Week 3 – Wednesday

COMP 2100

Last time

- What did we talk about last time?
- More on Big Oh
- ADTs
- List implementation with a dynamic array

Questions?

Assignment 2

Project 1

Bitmap Manipulator

Array backed list

```
public class ArrayList<E> {
   private E[] array;
   private int size;

   public ArrayList() {}
   public int  size() {}
   public void add(E element) {}
   public E get(int index) {}
   public boolean remove(Object o) {}
}
```

Constructor Implementation

Add Implementation

Remove implementation

Running time for `ArrayList` operations

- How long did each method we implemented last week take, in Θ notation, where *n* is the number of items already in the lists?
- Constructor
  - Θ(1)
- Add: Insert element at the end of the list
  - Θ(*n*) (but only in the worst case)
- Get: Retrieve element from arbitrary location
  - Θ(1)
- Size: Get the current number of elements stored
  - Θ(1)
- Remove: Remove an object from the list
  - Θ(*n*)

Amortized analysis

- **Amortized analysis** is one way to consider average running time
- The amortized cost per operation of *n* operations is the total cost divided by *n*
- If an operation is expensive only once in a while, the amortized running time might be much less than the worst case running time
  - A random search could make you take a long time to get through airport security
  - Since you don't usually get stopped, your average time isn't much different than when you don't get stopped

Amortized analysis of ArrayList

- In the Add operation, it usually only takes $\Theta(1)$ time to put an element at the end of the array
- The only time it takes $\Theta(n)$ time is when you have to resize the array
- To simplify the analysis, let's assume:
  - It takes 1 operation to add, ignoring the resize
  - It takes n operations (where n is the number of things already in the array) to resize
  - We are only adding, no other operations
- The amortized running time depends on your strategy for resizing

Strategy 0: Resize on every Add

- The most space efficient approach is to keep the array completely full
- Thus, you have to extend the array each time you add an item
- To add n items, you'll have the following resize costs:

  $1 + 2 + 3 + 4 + \ldots + (n - 1) = n(n - 1)/2$
- Plus, it would have cost an additional n to do the adds themselves
- Total cost: $n(n - 1)/2 + n$
- Amortized cost per operation: $(n - 1)/2 + 1$, which is $\Theta(n)$
- **Yuck.**

Strategy 1: Double the array length on resize

- If we double the length of the array when we resize, we won't have to resize as often
- To add *n* items where $2^k \leq n < 2^{k+1}$, you'll have the following resize costs:

  $1 + 2 + 4 + 8 + \ldots + (2^{k-1}) + 2^k =$

  $\sum_{i=0}^{k} 2^i = 2^{k+1} - 1 \leq 2n - 1$
- Plus, it would have cost an additional *n* to do the adds themselves
- Total cost: $3n - 1$
- Amortized cost per operation: essentially 3, which is $\Theta(1)$

Stacks

Stack

- A stack is a simple (but useful) ADT that has three basic operations:
  - **Push** Put an item on the top of the stack
  - **Pop** Remove an item from the top of the stack
  - **Top** Return the item currently on the top of the stack (sometimes called **peek**)

Keeping track of things

- When are stacks used?
  - Implicitly, in recursion (or in any function calls)
  - Explicitly, when turning recursive solutions into iterative solutions
  - When parsing programming languages
  - When converting infix to postfix

Array implementation of stack

```
public class ArrayStack<E> {
  private E[] data;
  private int size;


  public ArrayStack() {}
  public void push(E value) {}
  public E pop() {}
  public E peek() {} // Instead of top
  public int size() {}
}
```

Quiz

Upcoming

Next time…

- Array implementation of stacks
- Queues
- Array implementation of queues
- **Office hours from 4-5 p.m. cancelled today because of Faculty Assembly**

CAREER JUMPSTART EVENT

Engineering & Computer Science

THURSDAY, SEPTEMBER 12TH FROM 4:45PM-7PM

Otterbein University @ The Point

Come and network with alumni and recruitment partners and learn how to be successful with your field.

OTTERBEIN
UNIVERSITY

SCAN the QR CODE to REGISTER

Reminders

- Keep reading section 1.3
- Finish Assignment 2
  - Due this Friday by midnight!
- Keep working on Project 1
  - **Due next Friday, September 20 by midnight**