Asymptotics

CS 61B Spring 2016: Discussion 8

Announcements

Project 2 is due .... yesterday! How many are still working to complete with a late penalty?

Thanks for completing the survey on Project 2!

- It was helpful for us to know what you thought and felt as well as get your feedback.

The goal was for project 2 to be a rewarding but challenging learning experience.

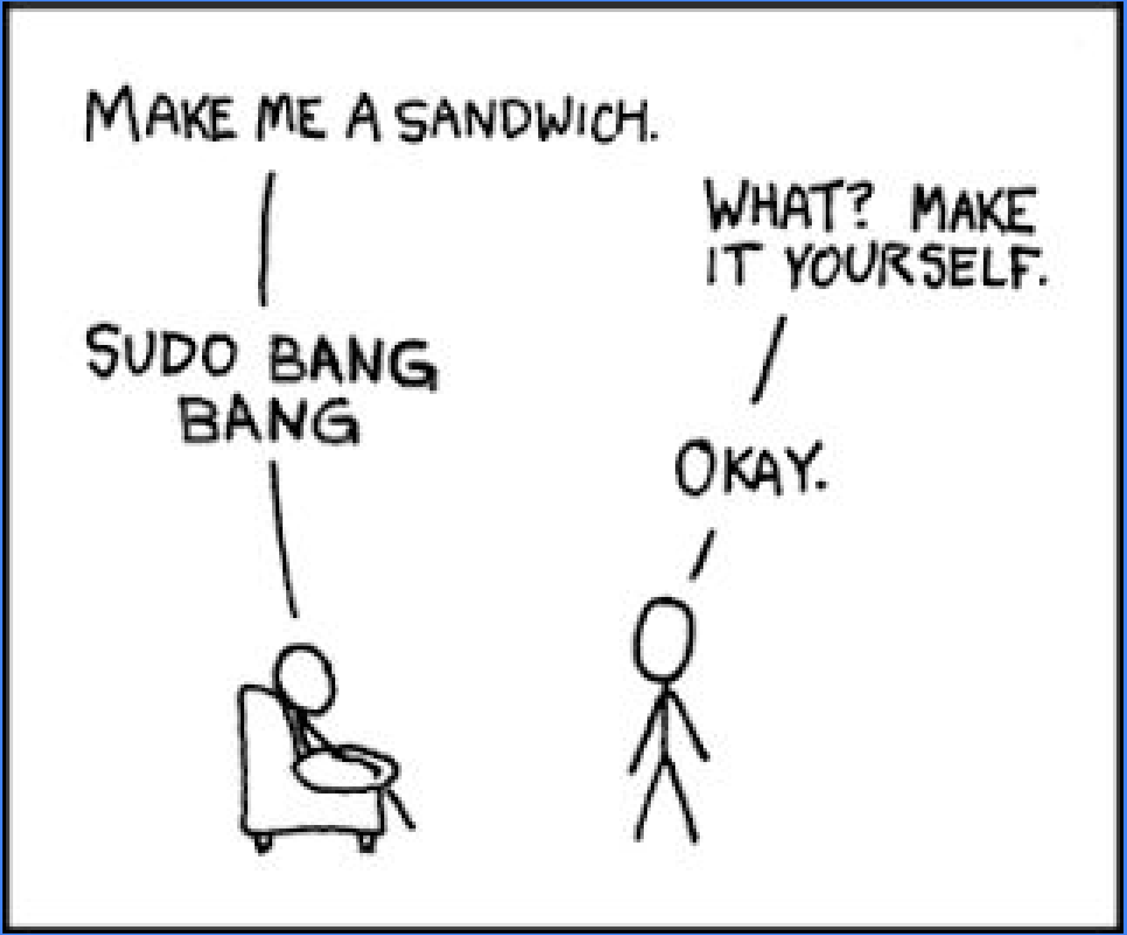
UNIX Tip of the Day: !!

UNIX Tip of the Day: !! (Bang Bang)

```
$ add proj2/editor/Editor.java
-bash: add: command not found
$ git !!
git add proj2/editor/Editor.java
$ commit -m “My very long and descriptive commit message because it’s good Git style for improved code health and source control.”
-bash: commit: command not found
$ git !!
git commit -m “My very long and descriptive commit message because it’s good Git style for improved code health and source control.”
```

UNIX Tip of the Day: !! (Bang Bang)

```
$ cp path/to/local/file
# oops, forgot to include destination path
$ !! path/to/destination
cp path/to/local/file path/to/destination
# not as useful because we probably could have just used the up arrow and then finished our command
```

