LAB 09

Midterm Review

Mingxiao Wei
mingxiaowei@berkeley.edu

Oct 24, 2022

LOGISTICS 🏡

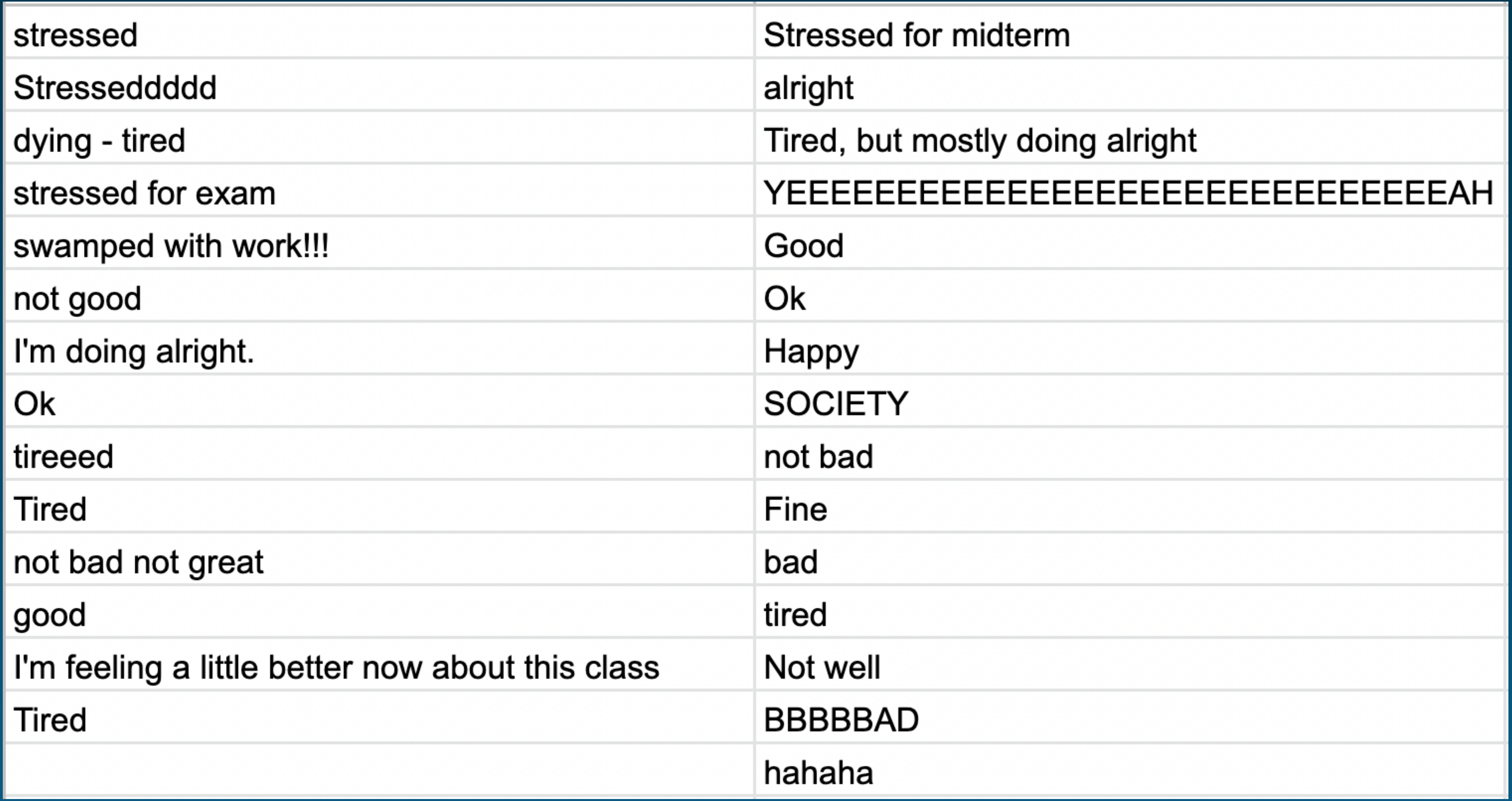
- No assignment due! Focus on the midterm :)
- Email confirmations for alteration requests will be sent out by Tue 10/25
- Seating assignments will be released via email by Wed 10/26
- Reminder - Homework 05 Recovery (Ed post #2128) due today 10/24

