

CPSC 221

Basic Algorithms and Data Structures

May 11, 2015

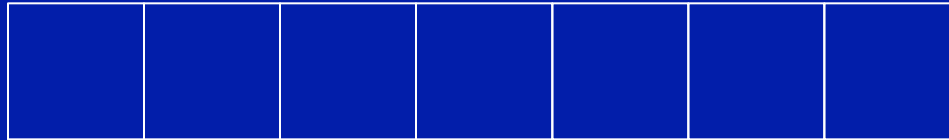
Who I Am

Your instructor is



Kurt Eiselt
eiselt@cs.ubc.ca
IKBLC 366A
office hours soon to be determined

We'll talk about TAs on Wednesday, after I've met them for the first time(!).



What's this?

B	F	G	D	A	C	E
0	1	2	3	4	5	6

What's this? Maybe it's more obvious now.

B	F	G	D	A	C	E
0	1	2	3	4	5	6

What's this? Maybe it's more obvious now.

It's an abstracted, logical representation of an array. You know the array from Java in CPSC 210. You probably used a more flexible version called the arraylist.

B	F	G	D	A	C	E
0	1	2	3	4	5	6

Given this array and a character, how do you determine if the character is in the array?

B	F	G	D	A	C	E
0	1	2	3	4	5	6

Given this array and a character, how do you determine if the character is in the array?

Start at index 0, look at the value there, if it's what I'm looking for then success, else increment the index and do it again. If I run out of array, then failure.

B	F	G	D	A	C	E
0	1	2	3	4	5	6

Given this array and a character, how do you determine if the character is in the array? Here's a Java-like approach:

```
for (i = 0; i < array.length; i++)  
{  
    if array[i] == target_char  
        return true;  
}  
return false;
```


B	F	G	D	A	C	E
0	1	2	3	4	5	6

How many times does the comparison happen when looking for the letter “B”?

```
for (i = 0; i < array.length; i++)  
{  
    if array[i] == target_char  
        return true;  
}  
return false;
```

B	F	G	D	A	C	E
0	1	2	3	4	5	6

How many times does the comparison happen when looking for the letter “B”? How about “E”?

```
for (i = 0; i < array.length; i++)  
{  
    if array[i] == target_char  
        return true;  
}  
return false;
```

B	F	G	D	A	C	E
0	1	2	3	4	5	6

How many times does the comparison happen when looking for the letter “B”? How about “E”? How about “Q”?

```
for (i = 0; i < array.length; i++)  
{  
    if array[i] == target_char  
        return true;  
}  
return false;
```

B	F	G	D	A	C	E	...	R	L	X
0	1	2	3	4	5	6		12	13	14

What if the array is bigger?

```

for (i = 0; i < array.length; i++)
{
    if array[i] == target_char
        return true;
}
return false;

```

B	F	G	D	A	C	E	...	R	L	X
0	1	2	3	4	5	6		12	13	14

What if the array is bigger? How many comparisons to find “B” now?

```
for (i = 0; i < array.length; i++)
{
    if array[i] == target_char
        return true;
}
return false;
```

B	F	G	D	A	C	E	...	R	L	X
0	1	2	3	4	5	6		12	13	14

What if the array is bigger? How many comparisons to find “B” now? How many comparisons to find “X”?

```
for (i = 0; i < array.length; i++)
{
    if array[i] == target_char
        return true;
}
return false;
```

B	F	G	D	A	C	E
0	1	2	3	4	5	6

...

R	L	X
12	13	14

Can we generalize over these two examples? In both arrays, the best case number of comparisons to find what we're looking for is...

B	F	G	D	A	C	E	...	R	L	X
0	1	2	3	4	5	6		12	13	14

Can we generalize over these two examples? In both arrays, the best case number of comparisons to find what we're looking for is 1.

In both arrays, the worst case number of comparisons would be...

B	F	G	D	A	C	E	...	R	L	X
0	1	2	3	4	5	6		12	13	14

Can we generalize over these two examples? In both arrays, the best case number of comparisons to find what we're looking for is 1.

In both arrays, the worst case number of comparisons would be n , where n is the length of the array.

B	F	G	D	A	C	E	...	R	L	X
0	1	2	3	4	5	6		12	13	14

Can we generalize over these two examples? In both arrays, the best case number of comparisons to find what we're looking for is 1.

In both arrays, the worst case number of comparisons would be n , where n is the length of the array.

What do you think the average number of comparisons would be?

B	F	G	D	A	C	E	...	R	L	X
0	1	2	3	4	5	6		12	13	14

Can we generalize over these two examples? In both arrays, the best case number of comparisons to find what we're looking for is 1.

