iOS Development Accelerator

- Animation*
- Binary Search Tree

*Sourced from Motion Design for iOS by Mike Rundle

Animating your app

- Thanks to iOS7's new design principles, there are really only 3 reasons to use animations in your app:
  - Transition: Smoothly animating from one visual state to the next helps the user understand what the app is doing and why.
  - Focus: Direct the user's attention to a specific aspect of the interface.
  - Delight: Make things look awesome and appealing.

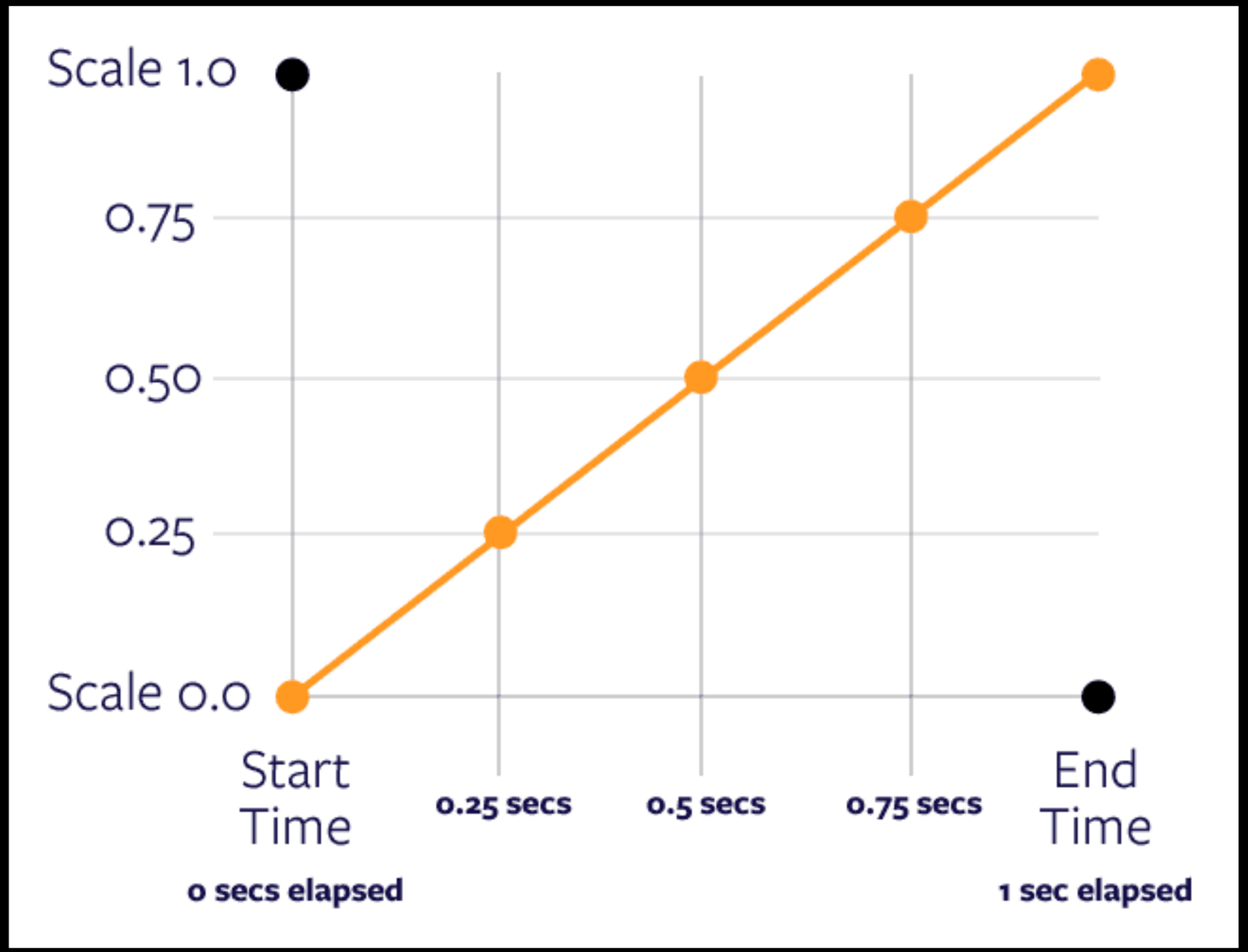
Properties that can be animated

- Position: Changing the X and Y values of a views origin
- Opacity: Changing the alpha from in the range of 0 to 1
- Scale:Increase or decrease the size of a view. 1 represents its normal size.
- Color: transition one color value to another.
- Rotation: Rotate a view by radians.
- 3D Transform: Rotate the third dimension of a view