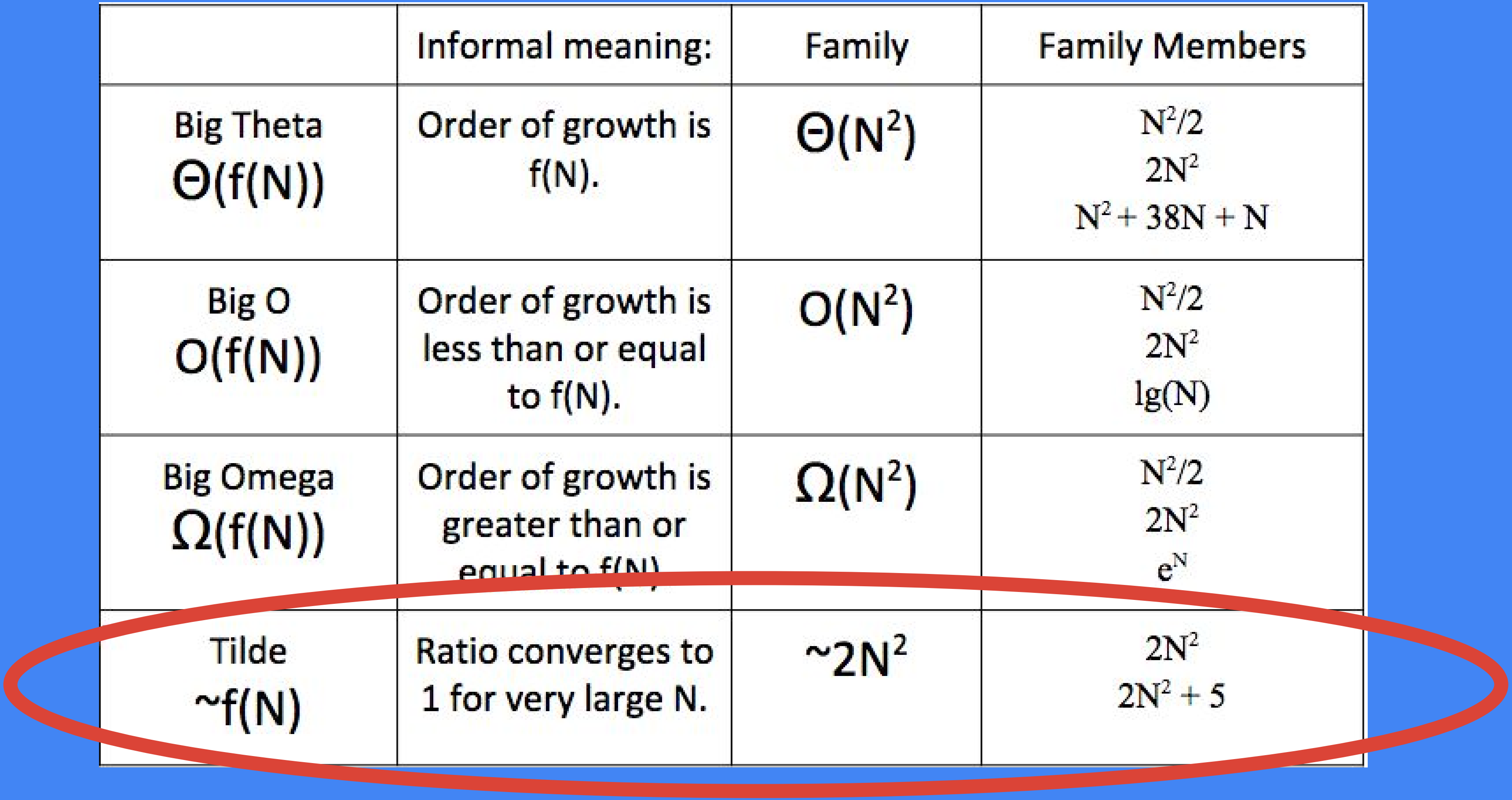


On UNIX-based systems, prepending a command with sudo runs the command using administrator privileges.

The equivalent in Windows is right-clicking a program and selecting “Run as administrator”.

Recall

- Big-O: Used for bounding above (less than).
- Big-Ω (Big-Omega): Used for bounding below (greater than).
- Big-Θ (Big-Theta): Used for bounding both above and below (equals).



	Informal meaning:	Family	Family Members
Big Theta $\Theta(f(N))$	Order of growth is f(N).	$\Theta(N^2)$	$N^2/2$ $2N^2$ $N^2 + 38N + N$
Big O $O(f(N))$	Order of growth is less than or equal to f(N).	$O(N^2)$	$N^2/2$ $2N^2$ $\lg(N)$
Big Omega $\Omega(f(N))$	Order of growth is greater than or equal to f(N).	$\Omega(N^2)$	$N^2/2$ $2N^2$ e^N
Tilde $\sim f(N)$	Ratio converges to 1 for very large N.	$\sim 2N^2$	$2N^2$ $2N^2 + 5$

Which is faster?