In both arrays, the worst case number of comparisons would be n , where n is the length of the array.

What do you think the average number of comparisons would be? Something like $n/2$?

'(B F G D A C E)

Now let's look at a different linear collection of items. What is this?

'(B F G D A C E)

Now let's look at a different linear collection of items. What is this? Yes, it's a list. All you CPSC 110 grads should be able to write a Racket function to find a given symbol in this list...

'(B F G D A C E)

Now let's look at a different linear collection of items. What is this? Yes, it's a list. All you CPSC 110 grads should be able to write a Racket function to find a given symbol in this list...

```
(define (find char charlist)
  (cond [(empty? charlist) false]
        [(equal? (first charlist) char) true]
        [else (find char (rest charlist))]))
```

No data definitions? No check expects? Bad! Bad! Bad!

'(B F G D A C E)

How many comparisons now will be required to find B?

```
(define (find char charlist)
  (cond [(empty? charlist) false]
        [(equal? (first charlist) char) true]
        [else (find char (rest charlist))]))
```

'(B F G D A C E)

How many comparisons now will be required to find B?
How many comparisons to find E?

```
(define (find char charlist)
  (cond [(empty? charlist) false]
        [(equal? (first charlist) char) true]
        [else (find char (rest charlist))]))
```


'(B F G D A C E)

How many comparisons now will be required to find B?

How many comparisons to find E?

What happens when the list is bigger? Generalize again...

```
(define (find char charlist)
  (cond [(empty? charlist) false]
        [(equal? (first charlist) char) true]
        [else (find char (rest charlist))]))
```

B	F	G	D	A	C	E
0	1	2	3	4	5	6

'(B F G D A C E)

Two different data structures, with associated algorithms for finding something in the structure. Which one is better?

B	F	G	D	A	C	E
0	1	2	3	4	5	6

'(B F G D A C E)

Two different data structures, with associated algorithms for finding something in the structure. Which one is better?

If we accept that number of comparisons is an approximation of time, then the time to find something in either of these structures is the same: proportional to the size of the structure itself.

B	F	G	D	A	C	E
0	1	2	3	4	5	6

'(B F G D A C E)

Can we make the search substantially more efficient?

I need two volunteers, both of whom know how to use an old-fashioned phone book.

Contestant #1

Contestant #1

In the phone book, find the phone number for
Michael Saville on Arbutus Street

Contestant #1

In the phone book, find the phone number for
Michael Saville on Arbutus Street
604-733-2627

Contestant #1

In the phone book, find the phone number for
Michael Saville on Arbutus Street
604-733-2627

Contestant #2

Contestant #1

In the phone book, find the phone number for
Michael Saville on Arbutus Street
604-733-2627

Contestant #2

In the phone book, find the name of the person whose
phone number is 604-733-0443

Contestant #1

In the phone book, find the phone number for
Michael Saville on Arbutus Street
604-733-2627

Contestant #2

In the phone book, find the name of the person whose
phone number is 604-733-0443
George Bush (not that George Bush)

What feature of the phone book made it easy for the first volunteer to find the phone number?

What feature of the phone book made it easy for the first volunteer to find the phone number? The phone book is sorted, by name, not by telephone number.


The phone book is essentially an array whose contents are sorted. An overwhelmingly simple and useless version could be modeled like this...

A	B	C	D	E	F	G
0	1	2	3	4	5	6

Describe a search algorithm that could take advantage of this sorting. Where would you start the search?

A	B	C	D	E	F	G
0	1	2	3	4	5	6

How many comparisons to find E?



A	B	C	D	E	F	G
0	1	2	3	4	5	6

How many comparisons to find E?
Is it here? No.



How many comparisons to find E?

Is it here? No.

Is it here? No.



How many comparisons to find E?

Is it here? No.

Is it here? No.

Is it here? Yes.



How many comparisons to find E?

Is it here? No.

Is it here? No.

Is it here? Yes.

Best case: 1 comparison

Worst case: 3 comparisons

A	B	C	D	E	F	G	L	M	N	P	Q	R	W	X
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

What if the sorted array is bigger?
How many comparisons to find L?

A	B	C	D	E	F	G	L	M	N	P	Q	R	W	X
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

What if the sorted array is bigger?

How many comparisons to find L? 1 comparison - best

How many comparisons to find E?

A	B	C	D	E	F	G	L	M	N	P	Q	R	W	X
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

What if the sorted array is bigger?

