CS 170 Section 1
Divide-and-Conquer

Owen Jow | owenjow@berkeley.edu

Agenda

- Introduction
- Asymptotics review
- Divide-and-conquer
- Master theorem
- Complex numbers

Introduction

About Me

- Owen Jow
- owenjow@berkeley.edu
    - but please only email me as a last resort, or if you have an administrative question
    - your alternatives for getting help include cs170@berkeley.edu, Piazza, OH, and your classmates
- Section: Wednesday 2-3pm in 3105 Etcheverry
- OH: Tuesday 11-12 pm in 411 Soda

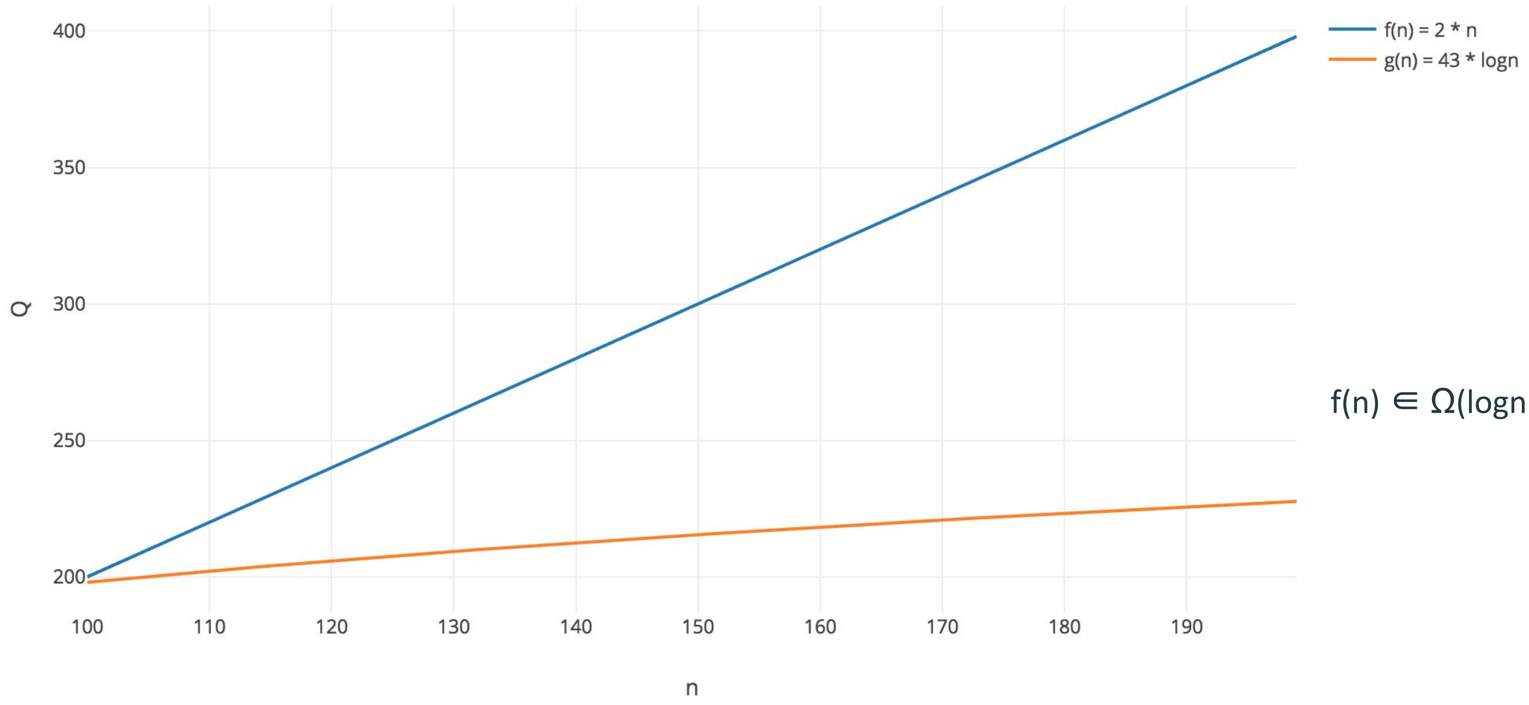
About CS 170

- Berkeley's premier CS algorithms course
    - Important for theory, software engineering interviews, and everything
- My main tip for success:
    - **More than anything else, focus on understanding and solving the homeworks on your own.** Start them early! *Immediately* after they come out, read all the problems and take the time to process them in your mind. Try to finish them as soon as possible, but hold off on asking other people for help. That being the case, you should be drawing out new approaches whenever you have time – it's okay to eat lunch with your problem set.
        - If you're keeping up with the homeworks, you're keeping up with the class.
        - If you're able to do the homeworks on your own, you're also in good stead for the exams!
- I also highly recommend the textbook (*Algorithms* by Dasgupta, PapaD, and our very own Vazirani).

