iOS Dev Accelerator Week 3 Day 4

- WebView
- UIScrollView
- Regex and input validation
- Swift Extensions
- Tech Thursdays : Big O

WKWebView

- “Use the WKWebView class to embed web content in your application”
- The simple workflow of a web view:
    1. add web view to the view hierarchy
    2. send it a request to load web content
- Can have a delegate that tracks loading of content, this will come in handy when we do the ‘in-app’ way of doing OAuth
- It’s sort of like a mini browser in your app, and you can customize it to not allow the users to go back or forward.
- Prior to iOS8, we had to use UIWebView, and it was horrible. It leaked memory every time you used it.

Demo

UIScrollView

Sourced From www.objc.io, issue 3, Understanding Scroll Views

`UIScrollView`

- A UIScrollView is technically a subclass of UIView, but its really just a UIView with a few simple added features.
- So to understand how scroll views work, you must first have a great understanding of how UIViews and the view hierarchy works in iOS.

Rasterization and Composition

- Rasterization and Composition combine to make up the rendering process of your iOS app.
- Rasterization is simply just a set of drawing instructions that produces an image.
- So a UIButton draws an image with a rounded rectangle and title in the center as its rosterization process.
- But these images are not drawn onto screen during the rasterization process, they are held onto by their views, in preparation for…

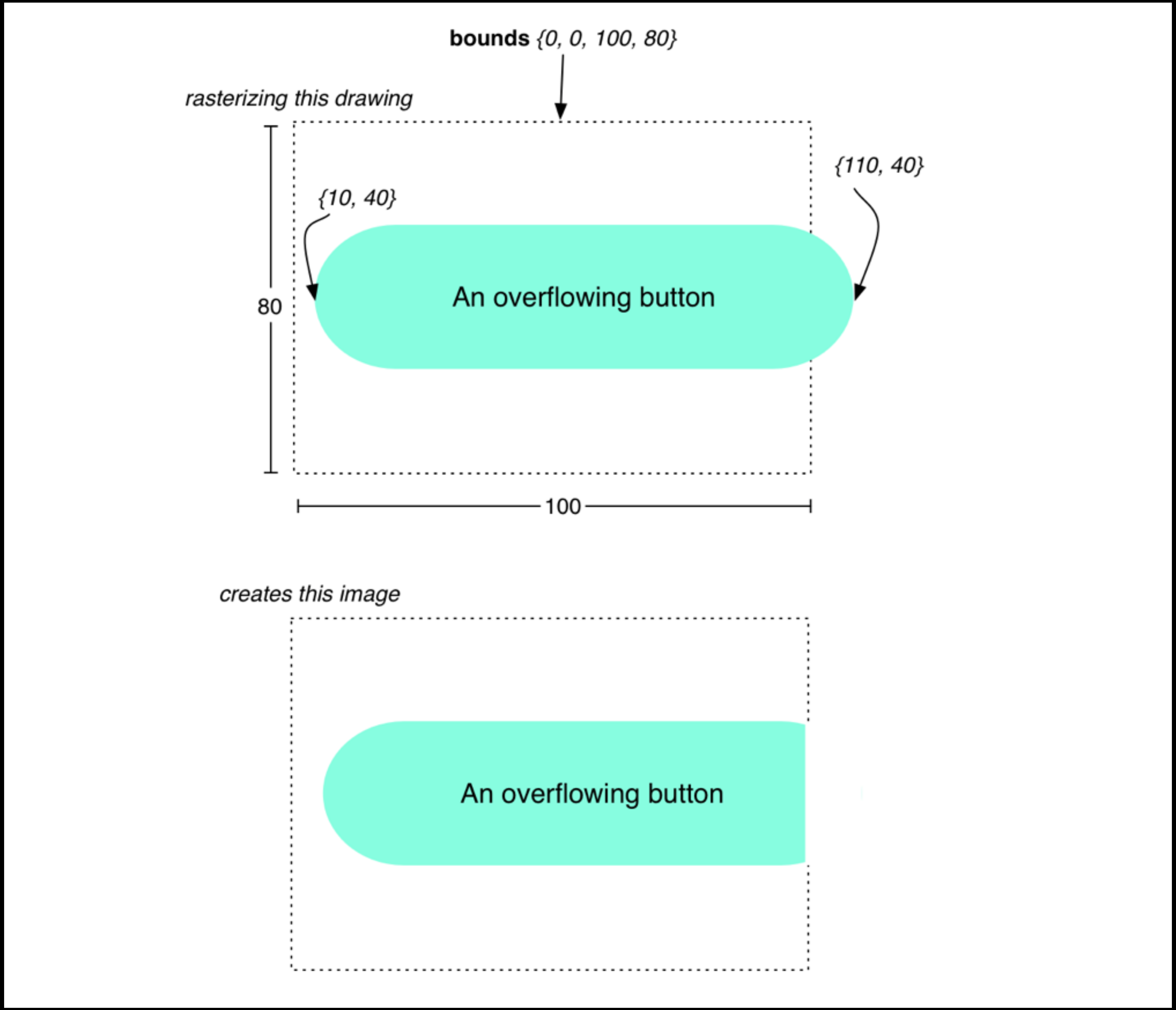
Composition

- The composition step involves all of the rosterized images being drawn on top of each other to produce one screen-sized image.
- A view’s image is composited on top of its superview’s image.
- Then the composited image is composited onto of the super-superview’s image, and so on.
- The view at the very top of the hierarchy is the window, and its composited image is what the user sees.
- So basically, you are layering independent images on top of each other to produce a final, flat image.