How many comparisons to find L? 1 comparison - best

How many comparisons to find E? 4 comparisons - worst

A	B	C	D	E	F	G	L	M	N	P	Q	R	W	X
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

Can you figure out the relationship between the size of the array and the worst case search performance?

worst case comparisons = 3

array size = 7

worst case comparisons = 4

array size = 15

worst case comparisons = ?

array size = 31

A	B	C	D	E	F	G	L	M	N	P	Q	R	W	X
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

Can you figure out the relationship between the size of the array and the worst case search performance?

worst case comparisons = 3

array size = 7

worst case comparisons = 4

array size = 15

worst case comparisons = 5

array size = 31

A	B	C	D	E	F	G	L	M	N	P	Q	R	W	X
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

Can you figure out the relationship between the size of the array and the worst case search performance?

worst case comparisons = 3

array size = 7

worst case comparisons = 4

array size = 15

worst case comparisons = 5

array size = 31

The Time to do the search (which correlates to the number of comparisons) on the sorted array of size n is some function f of n . What's the function f ?

A	B	C	D	E	F	G	L	M	N	P	Q	R	W	X
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

Can you figure out the relationship between the size of the array and the worst case search performance?

worst case comparisons = 3

array size = 7

worst case comparisons = 4

array size = 15

worst case comparisons = 5

array size = 31

$$T(n) = \text{some } f(n)$$

A	B	C	D	E	F	G	L	M	N	P	Q	R	W	X
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

Can you figure out the relationship between the size of the array and the worst case search performance?

worst case comparisons = 3

worst case comparisons = 4

worst case comparisons = 5

array size = 7

array size = 15

array size = 31

$$T(n) = \text{some } f(n)$$

$$f(n) = \log_2(n+1)$$

A	B	C	D	E	F	G	L	M	N	P	Q	R	W	X
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

$$T(n) = f(n) = \log_2(n+1)$$

That's pretty good search performance. Each time we double the size of the array being searched, we only incur at most one extra comparison!

A	B	C	D	E	F	G	L	M	N	P	Q	R	W	X
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

$$T(n) = f(n) = \log_2(n+1)$$

That's pretty good search performance. Each time we double the size of the array being searched, we only incur at most one extra comparison!

(And if you're not comfortable with logarithms, start brushing up. We'll be using them in this class.

'(A B C D E F G)

Do we gain the same benefit by sorting the elements of this list?

'(A B C D E F G)

Do we gain the same benefit by sorting the elements of this list? What would be the algorithm for performing this “telephone book” style of search on this list? Feel free to knock out some Racket code...

'(A B C D E F G)

Do we gain the same benefit by sorting the elements of this list? What would be the algorithm for performing this “telephone book” style of search on this list? Feel free to knock out some Racket code...

Let's pose the question a different way: is there a simple one-step way to get at the middle element of this list?

An answer requires an understanding of how that list data structure really works. Here's a more informative picture...



This is a singly linked list (sometimes called a one-way linked list). It's the kind of list that Racket provides.

Do you see a problem now with getting to the middle of the list in one simple step? With the sorted array, finding the midpoint of the unsearched remainder of the list was simple arithmetic. Here it requires traversing half the links, and if we get to the middle how do we go left? And wouldn't we have already searched to the left anyway? This isn't promising.



We could construct our own doubly linked list that would allow us to traverse the structure in both directions. How could you do that in Racket?



We could construct our own doubly linked list that would allow us to traverse the structure in both directions. How could you do that in Racket? Remember the `struct`?

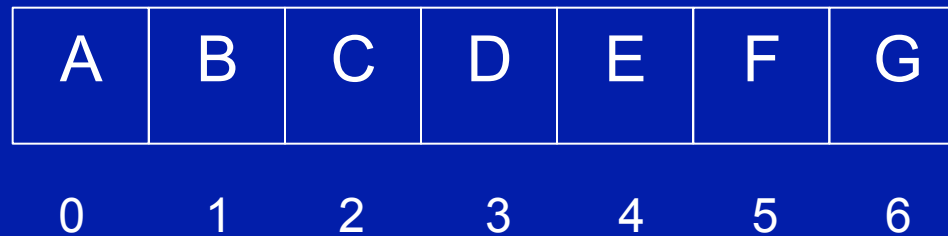
But we still have the issue of no simple way to get to the middle of the remaining unsearched list, so this isn't promising either.



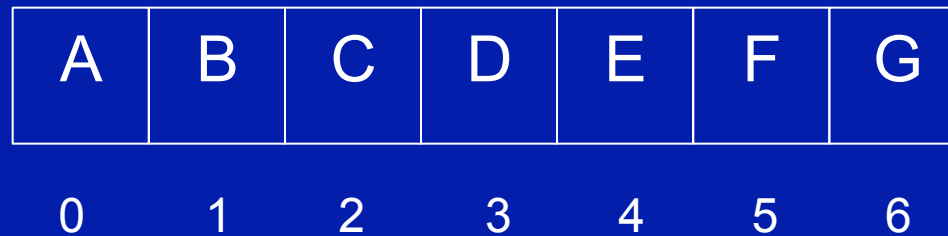
We could construct our own doubly linked list that would allow us to traverse the structure in both directions. How could you do that in Racket? Remember the `struct`?

