CS2100

More on housekeeping
Asymptotic analysis
Announcements
Last time

- Stacks implementation

2 options:

Trade off:
Finishing up "housekeeping":
3 Functions:

We (mostly) finished, but let's re-check the details.
Next: Asymptotic Analysis

Motivation: How should we compare 2 programs?
Speed:

- Exact speed can depend on many variables besides the algorithm.

Issues at play:

Alternative approach:

Count primitive operations, which are smallest operations.

In addition: generally only examine worst-case running time.

Why?
Now: How to actually compare?
- Remember small difference may be due to processor, language, or any number of things that aren't dependent on the algorithm.
- Also: need a way to account for inputs changing eg searching a list
Big-O notation
We say $f(n)$ is $O(g(n))$ if
$
\forall n > n_0, \exists c > 0 \text{ such that } f(n) \leq c \cdot g(n)
$
Examples

1. $5n$ is $O(n^2)$

2. $5n$ is $O(n)$

3. $16n^2 + 2n$ is $O(n^2)$
Common run times

1. $O(1)$
2. $O(\log n)$
3. $O(n)$
4. $O(n \log n)$
5. $O(n^2)$

(polynomial)

And: $O(2^n)$
$O(n!)$
Claim: Inserting a new element at the beginning of an array is $O(n)$ time.

pf:
Claim: Inserting an element at the head of a list is $O(1)$ time.
Nested for loops:

Ex: Find if any 2 elements in the array are equal.

```java
for (int i = 0; i < n; i++)
    for (int j = 1; j < n; j++)
        if (A[i] == A[j])
            return true;

return false;
```
From here on out, we'll use this analysis for any function or data structure we code.

Some may be obvious:

Some harder:
Runtime of Stack operations