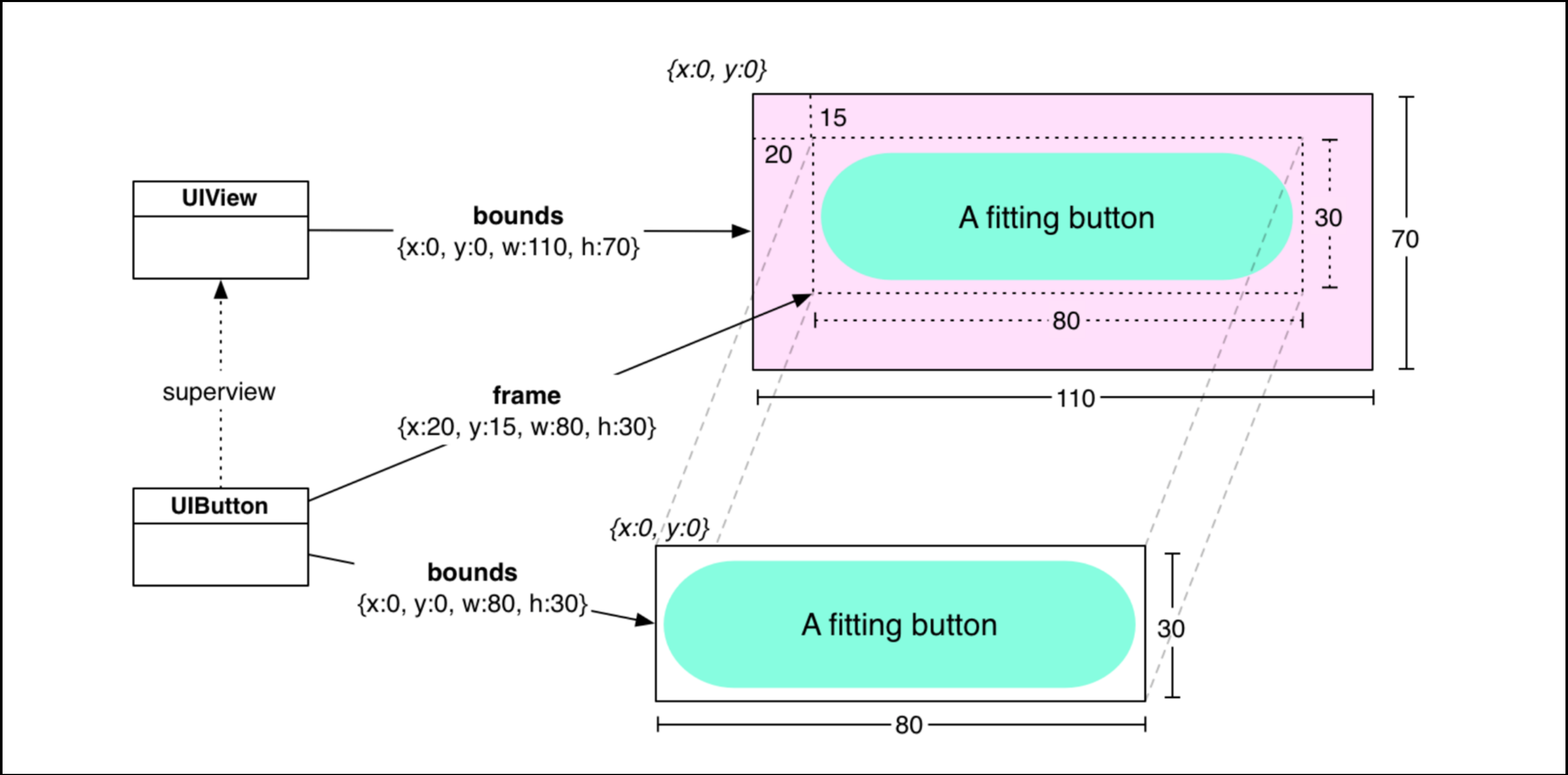
Bounds vs Frame

- So this is where bounds and frames come in to play.
- When the images are being laid out, the frame is used to figure out exactly where the image should be placed relative to its superview. So the frame is used in the composition step.
- During the rasterization step, a view doesn't care about its frame. The only thing it cares about is drawing its own content. Anything inside of its current bounds rect gets drawn onto the image. Anything that is outside of the bounds is discarded.
- Lets look at some pics!