Asymptotics Review

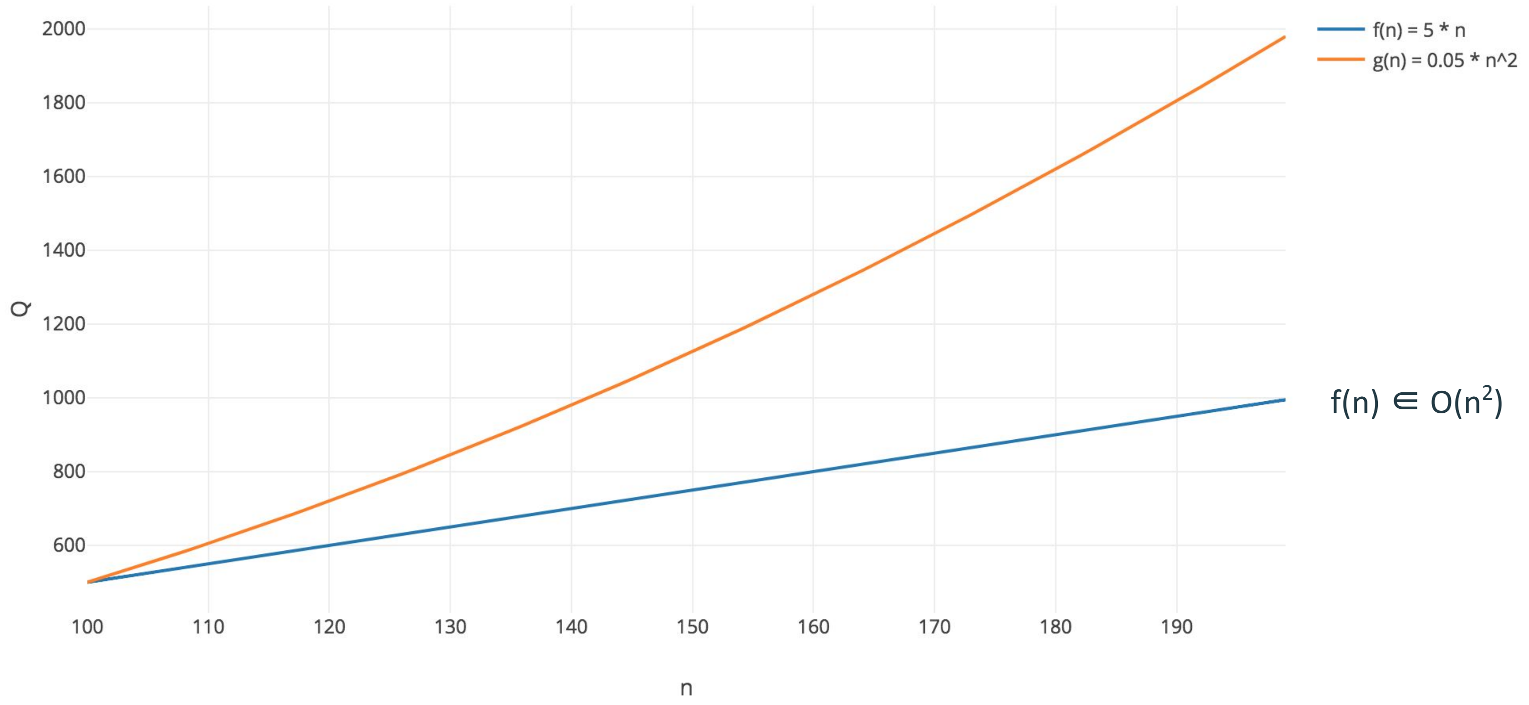
The Big-O Problem

- We are describing the **growth** of a quantity Q with respect to another quantity n
- We do this using functions: Q = f(n)
- However, it's messy to have every individual function be different ($5.9n^{3.1} + 66$, anyone?)…
  ...so we separate our growth functions into classes, and place each individual function into a class via big-Ω, big-Θ, or big-O notation.
- Typical classes are functions like 1, n, or n^2.



