CS 367 - Introduction to Data Structures
Week 3, 2017

Homework h1 graded. Email TA (di3@wisc.edu) by Friday, July 7th - 5 pm.
Program 1 due July 11th - 10 pm
Homework h3 posted, complete as soon as possible; Due by Sunday, July 9th - 10 pm. Multiple deadlines approaching, plan ahead!

Last Week

Listnode class, chain of nodes, LinkedList Class
Linked List Variations: header node, tail reference
LinkedListIterator Class

This Week

Read: *Complexity, Stacks and Queues, Tree Intro,* Priority Queue
More Linked List Variations
- double linking
- circular linking

Complexity
- concept, big-O notation
- analyzing algorithms practice
- analyzing Java code
- practice analyzing Java code
- best/worst cases
- significance of scaling
- complexity caveats

Comparing Complexity Analysis of ArrayList vs LinkedList
Shadow Array - improving array resizing
Stack ADT
- concept
- array implementations
- chain of nodes implementations

Queue ADT
- concept
- chain of nodes implementations

Priority Queue ADT
- concept
- operations
- implementation options

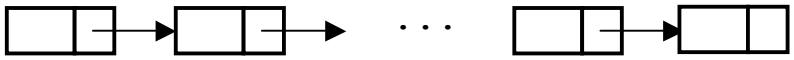
Next Week

Read: start *Recursion*
Recursion
- recursion vs. iteration
- constructing recursive code
- practice writing recursive code

