Discussion 7:

Linked Lists, OOG, & Midterm Review

Caroline Lemieux (clemieux@berkeley.edu)

Pi Day, 2019

Announcements

Homeworks

HW 6 and Lab 7 due tomorrow!

Projects

Ants due tonight!

Midterm 2

Next Tuesday!

Linked Lists

Link Class

1) `lnk.first`
2) `lnk.rest`
3) `lnk is Link.empty`

Box and Pointer Diagram

From Spring 2016 MT 2:

```
L = Link(1, Link(2))
P = L
Q = Link(L, Link(P))
P.rest.rest = Q
```

Attendance

links.cs61a.org/caro-disc

Orders of Growth

Runtime

- We care about the speed of programs
- **Big question:** How does the runtime of a program *change*, or *grow*, as the input size grows?
- We will be answering this question for various functions
- We answer question this by roughly estimating the **number of operations** as a function of the size of the input

Input type	Input size
number	magnitude of number (i.e. how big the number is)
list	length of the list
tree	number of nodes in the tree

No growth

Big question: How does the runtime of a program *change*, or *grow*, as the input size grows?

```python
def square(n):
    return n * n
```

No matter how big n is, `square(n)` always only takes 1 operation.

The runtime **doesn't grow** as the input size grows!

input	function call	return value	number of operations
1	square(1)	1*1	1
2	square(2)	2*2	1
...	...	...	...
100	square(100)	100*100	1
...	...	...	...
n	square(n)	n*n	1

Proportional growth

Big question: How does the runtime of a program *change*, or *grow*, as the input size grows?

```
def fact(n):
    if n == 0:
        return 1
    return n * fact(n-1)
```

The bigger n gets, the more operations we have to do.

The runtime of this function **grows proportionally to the input size.**

input	function call	return value	number of operations
1	fact(1)	1*1	1
2	fact(2)	2*1*1	2
...	...	...	...
100	fact(100)	100*99*...*1*1	100
...	...	...	...
n	fact(n)	n*(n-1)*...*1*1	n

Big Theta Notation

For this course, we want to give an lower and upper bound on runtime.

θ(f(n)) = “The function’s runtime will no worse and no better than the f(n), where n is input size.

Order of Growth	Name	Description
`θ(1)`	constant	Runtime is always the same regardless of input size, e.g. `square(n)`
`θ(log n)`	logarithmic	Input size is repeatedly reduced by some factor
`θ(n)`	linear	Runtime is proportional to input size, e.g. `factorial(n)`
$\theta(n^2)$, $O(n^3)$, etc.	polynomial	Variable number of operations per 1 unit of input (e.g. nested loops)
$\theta(2^n)$	exponential	Repeatedly multiplying the input size (e.g. tree recursion)

Recursive orders of growth

```
def sum_of_factorial(n):
    if n == 0:
        return 1
    else:
        return factorial(n) + sum_of_factorial(n - 1)
```

each call to `sum_of_factorial` calls `factorial`, which takes O(n) time, and sum_of_factorial(n-1), which takes ….

```
factorial(n) + sum_of_factorial(n - 1)
```
O(n)

```
factorial(n - 1) + sum_of_factorial(n - 2)
```
O(n - 1)

```
factorial(n - 2) + sum_of_factorial(n - 3)
```
O(n - 2)

...

Midterm Review!