CSE 373: Data Structures and Algorithms
Lecture 2: Wrap up Queues, Asymptotic Analysis, Proof by Induction

Instructor: Lilian de Greef
Quarter: Summer 2017
Today:

• Announcements
• Wrap up Queues
• Begin Asymptotic Analysis: Big-O
• Proof by Induction
Announcement: Office Hours

• Announced! See course webpage for times
• Most held in 3rd floor breakouts in CSE (whiteboards near stairs)
• Lilian’s additional "actual office" office hours
  • CSE 220 (a more private environment)
  • During listed times
  • And by appointment! (email me >24 hours ahead of time with several times that work for you)
  • Come talk to me about anything! (feedback, grad school, Ultimate Frisbee, life problems, whatever)
Announcement: Sections

• When & where: listed on course webpage
• What: TA-led...
  • Review sessions of course material
  • Practice problems
  • Question-answering
• Optional, but highly encouraged!

I wouldn't have passed 332 (Data Structures and Parallelism) without regularly going to section! – Vlad (TA)
Other Announcements

• Homework 1 is out
  • On material covered in Lecture 1
  • Go forth!
  • ...or at least get Eclipse set up today.

• Only required course reading:
  • 10 pages, easy read on commenting style
  • Due beginning of class on Monday

• July 3rd
  • Not an official UW holiday (sorry guys)
  • But I’m declaring it an unofficial holiday! Go enjoy a 4-day July 4th weekend

<table>
<thead>
<tr>
<th>University Holidays</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes are not in session on the following holidays:</td>
</tr>
<tr>
<td>SUMMER 2017</td>
</tr>
<tr>
<td>Full-term</td>
</tr>
<tr>
<td>July 4, 2017</td>
</tr>
<tr>
<td>Independence Day</td>
</tr>
</tbody>
</table>
Finishing up Queues

Let’s resolve that cliff-hanger!
If we can assume the queue is not empty, how can we implement dequeue()?

```java
public E dequeue() {
    size--;    
    E e = array[front];
    //Your code here!
    return e;
}
```

A) `front++;
   if (front == array.length)
       front = 0;

B) `rear = rear-1;`
   `if (rear < 0)`
       `rear = array.length-1;`

C) `for (int i = 0; i < rear; i++) {
    array[i] = array[i+1]
}
   front++;
   if (front == array.length)
       front = 0;

D) None of these are correct
(Notes for yourself)
If we can assume the array is not full, how can we implement enqueue(E e)?

Public enqueue(E e) {
    <Your code here!>
    size++;  
}

A) rear++;  
    if (rear == array.length)  
        rear = 0;  
    array[rear] = e;

B) rear++;  
    array[rear] = e;

C) for (int i=front; i<rear; i++) {
    array[i] = array[i+1]
}
    array[rear] = e;
    rear++;  

D) None of these are correct
(Notes for yourself)
Between arrays and linked-lists which one *always* is the fastest at enqueue, dequeue, and seeKthElement operations? (where seeKthElement lets you peek at the kth element in the stack)

<table>
<thead>
<tr>
<th>Fastest:</th>
<th>enqueue</th>
<th>dequeue</th>
<th>seeKthElement</th>
</tr>
</thead>
<tbody>
<tr>
<td>A)</td>
<td>Arrays</td>
<td>Linked-Lists</td>
<td>Neither</td>
</tr>
<tr>
<td>B)</td>
<td>Linked-lists</td>
<td>Neither</td>
<td>Neither</td>
</tr>
<tr>
<td>C)</td>
<td>Linked-lists</td>
<td>Neither</td>
<td>Arrays</td>
</tr>
<tr>
<td>D)</td>
<td><em>They’re all the same</em></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
(Notes for yourself)
Which one’s better?

Arrays

Linked-lists
Trade-offs!

• The ability to choose wisely between trade-offs is why it’s important to understand underlying data structures.

• Common Trade-offs
  • Time vs space
  • One operation’s efficiency vs another
  • Generality vs simplicity vs performance
Asymptotic Analysis

Oh ho! The Big-O!
Algorithm Analysis

• Why: to help choose the right algorithm or data structure for the job
• Often in asymptotic terms

• Most common way: Big-O Notation
  • General idea:

  • A common way to describe “worst-case running time”
Example #1:

The barn is an array of Cows, excitement is an integer, and Cow.addHat() runs in constant time.

println("The alien is visiting!");
println("Party time!");
excitement++;
for (int i=0; i<barn.length; i++) {
    Cow cow = barn[i];
    cow.addHat();
}

Let's assume that one line of code takes 1 "unit of time" to run. This is not always true, i.e. calls to non-constant-time methods.