ABOUT THE MIDTERM 😮

- Logistics - Ed post #2141
  - If you need alterations, you can still fill out this form - approval is not guaranteed though since it's past the priority deadline
- Familiarize yourself with the study guide - this is a good starting point to go over the topics!
- Preparations
  - Familiarize yourself with the topics in scope
  - Attend review session (or watch recordings/slides) for more topical review - see Ed for more info
  - Do practice exams!
    - Quality > quantity
    - Post on exam threads on Ed for help
    - Walkthrough videos/guide are your friend!

FROM LAST TIME... 👀

How are you feeling now?



stressed	Stressed for midterm
Stresseddddd	alright
dying - tired	Tired, but mostly doing alright
stressed for exam	YEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEAH
swamped with work!!!	Good
not good	Ok
I'm doing alright.	Happy
Ok	SOCIETY
tireeed	not bad
Tired	Fine
not bad not great	bad
good	tired
I'm feeling a little better now about this class	Not well
Tired	BBBBBAD
	hahaha

ORDER OF GROWTH

📈

ORDER OF GROWTH

- Order of growth (efficiency) - how the runtime of the function changes as the input size increases
- Input size (not the definition, but as a rule of thumb)
  - numeric input - magnitude of the number
  - Python lists - length of the list
  - linked list/trees/other recursive objects - number of nodes
- Runtime (not the definition, but as a rule of thumb)
  - often measured as the number of operations
- For 61A, we use the theta notation - for input of size n, the runtime of the function is denoted by $\Theta(f(n))$

ORDER OF GROWTH - OTHER NOTES

- constant < logarithmic < linear < quadratic < exponential
- Constants are ignored
  - For example, $\Theta(2n + 3)$ is essentially $\Theta(n)$
- We only consider the term that grows fastest
  - For example, $\Theta(n^2 + 2n + 3)$ is essentially $\Theta(n^2)$

ORDER OF GROWTH - CONSTANT

- Constant ↔ Θ(1)
- The runtime of the function does not change as the input size changes
- For example:

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

input	function call	return value	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

ORDER OF GROWTH - LOGARITHMIC

- Logarithmic $\leftrightarrow$ $\Theta(\log n)$
- Often when we keep dividing the input by a constant
- For example:

```python
def foo (x):
    while x > 0:
        print('hey')
        x //= 2
```

- Suppose the while loop runs n times before x reaches 0.
- Since we divide x by 2 for n times, $\frac{x}{2^n} = 1 \implies 2^n = x \implies n = \log_2 x$

ORDER OF GROWTH - LINEAR

- Linear $\leftrightarrow$ $\Theta(n)$
- For example:

```
def factorial(x):
    prod = 1
    for i in range(1, x + 1):
        prod *= i
    return prod
```

input	function call	return value	operations
1	`factorial(1)`	1*1	1
2	`factorial(2)`	2*1*1	2
...	...	...	...
100	`factorial(100)`	100*99*...*1*1	100
...	...	...	...
n	`factorial(n)`	n*(n-1)*...*1*1	n

ORDER OF GROWTH - QUADRATIC

- Quadratic $\leftrightarrow \Theta(n^2)$
- For example:

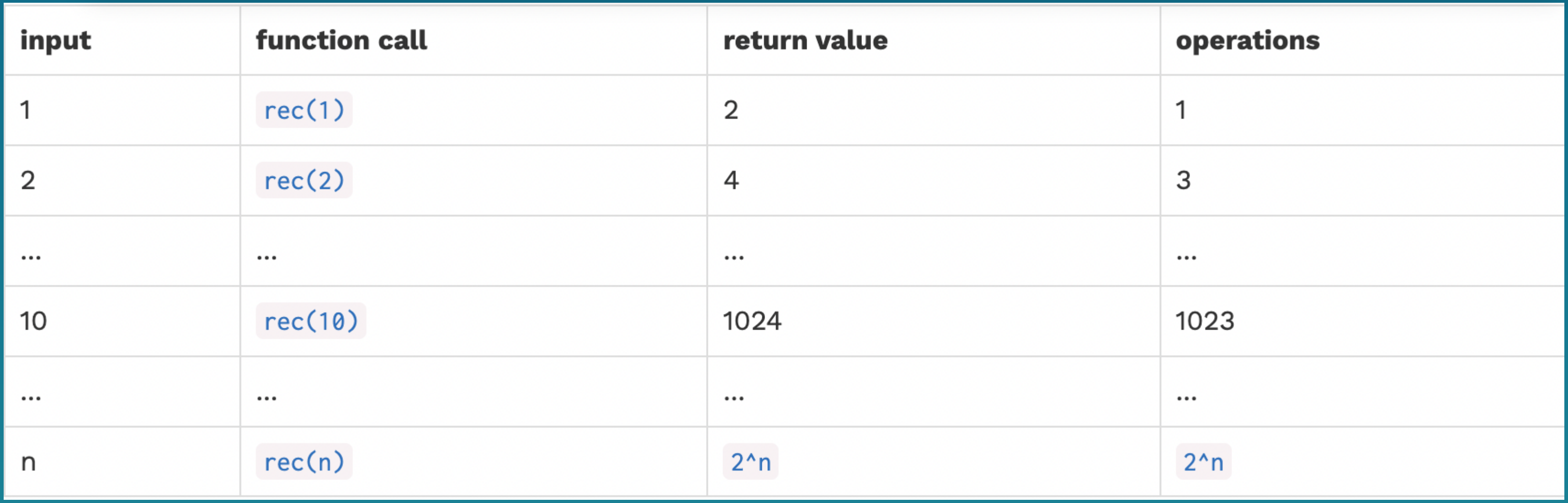
```python
def bar(n):
    for a in range(n):
        for b in range(n):
            print(a,b)
```

input	function call	operations (prints)
1	`bar(1)`	1
2	`bar(2)`	4
...	...	...
100	`bar(100)`	10000
...	...	...
n	`bar(n)`	n^2

ORDER OF GROWTH - EXPONENTIAL

- Exponential $\leftrightarrow \Theta(c^n)$, where c is a constant
- For example:

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



input	function call	return value	operations
1	`rec(1)`	2	1
2	`rec(2)`	4	3
...	...	...	...
10	`rec(10)`	1024	1023
...	...	...	...
n	`rec(n)`	`2^n`	`2^n`

ORDER OF GROWTH - TREE RECURSION

Draw out the tree recursion diagram, and do either of the following:

- Count by level
    - (work per level) * (# of levels)
    - useful when the work per level sums to the same number
- Count by node
    - (work per node) * (# of nodes)
    - useful when the work per node is the same

ORDER OF GROWTH - NESTED LOOPS

- (outer loops) * (middle loops) * (inner loops) * (work done in the inner loop)
- More generally, (# times the loops run) * (work done each time)

```
def factorial(n):
    # returns n! in linear time
    ...

def foo(n):
    for i in range(n):
        for j in range(n):
            print(factorial(n))
```

- `foo(n)` runs in $\Theta(n^3)$ time

ORDER OF GROWTH - NESTED FUNCTION CALLS

- Evaluate from the innermost call to the outermost one, and **add** the runtime together
- For outer functions, pay attention to their input in terms of n

```
def square(n):
    # returns n * n in constant time
def fact(n):
    # returns n! in linear time
def boo(n):
    for i in range(n):
        print('hi')
    return n
```

- `fact(square(n))`: $\Theta(1 + n^2) = \Theta(n^2)$
- `fact(boo(n))`: $\Theta(n + n) = \Theta(n)$
- `boo(square(n))`: $\Theta(1 + n^2) = \Theta(n^2)$

ATTENDANCE! 🤠

go.cs61a.org/mingxiao-att

- The attendance form and slides are both linked on our section website!
- Once again, please do remember to fill out the form by midnight today!!