1 + \ldots + 2^k/2$

$= 1 + \ldots + 2^{k-1} = 2^k - 1 = n - 1$

Doubling expansion costs ≈ **O(n)**

Which is faster?

One-at-a-time expansion costs ≈ **$O(n^2)$**

Doubling expansion costs ≈ **$O(n)$**

Doubling is Vin Diesel-approved.

A good practice induction problem

Prove: n cents can be obtained by using only 3-cent and 8-cent coins, for all $n \geq 15$.

Linked Lists

Linked List

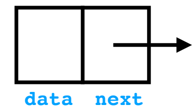
A **linked list** is a recursive data structure. A linked list is composed of simple pieces called **list nodes**. A list node contains **data** (of generic type **T**) and a **reference** (a “link”) to either **another list node** or **null**.

Linked List

$\varnothing$

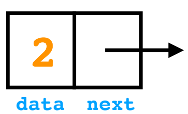
The empty list is defined as **null**.

Linked List



Every other list has at least one list node.

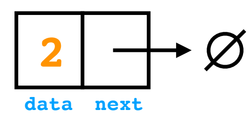
Linked List



A list node stores data of type **`T`**.

Here, **`T`** is **`Integer`**.

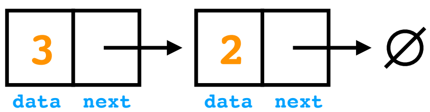
Linked List



The **`next`** field stores a reference (“link”) to the next node.

If the node is the last node, the next node is **null**.

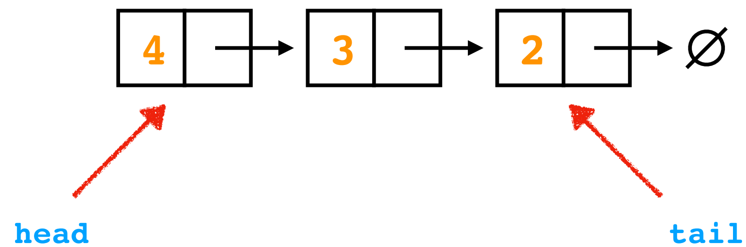
Linked List



If the next node is not **null**, it is, recursively, a list node.

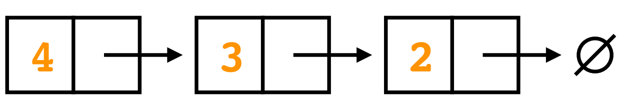
The last node in the list must always point to **null**.

Linked List



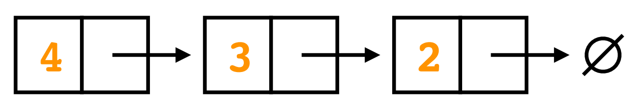
A list has parts.

Linked List



When we add data to a list, we always `append` to the `head`.

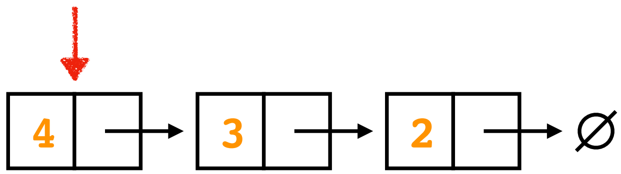
Linked List



To find a value, we must always traverse the list starting from the `head`.

E.g., looking for **2**…

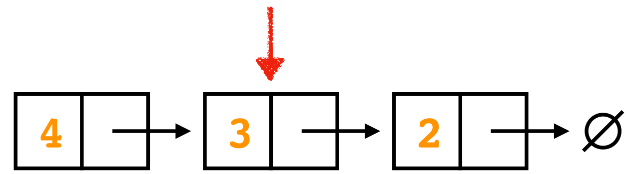
Linked List



To find a value, we must always traverse the list starting from the `head`.

E.g., looking for **2**…

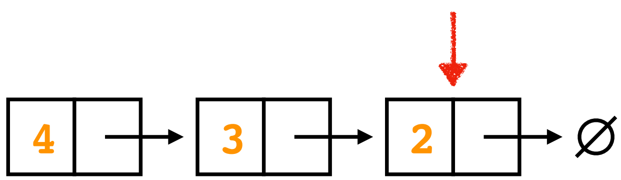
Linked List



To find a value, we must always traverse the list starting from the `head`.

E.g., looking for **2**…

Linked List



To find a value, we must always traverse the list starting from the `head`.

E.g., looking for **2**…

Recap & Next Class

Today:

- Why Vector should double
- Lists

Next class:

- ADTs
- More lists