Problem 1

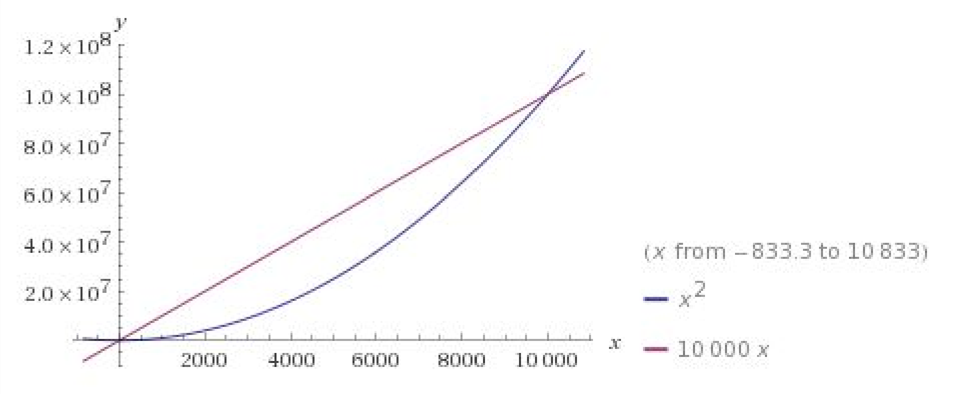
Problem 1

1. Algorithm 1. Θ gives tightest bounds, so $\Theta(N) < \Theta(N^2)$.
2. Neither. Something in $\Omega(N)$ could also be in $\Omega(N^2)$.
3. Neither. Something in $O(N^2)$ could also be in $O(1)$.
4. Algorithm 2. Algorithm 2 cannot be any slower than $O(\log N)$ while Algorithm 1 is constrained in the best and worst case by $\Theta(N^2)$.
5. Neither. Algorithm 1 **could** be faster, but not guaranteed. It is only guaranteed to be "as fast as or faster" Algorithm 2.

Problem 1

We cannot really make any claims about asymptotics for small values of N.

However, as an example, for small N, consider something like N^2 vs. 10000N:



N^2 is smaller for N < 10000, but larger for N > 10000.

(plot credits to WolframAlpha)

More analysis!

Problem 2

Problem 2

A. $\Theta(N^2)$

First run of inner loop is N/2, then N/2 - 1, then N/2 - 2, for a total of N/2 times (since outer loop runs N/2 times).

So, we have N/2 iterations (outer loop) times roughly N/2 iterations each time (inner loop), for a total of

$$\Theta((N/2)^2) = \Theta(N^2/4) = \Theta(N^2).$$

Problem 2

B. Θ(N)

Yes, the outer loop only happens $\log_2 N$ times. But the inner loop does i iterations, depending on whatever i is. Adding the values of i up (ignoring the -1 since it does not matter in this case), we get:

$$N + N/2 + N/4 + N/8 + \ldots < 2N$$

Note: This series converges to 2N.

Therefore, the algorithm is Θ(N), or linear.

EVEN MORE!

Problem 3

Problem 3a

The overall runtime of this algorithm can be described as Ω(1) ("best case") and O($\log_2 N$) ("worst case"). There is no overall Big-Theta runtime in this case.

In the best case, we start at the middle and immediately find `arr[mid] == mid`.

In the worst case, we go through all of the indices using this modified **binary search** algorithm, which recursively calls itself as most $\log_2 N$ times before hitting the base case and returning -1.

Problem 3a

Bonus: What is the method doing?

It looks for an element in arr such that arr[i] = i and returns that element if it exists. If it does not exist, it returns -1.

Problem 3b

`str.toCharArray()`: takes a string and returns an array of `char` such that we can iterate over it

`map.put(key, value)`: puts the (key, value) pair into the map (“dictionary”)

`map.containsKey(key)`: returns whether or not the key is in the map

`str.charAt(i)`: returns the character at index i in the string str

Problem 3b

In the best case, Θ(N).

In the worst case, Θ(N).

The first for loop will always require Θ(N) iterations to update the map (no ending early). The second loop **may end early**, but also iterates at most N times (so it is also Θ(N)).

Problem 3b

Bonus: What is the method doing?

It finds the first unique character in str and returns it. If there is no such unique character, it returns 0 (which is the NULL character -- more on this when you take CS 61C!).

Problem 3b

Bonus Bonus: Can you do it with only 1 for loop?

Nope.

Problem 3b

Bonus Bonus: Can you do it with only 1 for loop?

~~Nope.~~

Just kidding. We wouldn’t put it on the worksheet otherwise, right?

Problem 3b

Bonus Bonus: Can you do it with only 1 for loop?

Use 2 data structures instead of just the single HashMap.

1. Set A (“repeats”) to store characters that we have seen repeats of.
2. List B (“uniques”) to store characters we have only seen once so far.

Then, iterate through string. If character not in List B, then add it to List B. Else, remove it from List B and add it to Set A. After the for loop, return the **first character** in List B.

Problem 3b

```
Set<Character> repeats = new HashSet<>();
List<Character> uniques = new ArrayList<>();
for (int i = 0; i < str.length(); i++) {
  char chara = str.charAt(i);
  if (repeats.contains(chara)) {
    continue;
  }
  if (uniques.contains(chara)) {
    uniques.remove((Character) chara);
    repeats.add(chara);
  } else {
    uniques.add(chara);
  }
}
return uniques.get(0);
```

Runtime?

$O(N^2)$.

Removing from the ArrayList takes O(N) time, and in the worst case, we may have to remove every character in the string from the ArrayList.