But we still have the issue of no simple way to get to the middle of the remaining unsearched list, so this isn't promising either.

Is there some other linked structure that would facilitate the “telephone book” search? Yes. You’ve seen it in CPSC 110, and we’ll see it in CPSC 221 in the days ahead. Patience grasshopper.



Let's go back to comparing the sorted array to the sorted singly linked list. Now which one is better?



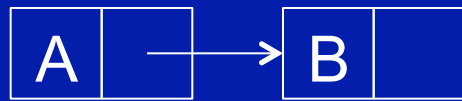
Let's go back to comparing the sorted array to the sorted singly linked list. Now which one is better? Why?

A	B	D	E	F	G	H
0	1	2	3	4	5	6

If you were thinking that arrays are the way to go here, think again. What if you want to insert a new element, C, in this sorted array? What's the algorithm for insertion? What if the array only has room for 7 elements?



What if you want to insert a new element, C, in this sorted list? How do you do the insertion in this case?



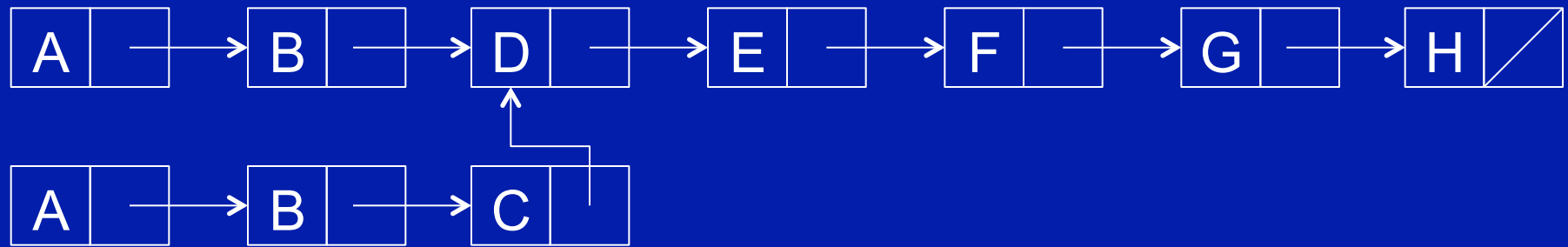
If you're in CPSC 110 and your list is not mutable, then just using `first`, `rest`, and `cons` you could make a new copy of the list just up to the insertion point,



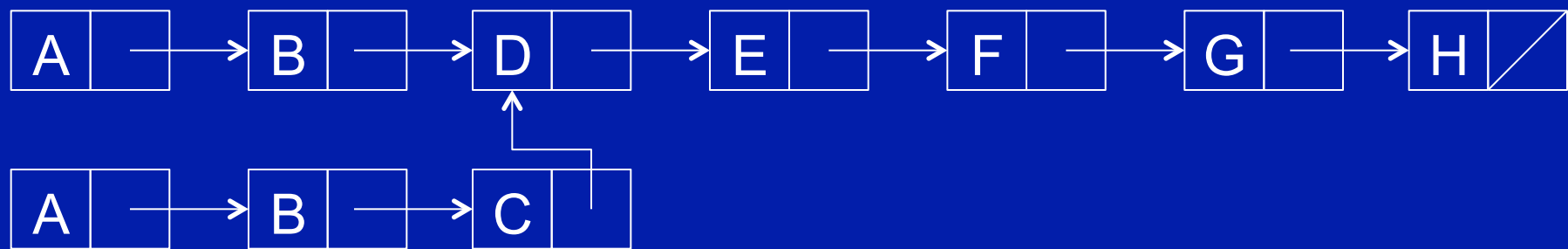
If you're in CPSC 110 and your list is not mutable, then just using first, rest, and cons you could make a new copy of the list just up to the insertion point, create a cons cell to hold C,



If you're in CPSC 110 and your list is not mutable, then just using first, rest, and cons you could make a new copy of the list just up to the insertion point, create a cons cell to hold C, cons the new list copy onto the inserted cons cell,