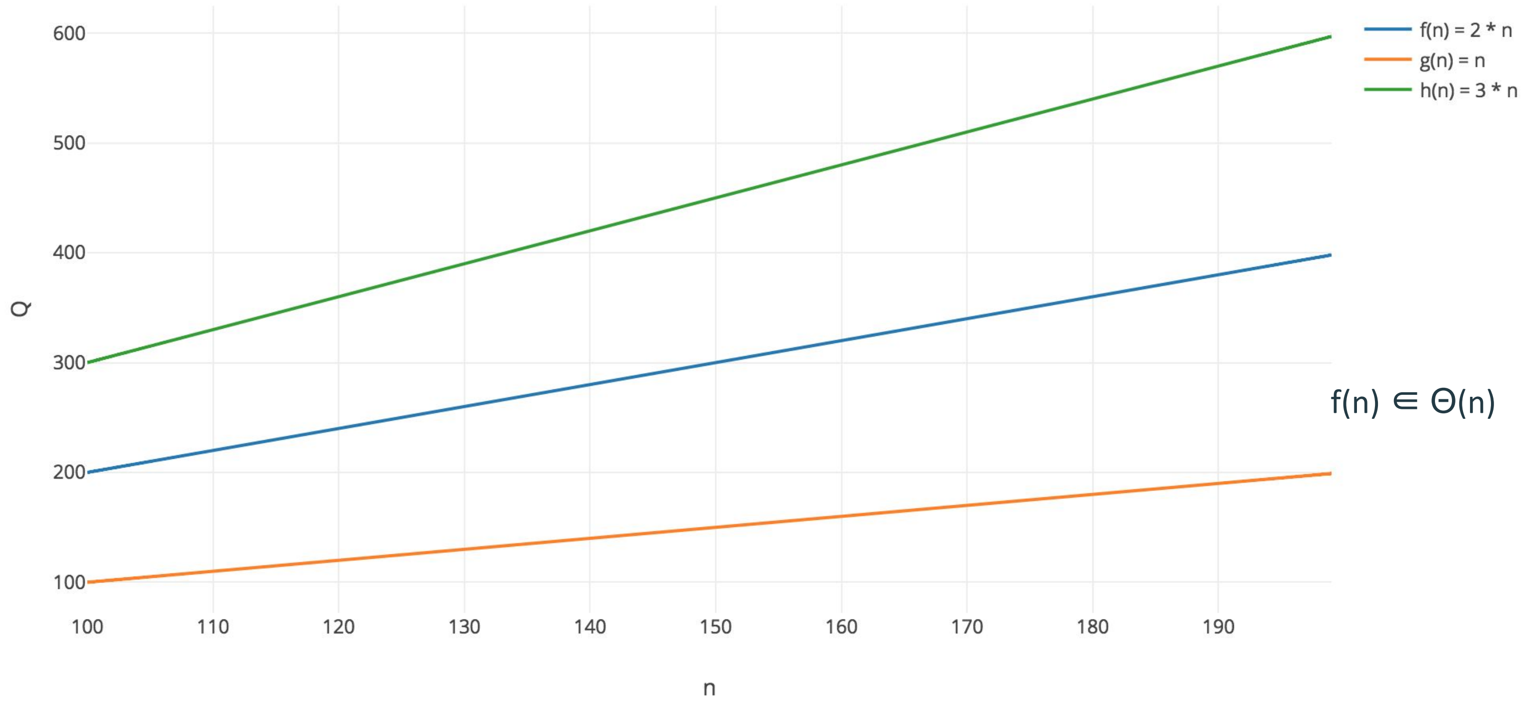
$f(n) \in \Omega(\log n)$

Ω is a lower bound.
It says “for large n, the true function **grows no slower** than a constant times this function.”



$f(n) \in O(n^2)$

O is an upper bound.
It says “for large n, the true function **grows no faster** than a constant times this function.”



$f(n) \in \Theta(n)$

Θ is both a lower bound and an upper bound.
It means both Ω and O at the same time.