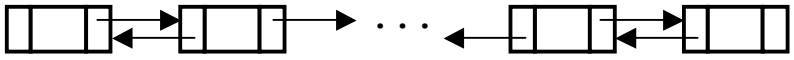
Double and Circular Linking

Doubly-Linked Chains of Nodes

Circular Singly-Linked Chains of Nodes



Circular Doubly-Linked Chains of Nodes

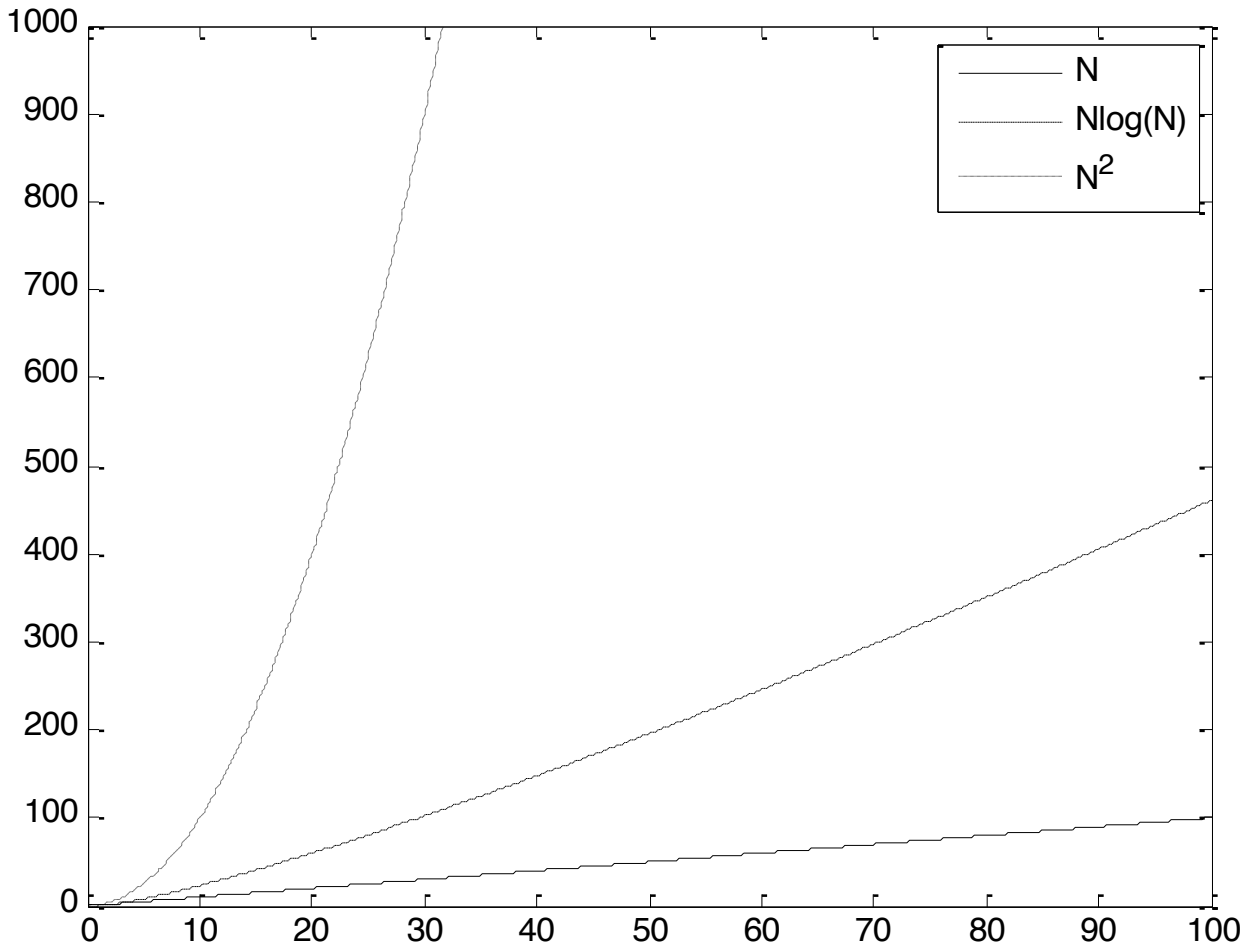


Analyzing Algorithm Efficiency

Complexity

If problem size doubles and the number of operations:

Example: Complexity Analysis of Giving a Toast



Complexity Analysis:

-

-

Big-O Notation

Concept

some growth rate functions:

Simplifying Equations

Formal Definition

Complexity of Java Code

Basic operations

Sequence of statements

```
statement1;
statement2;
...
statementk;
```

If-else

```
if (cond) {
   //if sequence of statements
}
else {
   //else sequence of statements
}
```

Complexity of Java Code (cont.)

Basic loops

→ What is the problem size based on?

```
for (i = 0; i < j; i++) {
   //sequence of statements
}
```

Nested loops

→ What is the problem size based on?

```
for (i = 0; i < N; i++) {
   for (j = 0; j < M; j++) {
      //sequence of statements
   }
}
```

Loops with nested method calls (assume problem size based on N)

```
for (i = 0; i < N; i++) {
   f1(i);  //assume O(1)
}
```

```
for (i = 0; i < N; i++) {
   f2(N);  //assume O(N)
}
```

```
for (i = 0; i < N; i++) {
   f3(i);  //assume O(i)
}
```

Practice - Complexity of Java Code

method1

→ What is the problem size based on?

```
public void method1(int[] A) {
   for (int i = 0; i < A.length - 1; i++)
      method2(A, i);
}
```

method2

```
public void method2(int[] B, int s) {
   for (int i = s; i < B.length - 1; i++)
      if (B[i] > B[i+1])
         method3(B, i, i+1);
}
```

method3

```
public void method3(int[] C, int x, int y) {
   int temp = C[x];
   C[x] = A[y];
   C[y] = temp;
}
```

Practice - Complexity of Java Code

method4

→ What is the problem size based on?

```
public void method4(int Q) {
   int sum = 0, R = 1000;

   for (int i = Q; i >= 1; i--)
      for (int j = 0; j < R; j++)
         sum += j;
}
```

method5

→ What is the problem size based on?

```
public void method5(int X) {
   int tmp, arr[];

   arr = new int[X];
   for (int i = 0; i < X; i++)
      arr[i] = X - i;

   for (int i = 0; i < X - 1; i++) {
      for (int j = i; j < X - 2; j++) {
         if (arr[j] > arr[j+1]) {
            tmp = arr[j];
            arr[j] = arr[j+1];
            arr[j+1] = tmp;
         }
      }
   }
}
```

Number Guessing Game

Picker picks a number (positive integer)
Repeat until number is guessed:
  Guesser guesses a number
  Picker answers "correct", "higher", or "lower"

problem size:
dominant operation:

→ **What is the complexity of each algorithm below** that the guesser uses to decide the sequence of numbers to give as guesses?