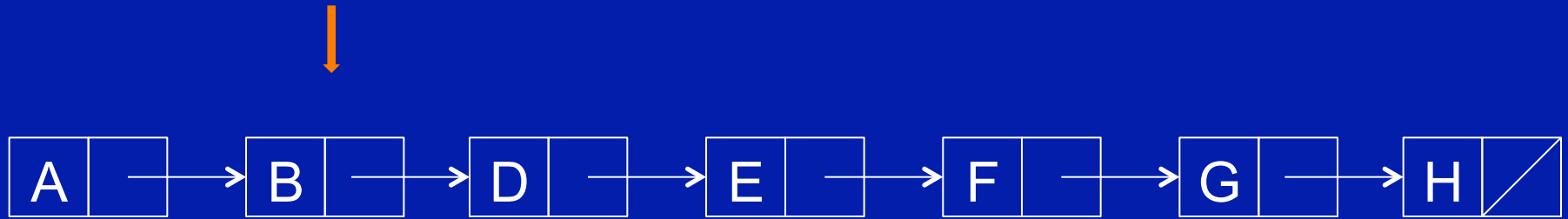


If you're in CPSC 110 and your list is not mutable, then just using `first`, `rest`, and `cons` you could make a new copy of the list just up to the insertion point, create a cons cell to hold C, cons the new list copy onto the inserted cons cell, and finally cons C onto the old list past the insertion point.

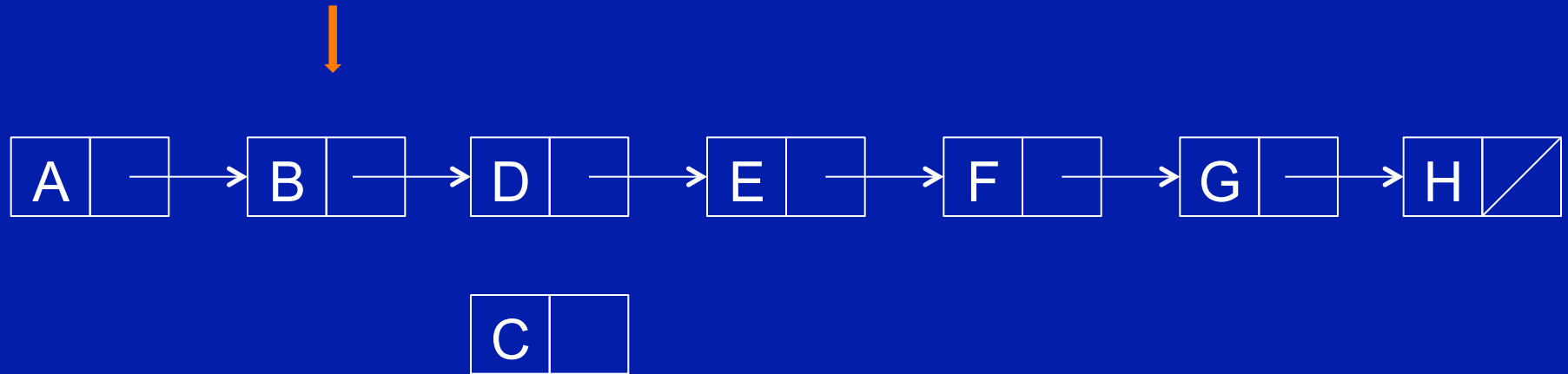


If you're in CPSC 110 and your list is not mutable, then just using `first`, `rest`, and `cons` you could make a new copy of the list just up to the insertion point, create a cons cell to hold C, cons the new list copy onto the inserted cons cell, and finally cons C onto the old list past the insertion point.

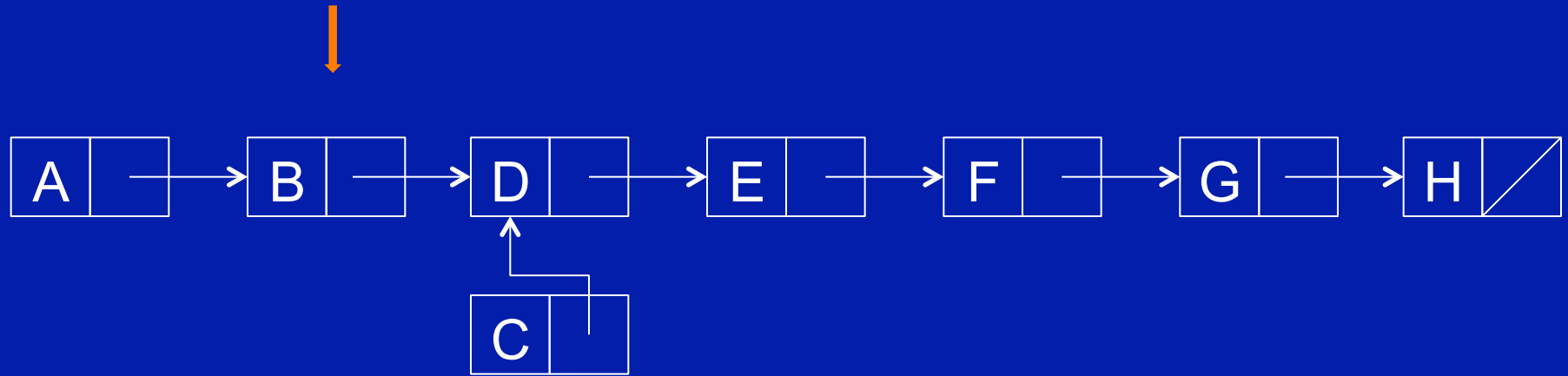
That's the 110 mostly functional programming approach. Most of what we do in this class will assume we don't have those constraints. We'll assume mutable data structures...