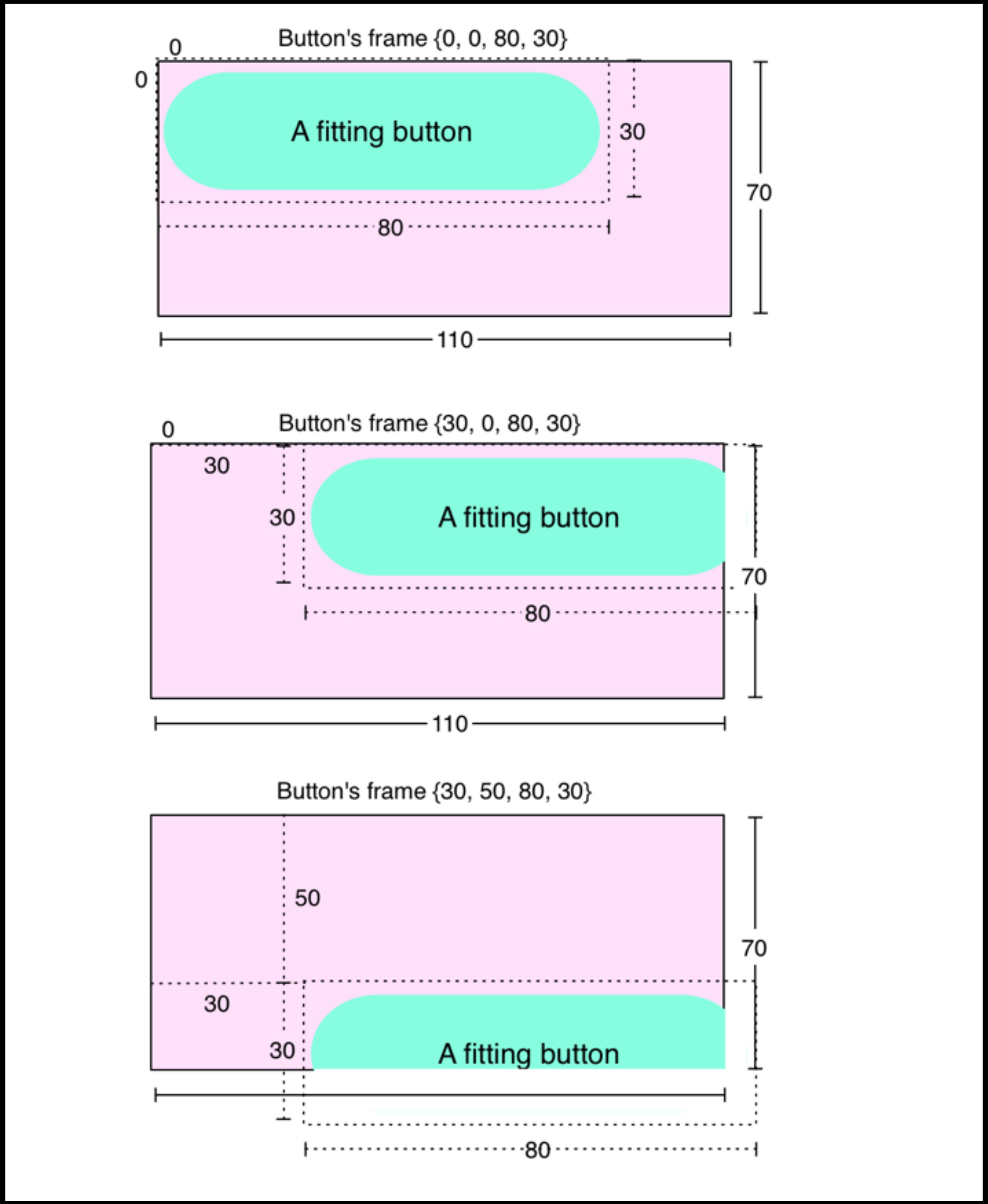
Bounds & rasterizing



Frame & composition



More Frames!

