Announcements

- HW due Monday
- Next program will be out soon
Primitive Operations

As a way to compare algorithms in a generic way, we instead count primitive operations.

Of 2 #s

( addition, subtraction, memory access, return, mult & div )

In addition, we (generally) only analyze the worst possible running time.

Why? guaranteeing a minimum performance
Big-O

We say \( f(n) \) is \( O(g(n)) \) if \( \forall n > n_0, \exists c > 0 \) such that \( f(n) \leq c \cdot g(n) \).
Functions we will use

1. $O(1)$ - constant time
2. $O(\log n)$ - logarithmic time
   + Binary Search
3. $O(n)$ - linear time
4. $O(n \log n)$
5. $O(n^2)$ - quadratic time
6. $O(n^3)$ - cubic time
7. $O(2^n)$ - exponential time

$\log_2 a + \log_2 b = \log_2(ab)$
$\log_2 x^c = c \log_2 x$
Algorithms

Claim: Inserting an element into the first spot in an array is $O(n)$ time.

Claim: Inserting at the beginning of a list is $O(1)$ time.
**Common running times**

- A for loop which goes from \( i = 0 \) to \( n-1 \) and reads \( i \) to an array

```plaintext
for (int i=0; i < n; i++)
    cin >> array[i];
```

**Analyze:**

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

\[
= 2n = O(n)
\]
Nested For loops: find if any 2 elements are identical

```
for (int i = 0; i < n; i++)
    for (int j = i+1; j < n; j++)
        if (A[i][j] == A[j][i])
            cout << "Two items are the same" << endl;
```

Analyze:

\[
\begin{align*}
\sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} 4 & = 4 \sum_{i=0}^{n-1} (n-1-i) = 4 \left( \sum_{i=0}^{n-1} i \right) + \sum_{i=0}^{n-1} (n-1-i) \\
& = 4 \sum_{i=0}^{n-1} i + \sum_{i=0}^{n-1} (n-1-i) \\
& = 4 \frac{n(n+1)}{2} + \frac{n(n-1)}{2} = \frac{n^2}{2} \end{align*}
\]

\((4\cdot4\cdot4\cdots+4) = 4(1+1+\cdots+1) = 4(n-1-i) = O(n^2)\)
Stack: a way to store a list of data

Ex: Web browser: Store history for "back" button

Same behavior

Ex: Text editors: store previously used commands
  type "undo"
The stack ADT:

- Supports 2 main functions:
  - push(e): add e to "top" of the stack
  - pop(): remove e from the top of the stack

```
push(5)  pop()
push(11)  pop()
push(3)  pop()
pop()  push(12)
```
Others

- top(): returns top element of the stack without removing it.

- empty(): returns true if stack is empty.

- size(): returns # of objects in the stack.
Now, how to build?

We know the functions to list in the .h file.

But how to do private data to store things on the stack?

Ideas?

- Use an array
- Use a singly linked list
Array-based:

private:

int _size; // # of elements in stack
Object * _A;
int _capacity; // max # stack can hold
Other functions to code

Housekeeping functions:
- Destructor
- Operator =
- Copy Constructor