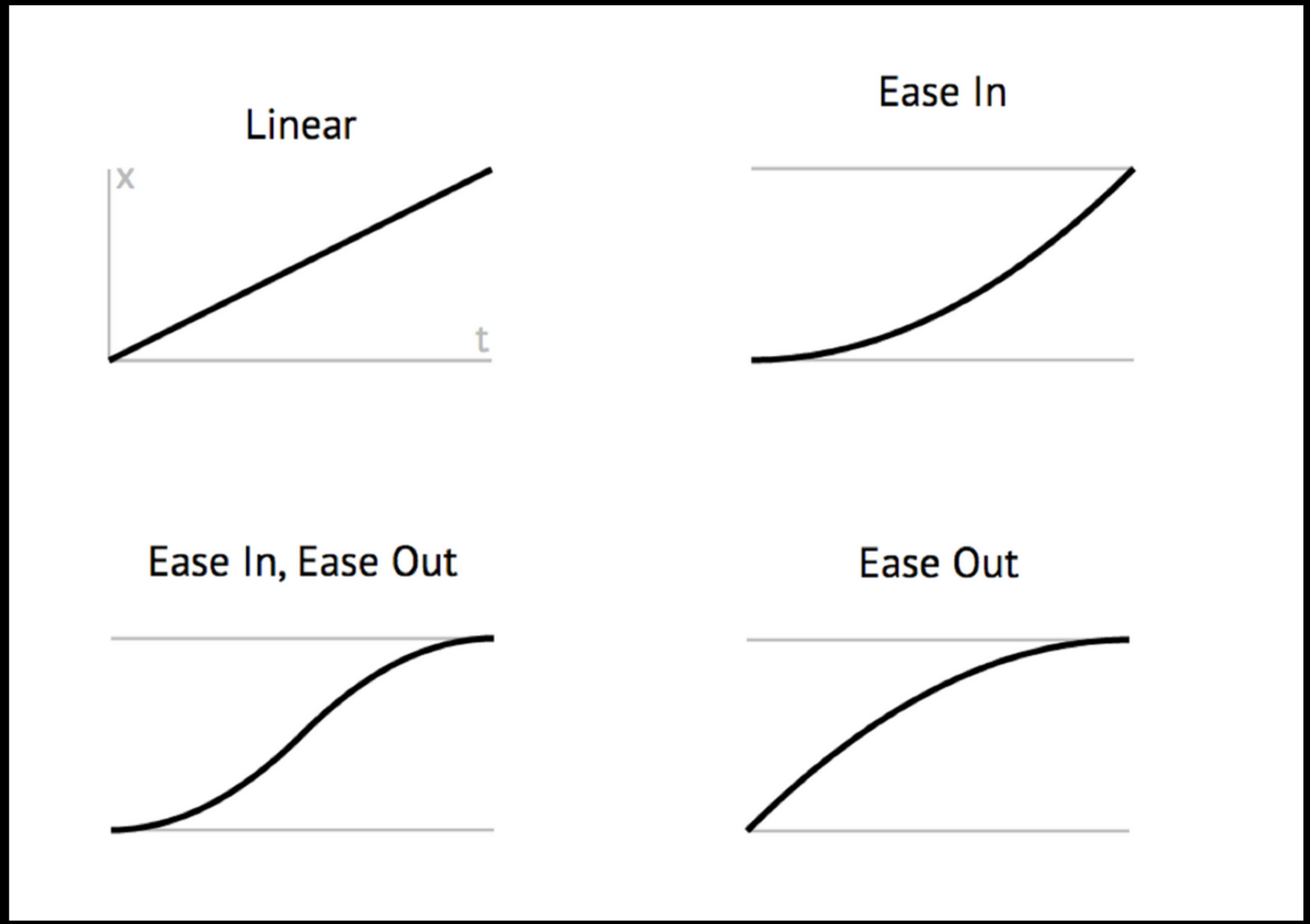
Planning out your animations

1. What are the initial properties of the item?
2. What are the final properties of the item?
3. How long should the animation take?
4. Whats happening to this item while it is animating?
5. What will happen once this item is done animating?

Timing is everything



Standard animation curves



Spring animations

- “The type of a motion being used to generate a nice, springy-feeling animation is typically modeled after a damped harmonic oscillation.
- The key properties of a spring based of a spring’s motion are:
  - Mass: the weight or heft of the object
  - Stiffness: how difficult it is to stretch out the spring
  - Damping: the restrictive force or friction pushing against the object