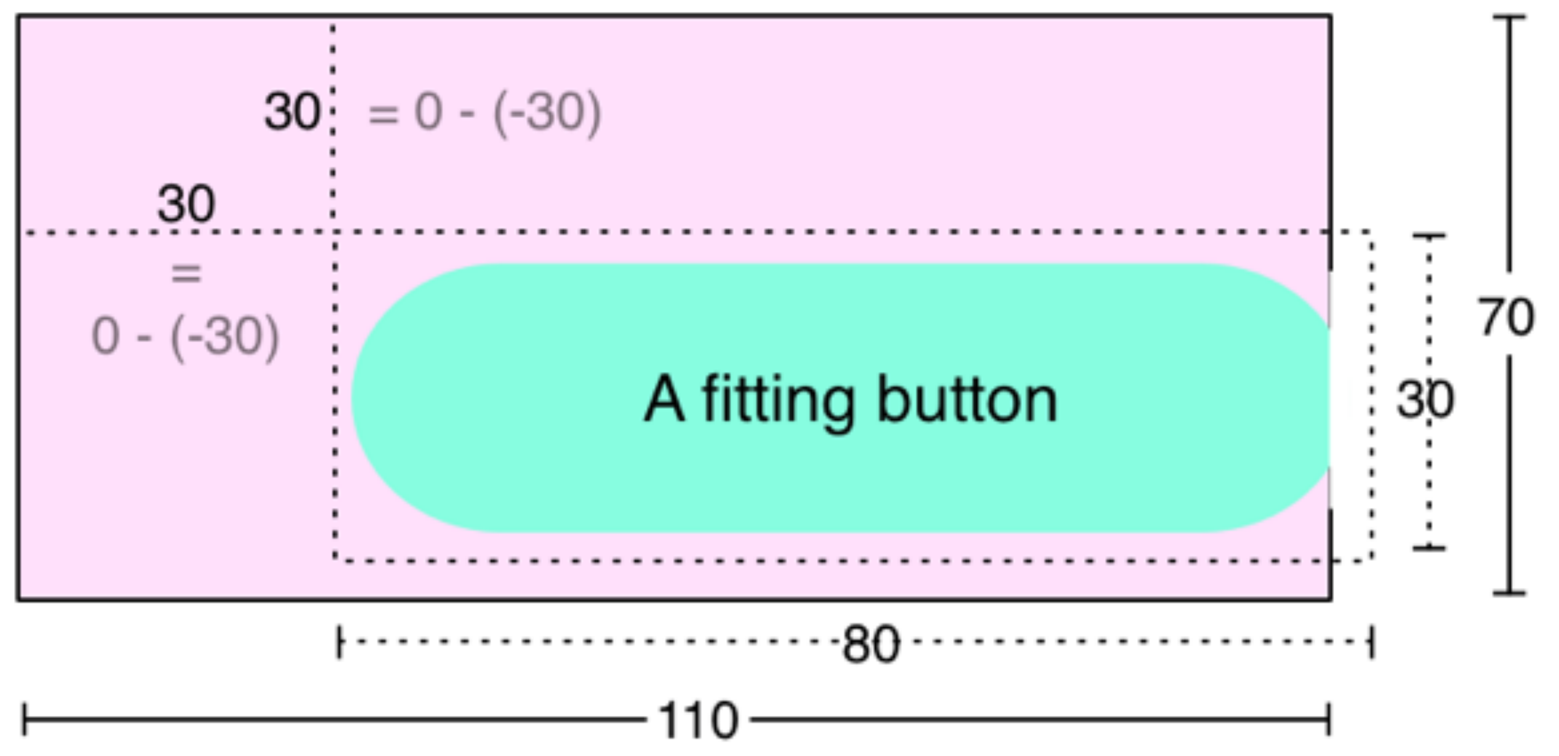


Button's frame {0, 0, 80, 30}
0
0
A fitting button
30
80
70
110
Button's frame {30, 0, 80, 30}
0
30
30
A fitting button
70
80
110
Button's frame {30, 50, 80, 30}
50
70
30
30
A fitting button
80