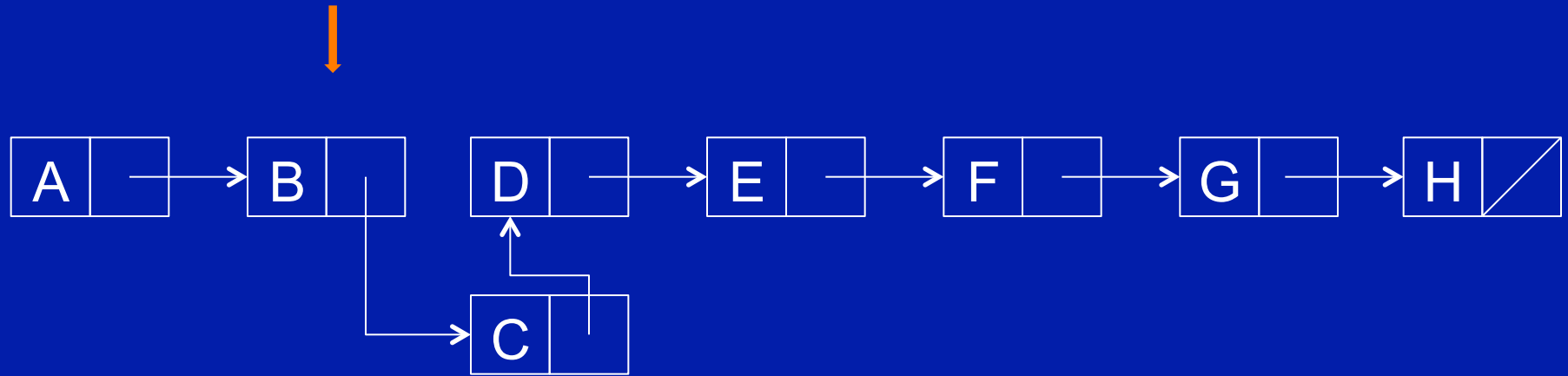
Start at the front of the list and traverse to the insertion point,



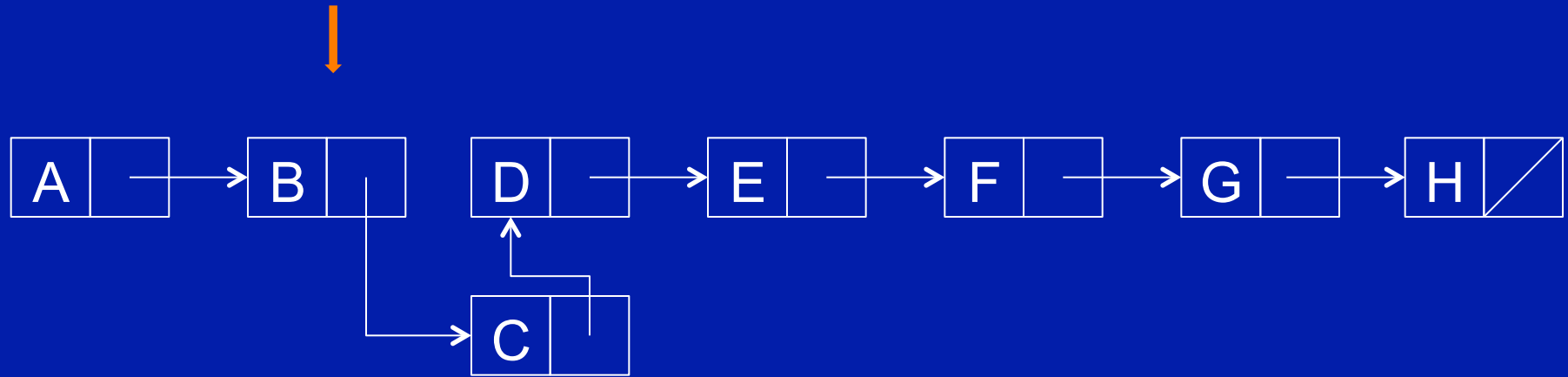
Start at the front of the list and traverse to the insertion point, create the new list element,



Start at the front of the list and traverse to the insertion point, create the new list element, copy the link from B into the link from C,



Start at the front of the list and traverse to the insertion point, create the new list element, copy the link from B into the link from C, and finally modify the link from B so that it points to C.