Algorithm 1:

```
guess = 1
repeat
   If guess incorrect, increment guess by 1
until correct
```

Algorithm 2:

```
guess =    /2
step  =    /4
repeat
   If guess is too small, increase guess by step
   otherwise decrease guess by step
   step = step/2 (alternate rounding up/down)
until correct
```

The Significance of Scaling

N	N log(N)	N^2	2^N	N!
2	2.0	4	4	2
4	8.0	16	16	24
6	15.5	36	64	720
8	24.0	64	256	
10	33.2	100	1024	
15	58.6	225		
20	86.4	400		
100	664.4	10,000		
1000	9965.8	1,000,000		

Complexity Caveats

Small Problem Size

Same Complexity

Comparing `ListADT` Implementations

Time Requirements
Problem size N is number of items

	construct or	add (E) "at end"	add (int,E) "at pos"	contains (E)	size	IsEmpty	get (int)	remove (int)
Array								
Singly-Linked List (SLL)								
Circular SLL								
Doubly-LL								
CircularDLL								

Comparing `ListADT` Implementations

Space Requirements

→ Problem size N is ?

Array:

Singly-Linked List:

Circular Singly-Linked List:

Doubly-Linked List:

Circular Doubly-Linked List:

Comparing `ListADT` Implementations

Ease of Implementation

Array:

Singly-Linked List:

Circular Singly-Linked List:

Doubly-Linked List:

Circular Doubly-Linked List:

Returning N Papers to N Students

problem size (N) = number of students
dominant operation ?

→ What is the complexity of each algorithm below?

Algorithm 1:
call out each name,
have student come forward & pick up

best-case:

worst-case:

Algorithm 2:
hand pile to first student,
student linearly searches through papers & takes hers/his,
pass pile to next student who does likewise

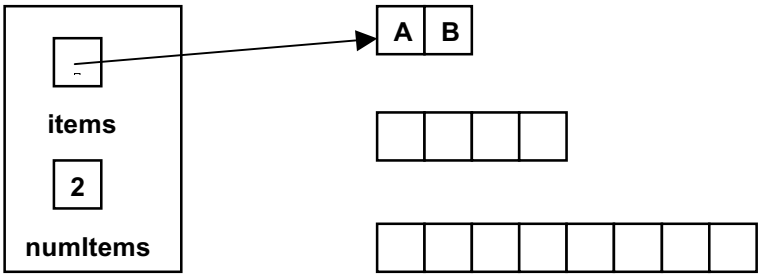
best-case:

worst-case:

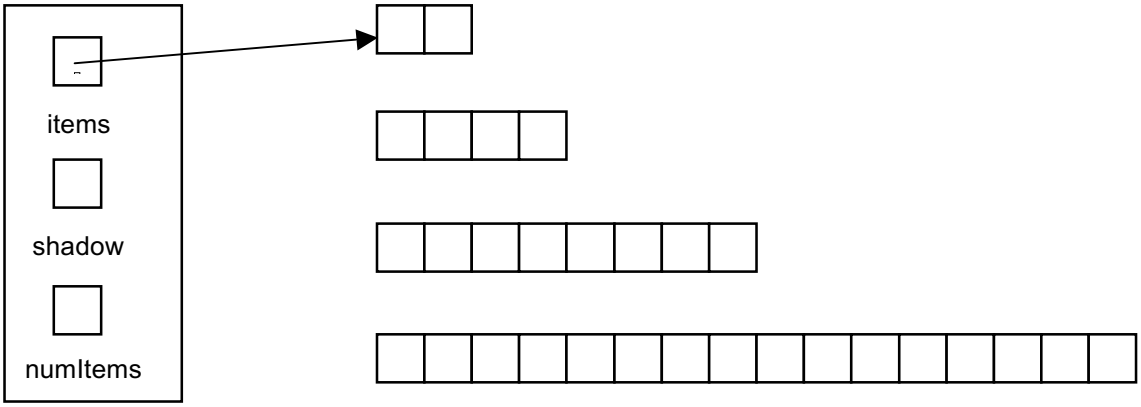
Algorithm 3:
sort the papers alphabetically,
hand pile to first student who does binary search,
pass to next student who does likewise

Shadow Array – Improving Array Resizing

"Naïve" Approach



"Shadow Array" Improvement



Stack ADT

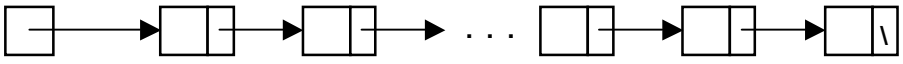
Concept

Operations

Implementing using an Array

→ Where should the top be located in the array?