Notes

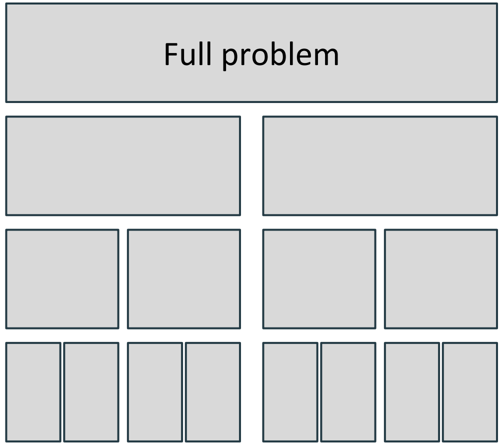
- In asymptotics problems, you can drop constant multipliers and addends. Or keep them in. It’s all the same.
- In 170, unlike in the 61 series, we often consider complexity at the bit level.

Divide-and-Conquer

Divide-and-Conquer

- A problem-solving approach that involves
  - **breaking the problem into smaller instances of the same problem**,
  - **recursively solving the smaller problems**, and finally
  - **merging all of the solutions into one**.

Classic examples: binary search, merge sort



Merge Sort

2. Recursively solving subproblems

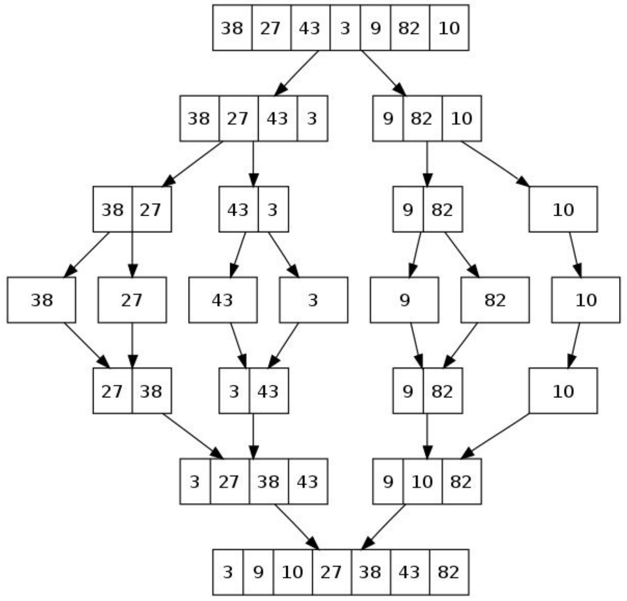
```
function mergesort(A[1...n])
    if n > 1:
        return merge(mergesort(<first half of A>),
                     mergesort(<other half of A>))
    else:
        return A
```