Important! Always begin by specifying what “n” is! (or “x” or “y” or whatever letter)
Example #1:

```java
println("The alien is visiting!");
println("Party time!");
excitement++;
for (int i=0; i<barn.length; i++) {
    Cow cow = barn[i];
    cow.addHat();
}
```
Example #2: Your turn!

```java
for (Person player: sportsTeam) {
    player.smile();
    for (Person teamMate: sportsTeam) {
        player.say("Good game!");
        player.highFive(teamMate);
    }
}
```

Assume that the above Person method calls run in constant time
What’s the asymptotic runtime of this (semi-)pseudocode?

```plaintext
x := 0;
for i=1 to N do
    for j=1 to i do
        x := x + 3;
return x;
```

A. O(n)
B. O(n^2)
C. O(n + n/2)
D. None of the above
What’s the asymptotic runtime of this (semi-)pseudocode?

```
x := 0;
for i=1 to N do
    for j=1 to i do
        x := x + 3;
return x;
```

A. O(n)
B. O(n²)
C. O(n + n/2)
D. None of the above

How do we prove the right answer?
Proof by Induction!
Inductive Proofs

(Interlude from Asymptotic Analysis)
Steps to Inductive Proof

1. If not given, **define n** (or “x” or “t” or whatever letter you use)

2. **Base Case**

3. **Inductive Hypothesis (IHOP):**
   Assume what you want to prove is true for some arbitrary value k (or “p” or “d” or whatever letter you choose)

4. **Inductive Step:**
   Use the base case and IHOP to prove it's true for n = k+1
Example #0:
Proof that I can climb any length ladder

1. **Let** $n$ = number of rungs on a ladder.
2. **Base Case:** for $n = 1$
3. **Inductive Hypothesis (IHOP):**
   Assume true for some arbitrary integer $n = k$.
4. **Inductive Step:** (aiming to prove it's true for $n = k+1$)
   - If I climb $k$ steps of the ladder, then I have one step left to go.
   - By IHOP, I can climb $k$ steps of the ladder.
   - By Base Case, I can climb the last step.
   - So I can climb $k+1$ steps.
   - I can climb forever!
Example #1

Prove that the number of loop iterations is \( \frac{n \times (n + 1)}{2} \)

\[
\begin{align*}
x & := 0; \\
\text{for } i=1 \text{ to } N \text{ do} \\
    \quad \text{for } j=1 \text{ to } i \text{ do} \\
    \quad \quad x & := x + 3; \\
\text{return } x;
\end{align*}
\]
(Extra room for notes)
Example #2:
Prove that $1 + 2 + 4 + 8 + \ldots + 2^n = 2^{n+1} - 1$
(Extra room for notes)
Useful Mathematical Property!

\[ \sum_{i=0}^{n} 2^i = 2^{n+1} - 1 \]

You’ll use it or see it again before the end of CSE 373.
Powers of 2

- A bit is 0 or 1 (just two different “letters” or “symbols”)
- A sequence of $n$ bits can represent $2^n$ distinct things
  (For example, the numbers 0 through $2^n-1$)
- $2^{10}$ is 1024 (“about a thousand”, kilo in CSE speak)
- $2^{20}$ is “about a million”, mega in CSE speak
- $2^{30}$ is “about a billion”, giga in CSE speak

Java: an `int` is 32 bits and signed, so “max int” is “about 2 billion”
    a `long` is 64 bits and signed, so “max long” is $2^{63}-1$
Which means...

You could give a unique id to...
• Every person in the U.S. with 29 bits
• Every person in the world with 33 bits
• Every person to have ever lived with 38 bits (estimate)
• Every atom in the universe with 250-300 bits

So if a password is 128 bits long and randomly generated, do you think you could guess it?
Example #3: (Parody) Reverse Induction!
Proof by Reverse Induction That You Can Always Cage a Lion:

Let $n =$ number of lions

Base Case: There exists some countable, arbitrarily large value of $M$ such that when $n = M$, the lions are so packed together that it's trivial to cage one.

IHOP: Assume this is also true for $n = k$.

Inductive Step: Then for $n = k-1$, release a lion to reduce the problem to the case of $n = k$, which by the IHOP is true.

QED :)

Fun fact: Reverse induction is a thing! The math part of the above is actually correct.