Start at the front of the list and traverse to the insertion point, create the new list element, copy the link from B into the link from C, and finally modify the link from B so that it points to C.

If you're doing lots of insertions, which data structure would you rather be using now?

The question of “which is better?” is an intentional red herring. Which is “better” depends on the context: how will you be using the data structure and its associated algorithms?

Lots of searching/few insertions? The array seems better.

Many insertions/little searching? The list seems better.

Realistically, if you want both (and you probably do, plus deletions) there are better choices. More about these in the future.

Questions like “which data structure is better?” or “which algorithm is better?” may not have absolute answers.

“It depends...” may be the beginning of many such analyses.

The point of the whole discussion we just had is just to give you a sense for what this class is all about.

Let's take a break.

When we come back, I have a little survey for you to complete, and then we'll talk about course administration details.

Survey

Please answer the questions on the survey as best you can.

If you don't understand what the question is asking about, please don't answer the question.

Do not identify yourself on the survey: no name, no student ID number...just your answers.

Return the survey to me when you've completed it.

What Is This Course About?

Calendar description: Design and analysis of basic algorithms and data structures; algorithm analysis methods, searching and sorting algorithms, basic data structures, graphs and concurrency.

What Is This Course About?

- Some classic algorithms
- Some classic data structures
- Analysis tools and techniques for the above

What's an algorithm?

- Some classic **algorithms**
- Some classic data structures
- Analysis tools and techniques for the above
- Algorithm (typical definition): a high-level, language-independent description of a step-by-step process for solving a problem

What's an algorithm?

- Some classic **algorithms**
- Some classic data structures
- Analysis tools and techniques for the above
- Algorithm (street definition): a smarter way to solve the problem

What's a data structure?

- Some classic algorithms
- Some classic **data structures**
- Analysis tools and techniques for the above
- Data structure (street definition): how to organize your data to get the results you want, along with the supporting algorithms

Why study classic examples?

- Some classic algorithms
- Some classic data structures
- Analysis tools and techniques for the above
- They are useful!
 - like prepackaged intelligence in a can
 - don't have to work hard to come up with your own solution

Why study classic examples?

- Some classic algorithms
- Some classic data structures
- Analysis tools and techniques for the above
- They let you abstract away details!
 - these are “power tools” for programming
 - let you focus on solving bigger problems, ignore details

Why study classic examples?

- Some classic algorithms
- Some classic data structures
- Analysis tools and techniques for the above
- You learn general solution ideas
 - this will help you solve new, unexpected problems
 - the masters in any field study the classic examples from their field

Why the theory and math?

- Some classic algorithms
- Some classic data structures
- Analysis tools and techniques for the above
- This gives you the tools to determine:
 - what's good or bad
 - what trade-offs are being madeand explain clearly why

What is this course about?

- Some classic algorithms
- Some classic data structures
- Analysis tools and techniques for the above
- Overall, this course is a big step that distinguishes you from being just some schlub who learned some coding

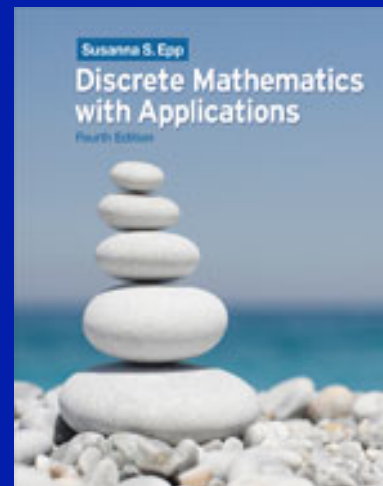
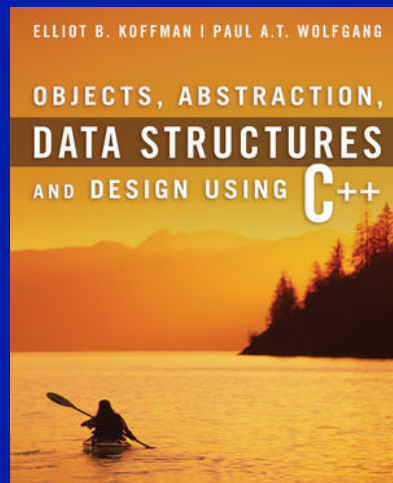
Course Goals

- Become familiar with some of the fundamental data structures and algorithms in computer science
- Improve ability to solve problems abstractly
 - data structures and algorithms are the building blocks
- Improve ability to analyze your algorithms
 - prove correctness
 - gauge, compare, and improve time and space complexity
- Become modestly skilled with C++ and UNIX, *but this is largely on your own!*

Textbooks

Objects, Abstraction, Data Structures and Design Using C++, Elliot B. Koffman and Paul A.T. Wolfgang, Wiley, 2006.