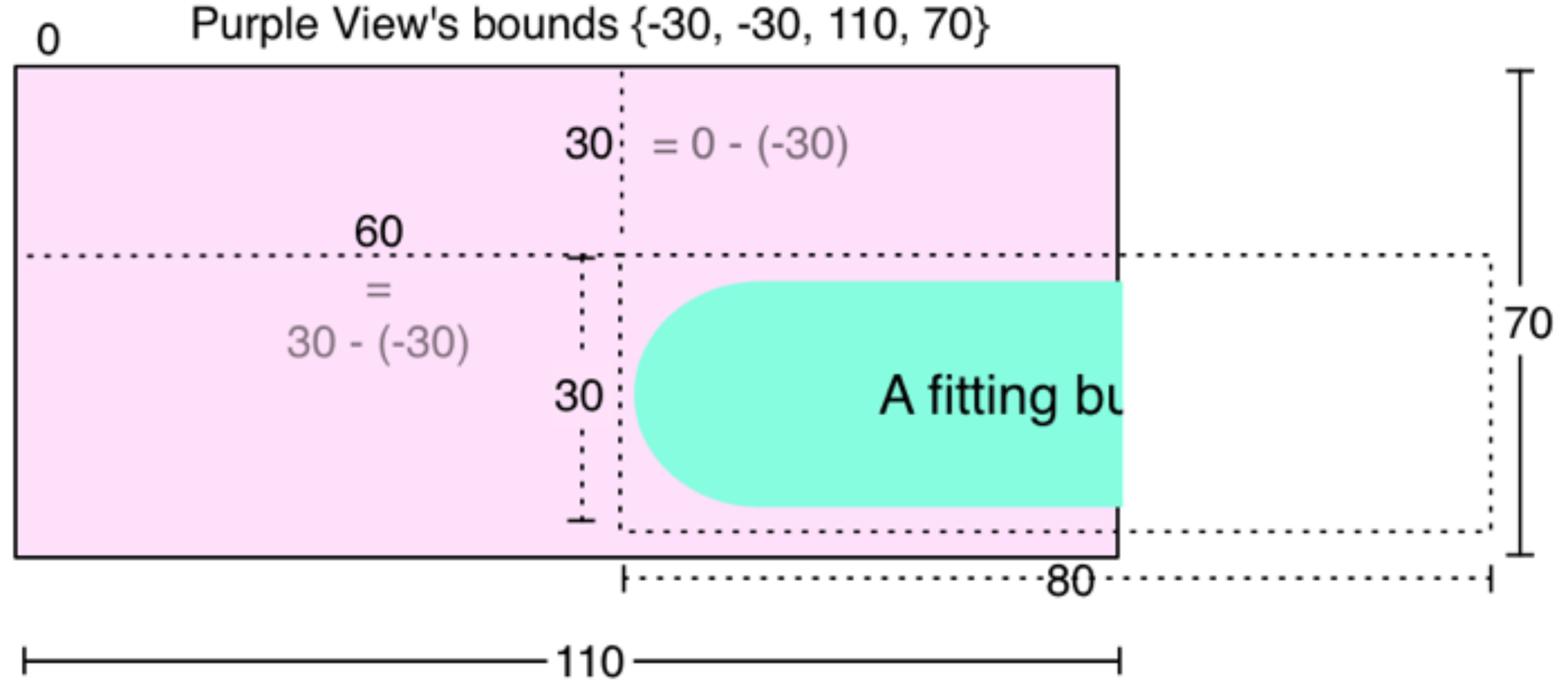
ScrollView

- So what does this have to do with scroll views? Everything.
- When you scroll a scroll view, the offset value changes. Under the hood, this is just changing the origin of the bounds of the scroll view.
- You could accomplish the same effect by changing the frames of all the scroll views subviews as it scrolls, but you would have to loop through every single subview every time the scroll view scrolls, which would suck.
- Visuals!

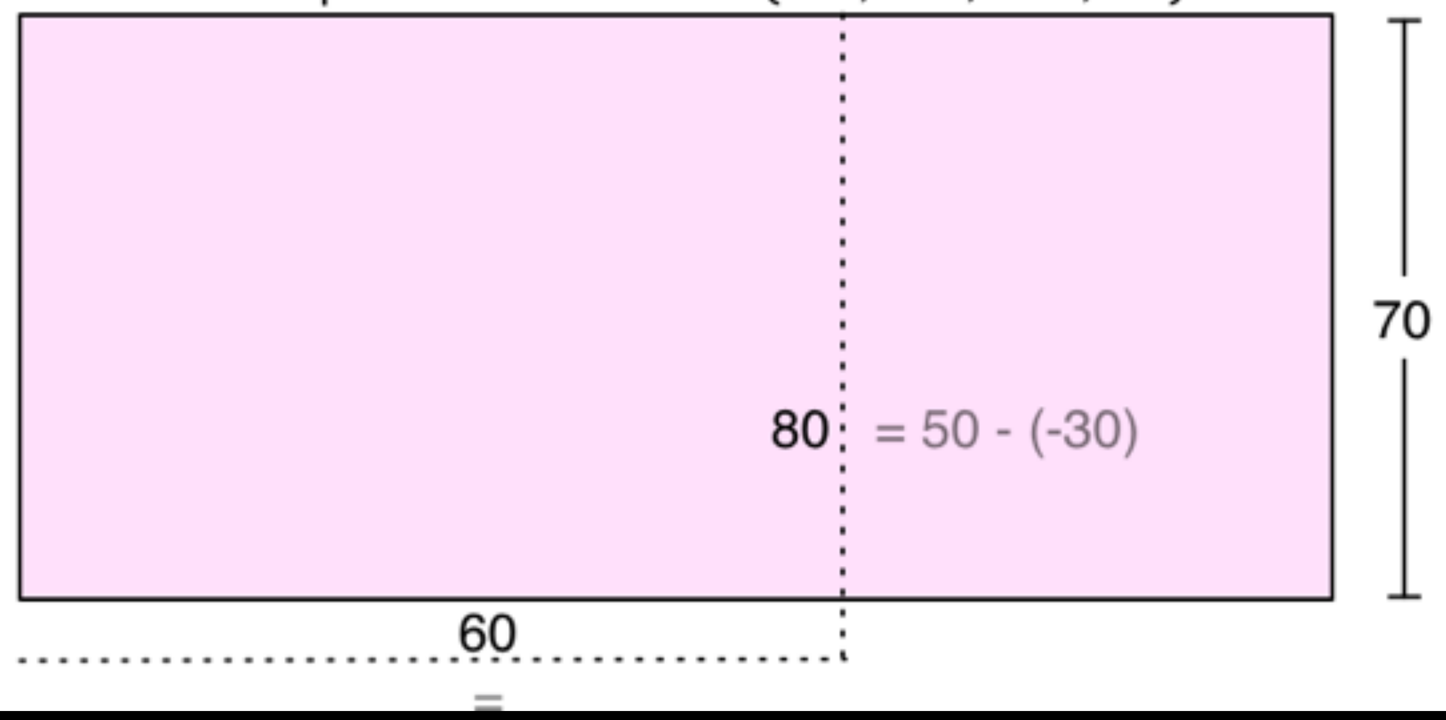
Button's frame {0, 0, 80, 30}
Purple View's bounds {-30, -30, 110, 70}