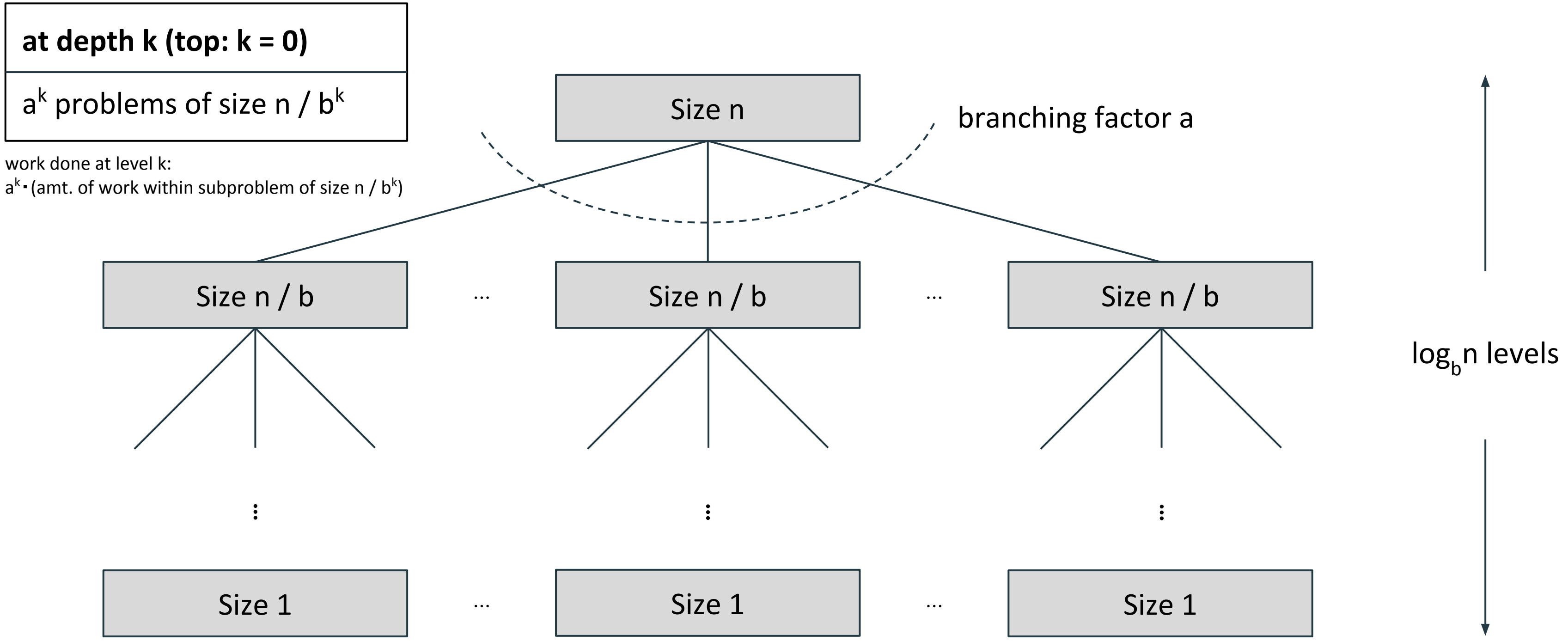
1. Breaking into subproblems

3. Merging solutions into one

Merge Sort

```
function mergesort(A[1...n])
    if n > 1:
        return merge(mergesort(<first half of A>),
                     mergesort(<other half of A>))
    else:
        return A
```




at depth k (top: k = 0)
a^k problems of size n / b^k
work done at level k:
$a^k \cdot$ (amt. of work within subproblem of size n / b^k)
Size n
branching factor a
Size n / b
…
Size n / b
…
Size n / b
⋮
⋮
⋮
Size 1
…
Size 1
…
Size 1
$\log_b n$ levels

With regard to asymptotic analysis for divide-and-conquer functions, I particularly enjoy the diagram and explanation from Chapter 2 of the textbook (reproduced above from *Algorithms* by Dasgupta et al.).

Solving Recurrence Relations

- “solve” means “identify bound(s) on the recurrence relation, i.e. on the total amount of work done”
- To start, draw the divide-and-conquer tree (of the form shown on the previous slide)! Determine from it how much work is done in total.

Potentially useful: the sum of the first n terms of a geometric series is
$a * (1 - r^n) / (1 - r)$, where a is the first term of the series and r is the common ratio

- In special cases, we can apply the **master theorem**...

Master Theorem

The Almighty Master Theorem

- A formula for ascertaining an order of growth for a divide-and-conquer algorithm
- Only works if the algorithm follows a recurrence relation of the form
    - $T(n) = a * T(n / b) + O(n^d)$
- Do read the book for the derivation!

Basically, if the recurrence relation is of the form $T(n) = a * T(n / b) + O(n^d)$, then the total work done is

$$\sum_{k=0}^{\log_b n} O(n^d) \cdot \left(\frac{a}{b^d}\right)^k$$

and the sum of the series is simply the sum of a geometric series with first term $O(n^d)$ and ratio (a / b^d). The result of this sum depends on whether $a > b^d$, $a = b^d$, or $a < b^d$ (i.e. whether $\log_b a > d$, $\log_b a = d$, or $\log_b a < d$).

Breaking Down a Recurrence Relation

A recurrence relation often takes the form T(n) = a * T(n / b) + O(f(n)).

- **a**: branching factor (each problem will give rise to *this many* subproblems)
- **n / b**: the size of each subproblem in terms of the current problem size
- **O(f(n))**: the amount of work done in each problem

Each problem corresponds to a node of the tree!