Discrete Mathematics with Applications
(4th edition), Susanna S. Epp, Brooks/Cole, 2011.



Programming Language

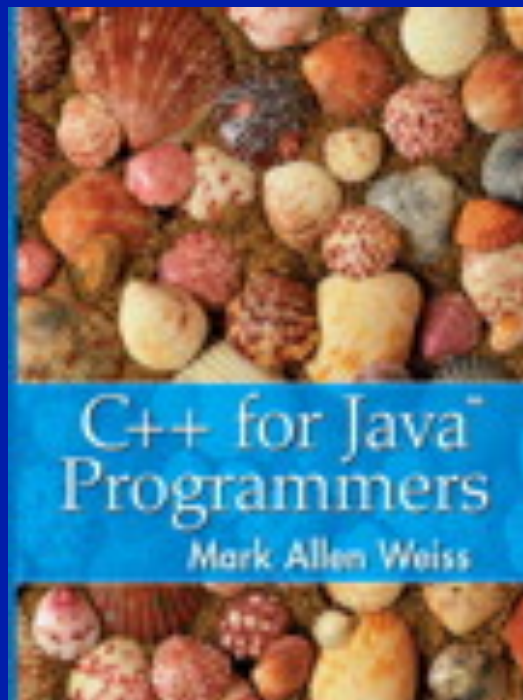
You'll be using C++, but we won't teach it in class. We expect you to pick it up on your own! Sorry.

Read Chapters P and 1 of Objects, Abstraction, Data Structures and Design Using C++ before your first lab.

There's plenty of C++ info on the Internet, in the CS Reading Room, and in the UBC Library. We'll try to put pointers to some C++ resources on UBC Connect.

Programming Language

Your best resource for C++ help may be your teaching assistants. They all know more than I do. Here's where I'm learning C++:



Grades

Tentative schedule

Midterm exam:	Wednesday, June 3 (in class)
Final exam period:	June 22 - 26

Tentative grade calculation (subject to change)

Labs	10%
Assignments	10%
Midterm exam	30%
Final exam	50%

More Grades

Please note that in order to pass the course you must:

- *obtain an overall grade of at least 50%*
- *obtain a grade of at least 50% on the final exam*
- *obtain an overall grade of at least 50% on the combined lab and assignment grades*

If you fail to satisfy any of the above criteria, a grade no greater than 45% will be assigned in the course. The instructor reserves the right to modify this grading scheme as necessary throughout the term, with reasonable advance notice.

Note that you will not pass the course if you are not enrolled in a lab section.

Labs and Assignments

You have two labs per week, beginning on Thursday and Friday of this week, then Tuesday through Friday of the weeks thereafter.*

- *You must be enrolled in a lab section to pass the class.*
- *You must attend the lab in which you are enrolled.*
- *You will not be permitted to make-up missed labs by attending a lab section in which you are not enrolled.*

You'll also have some number (yet to be determined) of homework assignments.

* Except we might take a lab break somewhere in there.

Collaboration

I support collaborative learning. We'll talk about the collaboration guidelines for this class in the near future.

Lectures

Typically we'll use the entire 2.5 hours every Monday, Wednesday, and Friday, with a ten minute break somewhere in the middle. **That break is for me as well as you. Do not expect that I will spend the break answering questions.**

Course Communication

Some supplementary course materials will soon be available through UBC Connect. We'll also use UBC Connect to make important announcements. You should check in with UBC Connect daily.

You can find a tentative course schedule online at <http://www.techweenie.org/cpsc-221.html>

How intense is the summer version?

3 lectures per week x 6 weeks = 18 lectures

midterm exam: subtract 1 lecture = 17 lectures

term ends on Thurs: subtract 1 lecture = 16 lectures

holiday on May 18: subtract 1 lecture = 15 lectures

In the winter, this class runs 13 weeks. So in the summer, we have 15 lectures to cover 13 weeks of material. One lecture in the summer is almost a week's worth of material from the winter session. Do not fall behind.

Oh, did I mention that your midterm exam is 3 weeks from this Wednesday?