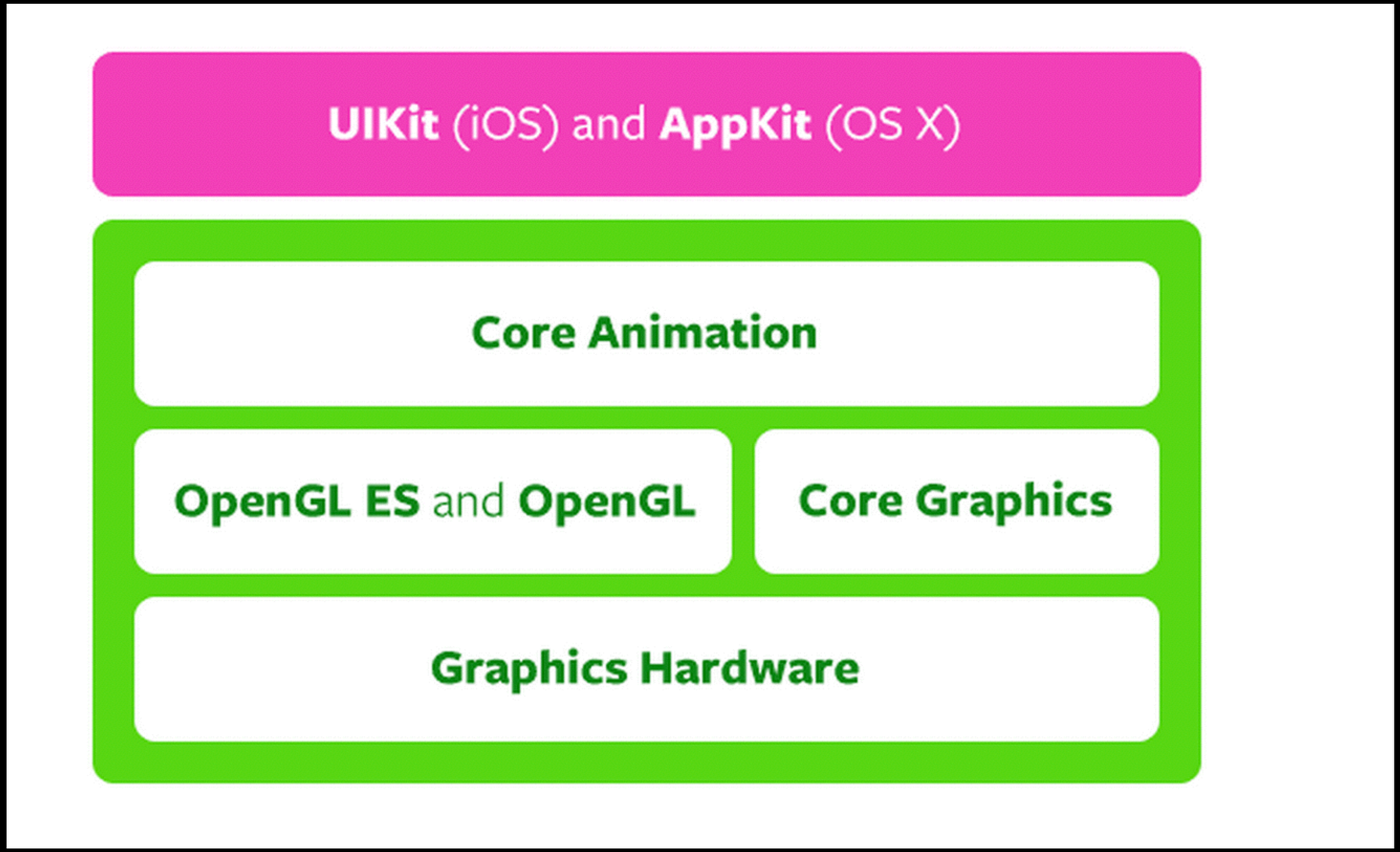
Spring animations

- “The type of a motion being used to generate a nice, springy-feeling animation is typically modeled after a damped harmonic oscillation.
- The key properties of a spring based of a spring’s motion are:
  - Mass: the weight or heft of the object
  - Stiffness: how difficult it is to stretch out the spring
  - Damping: the restrictive force or friction pushing against the object

Core Animation

- “Core animation is an animation and graphics compositing framework made for speed and efficiency”
- Despite it having the word animation, Core Animation is actually in charge of all rendering on screen. Not just animations.
- Core Animation uses CALayer objects as its main unit of work. UIView objects are just thing wrappers for CALayer’s.
- Layers can be arranged in a hierarchy just like UIView’s, in fact you can build your entire interface using just layers if you wanted.

Core Animation



Built-in Animation Techniques

- Apple provides a 2 different ways to create animations:
- Adding CAAnimation and its subclasses to a layer (Advanced)
- Simple block/closure based system using class methods on UIView (Fairly Easy)
- Begin/End animation methods, where every animatable property you change after begin is called and before end is animated.