Button's frame {30, 0, 80, 30}
Purple View's bounds {-30, -30, 110, 70}



Button's frame {30, 50, 80, 30}
Purple View's bounds {-30, -30, 110, 70}



Scroll View’s contentSize

- A scroll view’s contentSize is the just the entire scrollable area.
- It doesn't effect the frame or bounds of a scroll view.
- By default it is set to 0,0
- When the content size is larger than the scroll view’s bounds, the user is allowed to scroll.
- Think of the bounds of the scroll view as a window into the scrollable area defined by the content size.

Scroll View’s contentOffset

- “The point at which the origin of the content view is offset from the origin of the scroll view”
- The contentOffset is just how much the scroll view has scrolled.

Scroll View’s contentOffset

- “The point at which the origin of the content view is offset from the origin of the scroll view”
- The contentOffset is just how much the scroll view has scrolled.
- In addition to panning, you can set it programmatically with setContentOffset:animated:

Scroll View’s delegate

- scroll view has an awesome delegate that is constantly notified of the state of the scroll view:
    - scrollViewDidScroll:
    - scrollViewWillBeginDragging:
    - scrollViewWillEndDragging:withVelocity:targetContentOffset:
    - scrollViewDidZoom:
    - etc

Demo

Regex and Input Validation

Input Validation

- “An app sometimes cannot accept the strings entered in textfields and text views without validating the value first”
- In our Github app, we cant have spaces in user’s search querys. We also don’t want them entering in symbols like = or $.
- So how can we ensure our users input doesn't break our app?

Delegate methods on input views

- The go to methods for text validation are:
  - For UITextField = textFieldShoudEndEditing:
  - For UITextView = textViewShouldEndEditing:
  - For UISearchBar = searchBar:shouldChangeTextInRange:
- These methods are called just before the textview/field is about to resign first responder.
- Returning false prevents that from happening. So return false when the text isn't valid. Also pop up an alert view or show some indicator explaining why its invalid.
- So how do we check if the text is valid?

Regex

- Regular Expressions are a pattern-matching standard for text validation.
- The regular expression is a pattern that you compare with the text you are validating.
- Regex’s can be used for finding patterns, replacing text, and extracting substrings in strings.

The simplest Regex

- The most simple regex is just a string. Like “seahawks”.
- The regex “seahawks” will find a match on the string “go seahawks”
- The regex “sea hawks” will not find a match on the string “go seahawks”
- In these two examples, we are only using literal characters. So its basically just running a find operation looking for our regex string.
- There are special reserved characters we can use in regex to make it much more powerful

+ and *

- + is used to match the previous character 1 or more times
- so sea+ will match sea and seaaaaaaaaa and seaaaa
- * works the same way, except it matches 0 or more times
- so sea* matches seaaaaaa, sea, and also se

`(Capturing parens)`

- Parentheses are used to create a capturing group.
- Capturing groups capture the text matched by the regex inside the parens and put them into a numbered group that can be referenced.

? optional

- The question mark makes the preceding token in the regular expression optional.
- So seahawks? matches both seahawks and seahawk
- You can use grouping to make groups optional
- So sea(hawks)? matches seahawks and sea

[Character Classes]

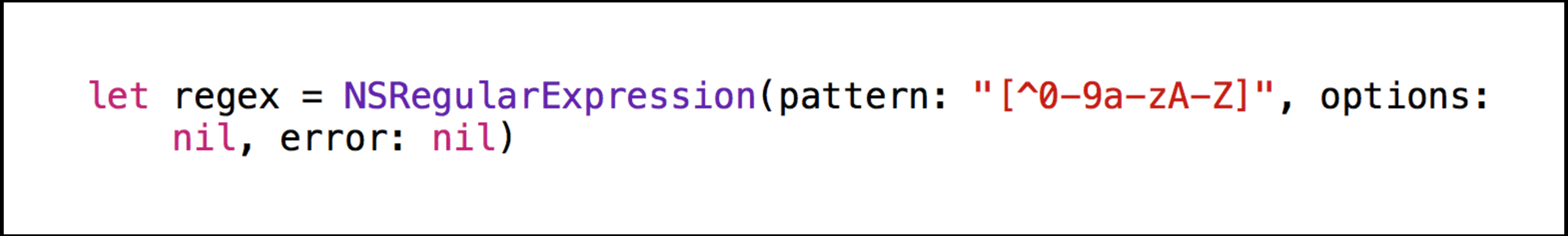
- With a character class, sometimes called character set, you can have the regex only match one of several characters.
- If you want to match a t or d you will use [td]
- so foo[td] will match foot and food
- You can use a hypen inside the character classes to specify ranges
- [0-9] matches any single number
- You can combine ranges [0-9a-zA-Z] will match any regular literal character

