CS 367 - Introduction to Data Structures
Week 4, 2017

Midterm Exam 1

- Friday July 14th, 11:00 am to 1:00 pm, CS1240
- UW ID required
- See posted exam information
- Make up exam scheduled

Program 1:

- Due 10 pm Tuesday, July 11th, TIME IS RUNNING OUT!
- Make sure your code complies on the CSL machine.
- Both partners need to turn in same copy of code excepting with different file headers.

Homework 1: Grades posted to Learnatuw. Send me an email if you find discrepancy!

Last Week: StackADT, QueueADT, array vs chain of nodes implementations, circular array data structure

This Week:

Tree terms
Read: *Tree Intro, Priority Queue, Recursion*

- concept
- operations
- implementation options

Java's Comparable Interface
Heap Data Structure

- insert
- removeMax

Java's Stack, Queue, PriorityQueue
Call Stack Tracing
Recursion

- recursion vs. iteration
- constructing recursive code
- practice writing recursive code

Next Week

Read: finish *Recursion*, *Search*
Read: *Trees*
Execution tree tracing
Searching
Categorizing ADTs Part 1
General Trees

- implementing
- determining tree height

Java's `Comparable` Interface

Implementing a Priority Queue ADT using a Heap

Heap

min heap
max heap

Shape Constraint

Ordering Constraint (max)

Implementing Heaps

Max Heap Example:

	56	42	37	38	14	12	26	29	16	8

→ Draw the corresponding binary tree:

Inserting into a Max Heap

Algorithm

Given the following max heap:

	64	52	35	46	17	15	34	12	23	14		

→ Show the heap after inserting 36:

→ Show the heap after inserting 57:

Complexity

Inserting into a Max Heap (cont.)

PriorityQueue Class Instance Variables:

```
private Comparable[] queue;
private int numItems;
```

Pseudo-code

```
public void insert(Comparable item) {
```

Removing from a Max Heap

Algorithm

Given the following max heap:

	64	52	57	46	36	35	34	12	23	14	17	15

→ What will the heap look like after doing a removeMax?

→ What will the heap look like after doing another removeMax?

Complexity

Call Stack Tracing - Displaying a Singly-Linked Chain of Nodes

Method Call:

```
print(head);
```

Iterative Implementation:

```
void print (Listnode<String> curr) {
   while (curr != null) {
      System.out.println(curr.getData());
      curr = curr.getNext();
   }
}
```

Recursive Implementation:

```
void print(Listnode<String> curr) {
   if (curr == null) return;
   System.out.println(curr.getData());
   print(curr.getNext());
}
```

How do these work?

Recursion vs. Iteration

Recursion is like iteration:

Iteration Recursion

Recursion is NOT like iteration:

- Each loop iteration

- A loop with a bad stopping condition

Rules for Recursion

1.

2.

Recursion

What is it?

Why use it?

→ How would you modify the print method to display a singly-linked chain of nodes in reverse order?

Factorials: n!

Consider the factorial of n (assume n >=0):

n! = n x (n-1) x (n-2) x (n-3) x . . . x 2 x 1
6! = 6 x 5 x 4 x 3 x 2 x 1 = 720

Method Call:

```
factorial(6);
```

Iterative Implementation:

```
int factorial(int n) {
   int result = 1;
   for (; n > 1; n--)
      result = result * n;
   return result;
}
```

Recursive Definition:

→ Complete the Recursive Implementation:

```
int factorial(int n) {
```

Constructing Recursive Code

→ Write a recursive method that computes n^m
that is, it computes `double` n raised to an `int` power m?

recursive definition:

recursive implementation:

Key Questions:

1.

2.

3.

4.

Practice – ListADT

→ Write a recursive method that displays the values in a (non-null) list of strings.

1.

2.

3.

4.

```
void display(ListADT<String> list) {
```

Making a Reversed Copy of a Chain of Nodes

Method Use:

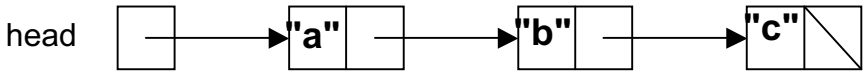
```
Listnode<String> reversed = reverseCopy(head, null);
```

Method Implementation:

```
Listnode<String> reverseCopy
        (Listnode<String> curr, Listnode<String> rev) {

   if (curr == null) return rev;
   rev = new Listnode<String>(curr.getData(), rev);
   return reverseCopy(curr.getNext(), rev);
}
```

How does it work?



Practice – Array

→ **Write a recursive method that counts the number of even values in an (non-null) array filled with integers.**

1.

2.

3.

4.

```
int evenCount(int[] array) {
```

Analyzing Complexity of Recursive Methods

Options:

1.
2.

Steps

1.

2.

3.

4.

Practice – Complexity of Recursive `evenCount`

Problem size N is

1. Equations

2. Table

3. Verify

4. Complexity

Practice – Strings

→ Write a recursive method that determines if a string is a palindrome.

Examples:

- eye
- mom
- radar
- racecar
- Rise to vote, sir!
- Never odd or even!
- A nut for a jar of tuna.
- Campus Motto: Bottoms up, Mac.
- Ed, I saw Harpo Marx ram Oprah W aside!
- Doc note: I dissent. A fast never prevents a fatness. I diet on cod.

Assumptions: non-null input string, all spaces and punctuation removed, all lower-case

Useful string methods:

- char charAt(int index)
- int length()
- String substring(int begin, int one_past_last)

Practice – Complexity of Recursive `isPalindrome`

Problem size N is

1. Equations

2. Table

3. Verify

4. Complexity

Towers of Hanoi

Algorithm

```
solveTowers(count, src, dest, spare) {
```

Complexity

Problem size N is

1. Equations

2. Table

3 Verify

4. Complexity