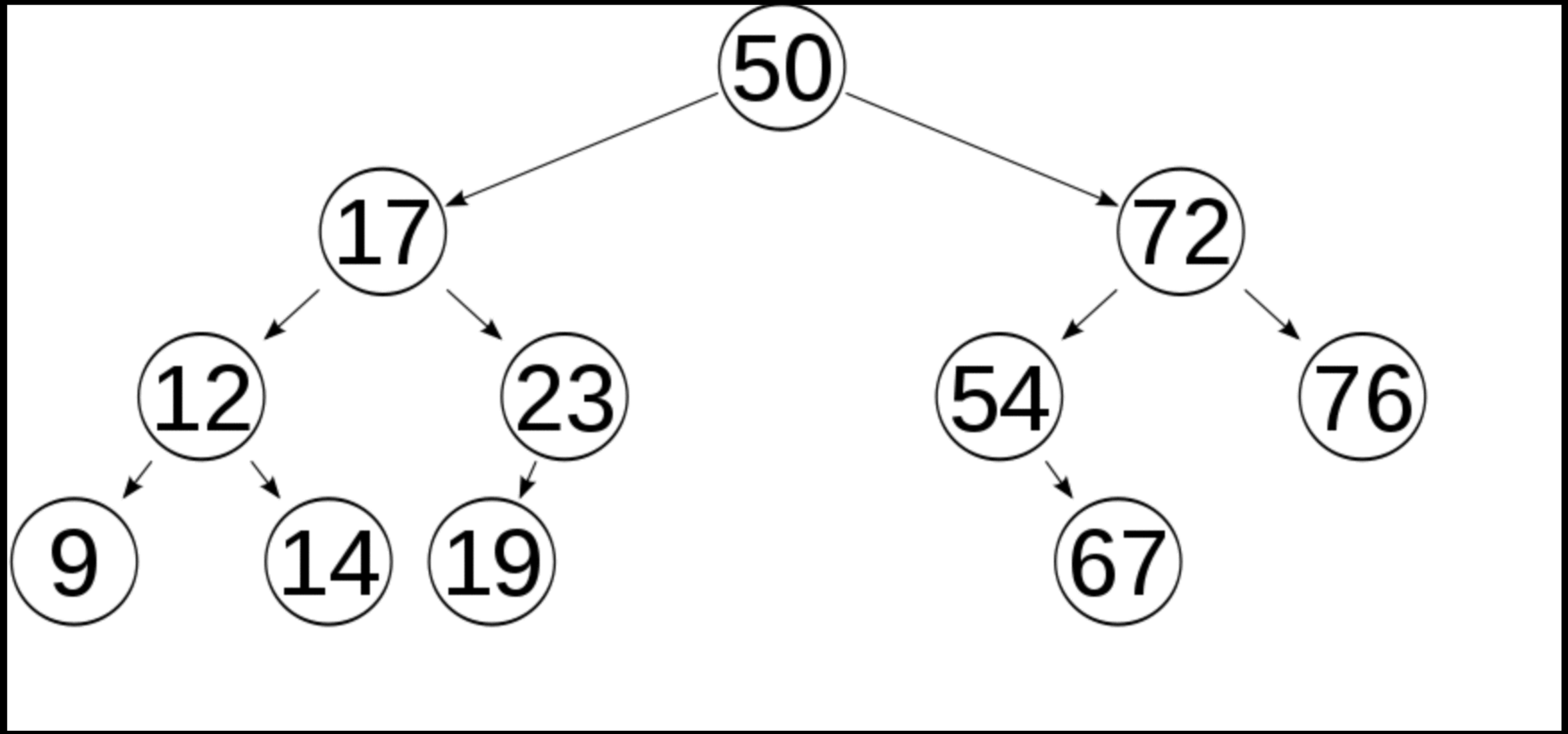
Block/Closure Based

- There are a couple different class methods for the block based animation system.
- One does not have a completion block, all the others do.
- Two of them provide a parameter for a delay before the animation fires.
- One provides an options parameters where you specify the curve type to use (ease in, ease out, etc)
- One provides everything above, plus spring dampening!

Binary Search Tree

- A Binary Search Tree is a node based data structure that satisfies two key requirements:
  - the value in any node is larger than the values in the nodes left sub tree, and less than the values in the nodes of the right sub tree.
  - A node has at most two children nodes.
- The main advantage of a BST is that it remains ordered as you add objects to it, so search times are very fast compared to other data structures.