[Negating Character Classes]

- Adding a caret after the opening square bracket negates the character class.
- This makes it so the character class matches any character that is NOT in the character class.
- So [^0-9a-zA-Z] will match any non regular literal character.
- Hey I think we can use that in our app!

Regex and iOS

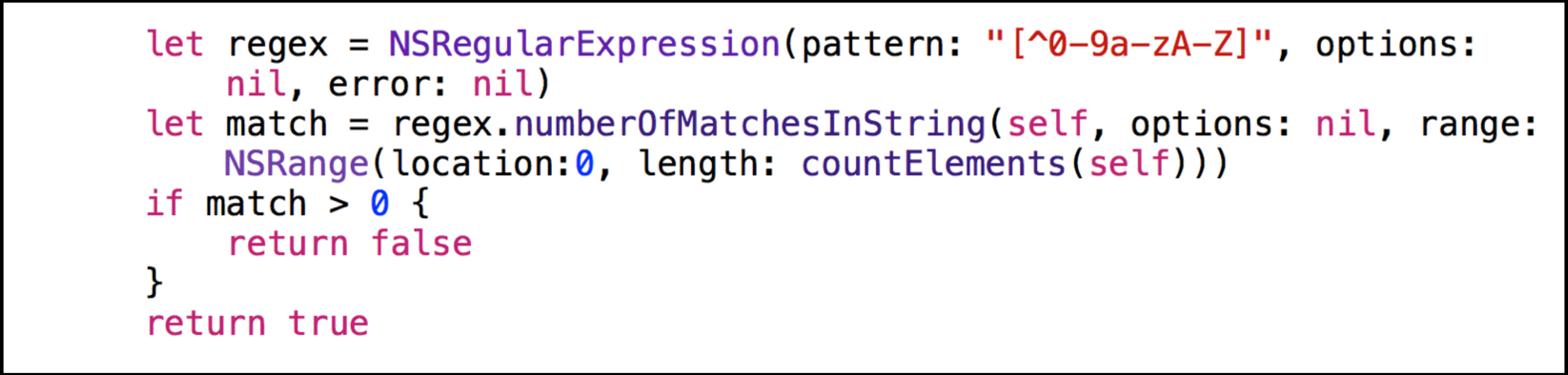
- You can use the NSRegularExpression class to natively use regex in your app.
- You instantiate an instance of NSRegularExpression with your regex pattern, options, and an error pointer:



```
let regex = NSRegularExpression(pattern: "[^0-9a-zA-Z]", options:
    nil, error: nil)
```

Search for matches

- NSRegularExpression has methods for returning the total count of matches, enumerating through each match, returning an array of matches, returning the first match, and the range of the first match.
- In our app we can just use the number of matches method, and if its greater than 0 we know the user typed in something invalid:



```
let regex = NSRegularExpression(pattern: "[^0-9a-zA-Z]", options:
    nil, error: nil)
let match = regex.numberOfMatchesInString(self, options: nil, range:
    NSRange(location:0, length: countElements(self)))
if match > 0 {
    return false
}
return true
```

Demo

Swift Extensions

Extensions

- Extensions add new functionality to pre-existing classes, structs, or enums.
- You can even do this on types you do not have access to the source code.
- Similar to categories in Objective-C, except extensions dont have names.

Things you can add with extensions:

- computed properties
- instance methods
- type methods
- new inits
- subscripts
- nested types
- make an existing type conform to a protocol

Extension Syntax

```
extension SomeType {
    // new functionality to add to SomeType goes
        here
}
```