Utilizing the Master Theorem

- Say that a relation $T(n) = a * T(n / b) + O(n^d)$ has been identified.
  - If **$d > \log_b a$**, the order of growth is **$O(n^d)$**
    - decreasing geometric series, sum given by first term $O(n^d)$
  - If **$d = \log_b a$**, the order of growth is **$O(n^d \log n)$**
    - $O(n^d)$ work done at each of logn layers
  - If **$d < \log_b a$**, the order of growth is **$O(n^{\log_b a})$**
    - increasing geometric series, sum given by last term $O(n^{\log_b a})$

Complex Numbers

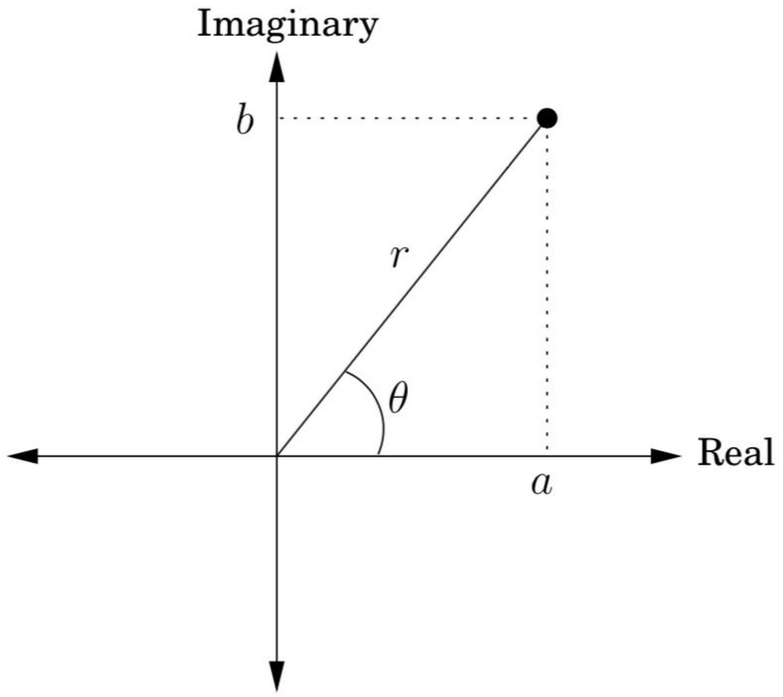
Geometric Representation

Given the complex number z in **rectangular form** a + bi,

- Compute $r = \mathrm{sqrt}(a^2 + b^2)$
- Compute θ as $\arccos(a / r)$ or $\arcsin(b / r)$
- z in polar coordinates is $r(\cos\theta + i\sin\theta) = re^{i\theta}$, denoted (r, θ)

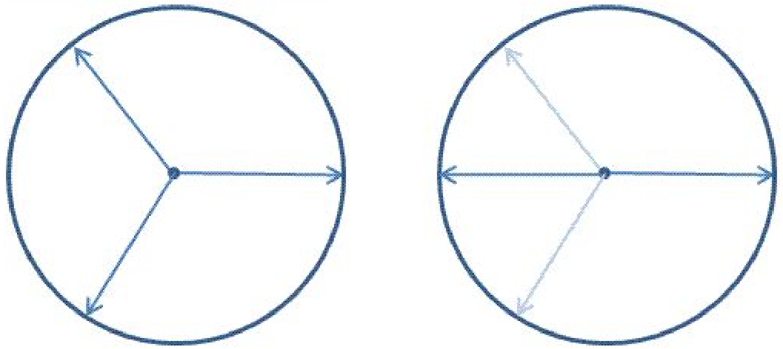
Given the complex number z in **polar coordinates** (r, θ),

- Compute $a = r\cos\theta$, $b = r\sin\theta$
- z in rectangular form is a + bi



Roots of Unity

- nth roots of unity: solutions to the equation $z^n = 1$
  - e.g. 3rd roots of unity are solutions to $z^3 = 1$, and in (r, θ) coordinates are $(1, 0)$, $(1, 2\pi / 3)$, $(1, 4\pi / 3)$
  - in standard form, the 3rd roots of unity are $1 + 0i$, $-1/2 + \sqrt{3}/2i$, $-1/2 - \sqrt{3}/2i$
- In general, the nth roots of unity are
  $z = (1, \theta)$, for θ the n multiples of $2\pi / n$
  in the range $[0, 2\pi)$



Notes

- Multiplication in polar coordinates: $(r_1, \theta_1) * (r_2, \theta_2) = (r_1r_2, \theta_1 + \theta_2)$
- The product of two numbers on the unit circle will also be on the unit circle. Furthermore, if $z = (1, \theta)$ then $z^n = (1, n\theta)$.
- See figure 2.6 in the textbook for a whole host of great information!