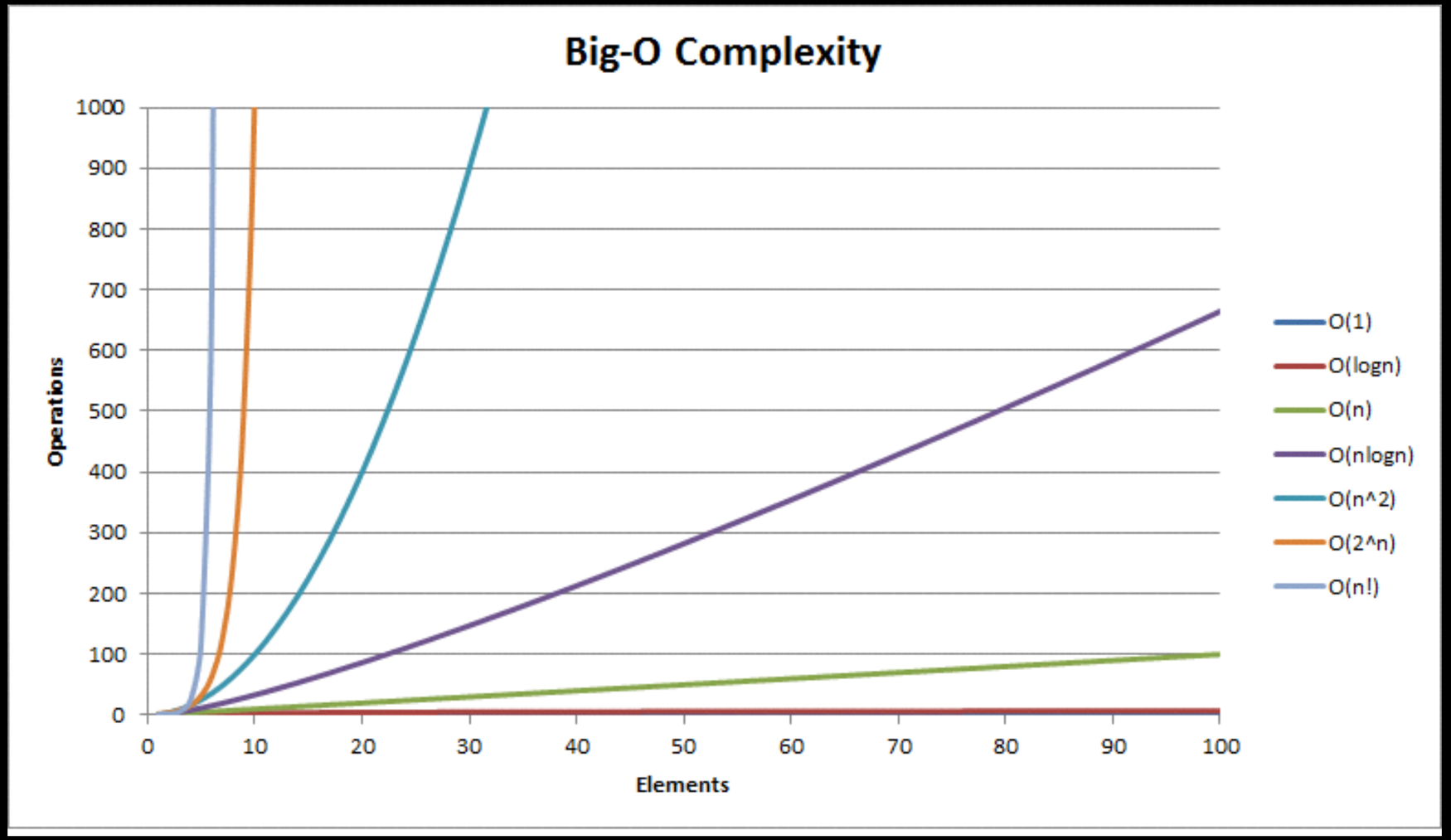
Binary Search Tree



BST & Big-O

- The Big-O of searching through a BST is O(log n)
- It is log n because the search size gets cut in half each step you take through the tree.
- Any algorithm that cuts its search space in half in each step has a Big-O of O(log n)

BST & Big-O


Big-O Complexity
Operations
1000
900
800
700
600
500
400
300
200
100
0
0
10
20
30
40
50
60
70
80
90
100
Elements
O(1)
O(logn)
O(n)
O(nlogn)
O(n^2)
O(2^n)
O(n!)

BST & Big-O

- The Big-O of searching through a BST is O(log n)
- It is log n because the search size gets cut in half each step you take through the tree.
- Any algorithm that cuts its search space in half in each step has a Big-O of O(log n)