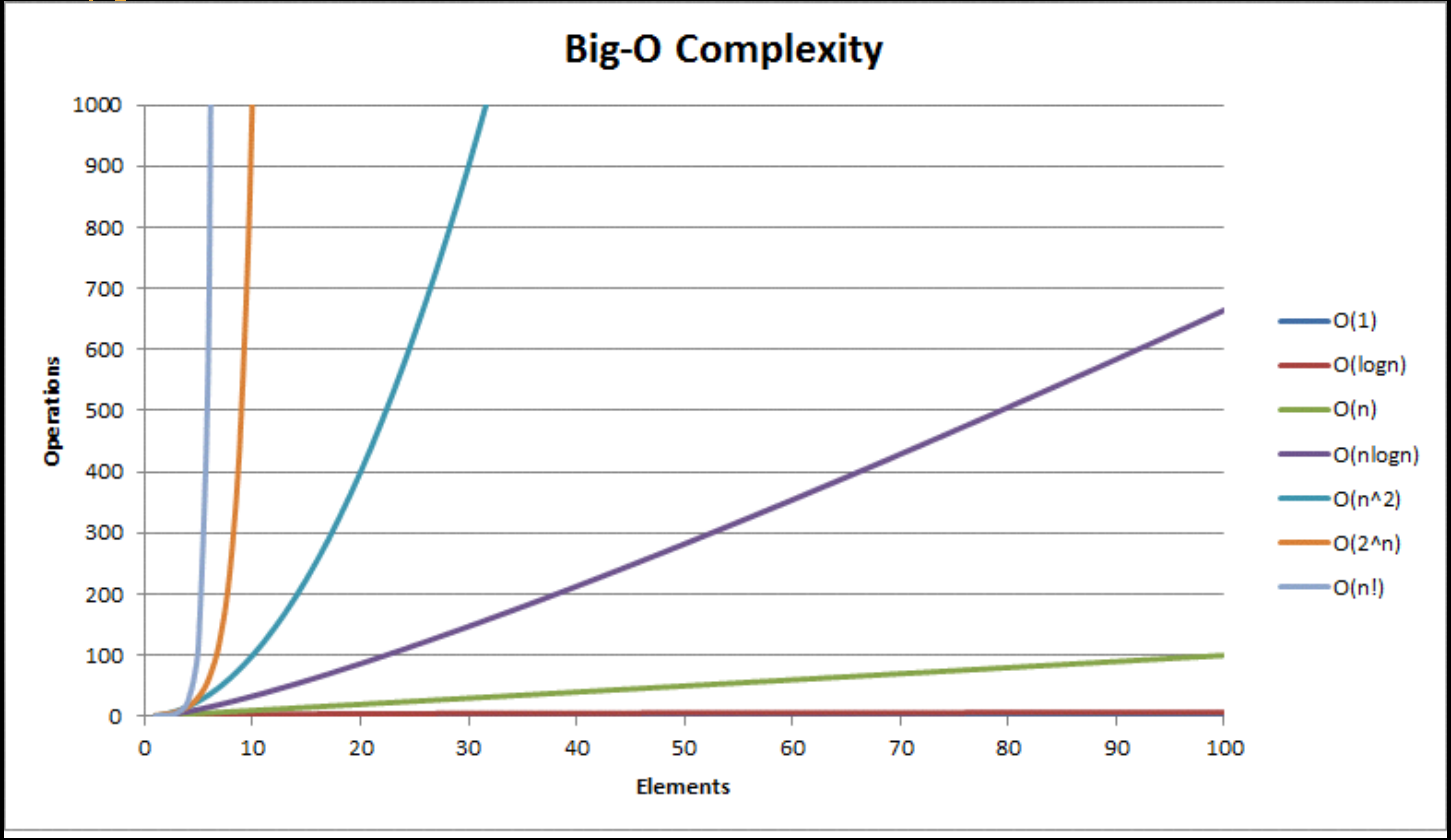
Demo

Tech Thursday
Big O Notation

Big O notation

- In computer science, big o notation is used to measure the efficiency of algorithms.
- The O stands for order, because because the growth rate of an algorithm is also refereed to as the order of the algorithm.
- In most big o notations, you will see the variable N. N refers to the size of the data set on which the algorithm is being performed. So if we are searching through an array of 10 strings, N == 10.
- Big-O usually refers to the worst case scenario.

Big O notation



O(1) – Constant Time

- The running time of any ‘simple’ operation is considered constant time, or O(1). Things like setting properties, checking bools, simple math equations are constant time.
- A constant time efficiency is awesome!

O(N) - Linear

- An example of a O(N) algorithm is a for loop.
- The loop executes N times. Lets say you are looping through an array of 10 Ints looking for a specific value. So N is 10 here. At worst case, the value you are looking for is at the end of the array, and it took you N (or 10) operations to find it.
- The operation inside of the for loop is a constant time value check. So you might think the Big-O notation of this algorithm is O(N) * O(1). But when you are working with Big-O, you can drop constant times since they are trivial, so the Big-O of a regular for loop is just O(N)

O(N^2^2)

- O(N^2) refers to any algorithm whose Big-O is the square of the input data set.
- An example of this would be a nested for loop.
- Lets say we are searching for duplicates in an array of Strings. For each string in the array, we search through every other string and check if the values are the same. So if we have 7 Strings, this operation will run 49 times, or 7x7, or 7^2.

O(log N)

- A good rule of thumb: if your algorithm cuts the data set in half for in step of the algo, you are probably working in O(log N).
- Looking at our graph, we can see algorithms with this notation peaks at the beginning and then quickly levels out as the data size increases.
- O(log N) is great! Even when you double your N, the worst case time to run the algorithm only increases by a small amount. The classic example of this is a Binary Search Tree.