Implementing using a Chain of Nodes



→ Where should the top be located in the chain of nodes?

Complexities

Queue ADT

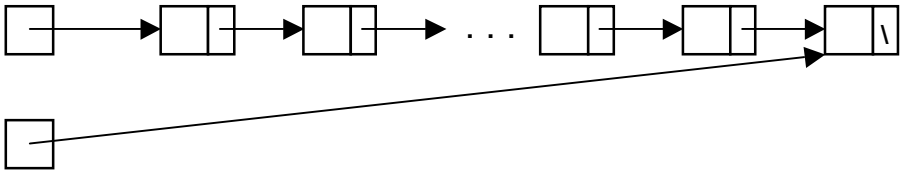
Concept

Operations

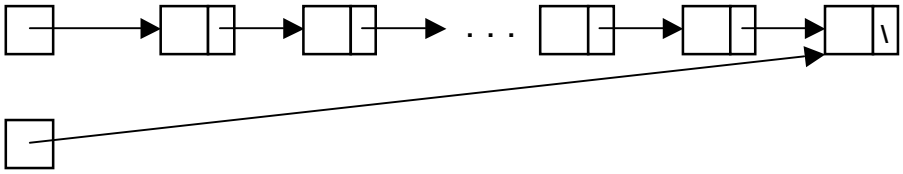
Implementing a using a Chain of Nodes

→ Is one option better than the other?

Option 1: front of queue is at head, rear of queue is at tail



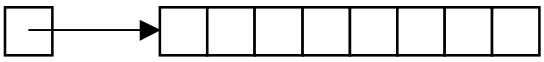
Option 2: front of queue is at tail, rear of queue is at head



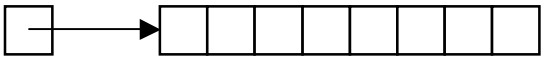
Implementing a Queue ADT using an Array

Assume a shadow array is used so that expand is O(1).

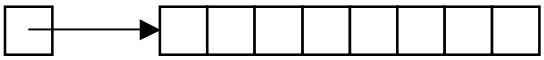
Option 1: front of queue is at _____, rear of queue is at _____



Option 2: front of queue is at _____, rear of queue is at _____



Option 3: front of queue is at _____, rear of queue is at _____



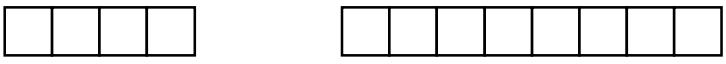
Implementing a Queue ADT using Circular Array

Concept

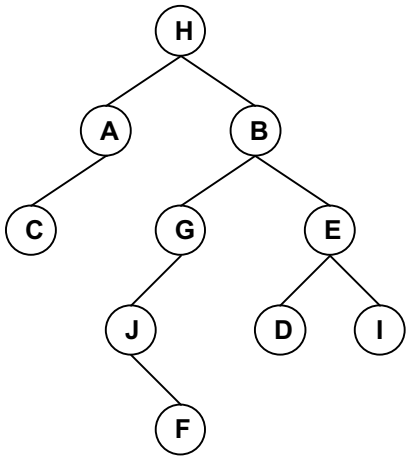
enqueue(item)

dequeue()

expand()



Tree Terminology



1. Which is the **root**?

2. How many **leaves** are there?

3. How many nodes are in the right **branch/subtree** of B?

4. Which is the **parent** of G?

5. How many **children** does E have (**degree** of E)?

6. Which is the **sibling** of E?

7. How many **descendants** does B have?

8. What are the **ancestors** of C?

9. What is the **length** of the **path** from B to D?

10. What is the **height** of the tree?

11. What is the **depth/level** of J?

Priority Queue ADT

Priorities

Concept

goal:

Operations

Options for Implementing a Priority Queue ADT

data structure	`insert`	`removeMax`
unordered array		
ordered array		
unordered chain of nodes		
ordered chain of nodes		