iOS Dev Accelerator
Week 5 Day 4

- Local Notifications
- MKMapOverLay
- PFUser
- Big O
- Linked Lists

Local & Push Notifications

Notifications

- “Local and push notifications are ways for an application that isn’t running in the foreground to let its users know it has information for them”
- Local and push look and sound the same.
- Can be displayed as an alert message and/or badge icon.
- Can play a sound.
- Not related to NSNotificationCenter!

Push vs Local

- Local notifications are scheduled by an app and delivered on the same app. Everything is done locally.
- Push notifications are sent by your server to the Apple Notification service, which pushes it to the device(s).
- While they appear the exact same to the user, they appear different to your app.

Launching with Notifications

- If your app is in the foreground, you will receive either application:didRecieveRemoteNotification: or application:didRecieveLocalNotification: in the app delegate.
- In that case, the system does not automatically show a popup or indication of a notification, **it is completely up to you notify the user of the event.**
- If your app is not in the foreground or not running, the system will display either a notification or a badge by your app icon. When the user launches the app, you need to check the launch dictionary for that information.

Local notifications

- Suited for time based or location based behaviors.
- Local notifications are instances of UILocalNotification
- 3 Properties:
  - Scheduled Time: Known as the fire date. Can set the time zone as well. Can also request it be rescheduled to repeat at regular intervals.
  - Notification Type: The alert message, the title of action button, the icon badge number, and a sound to play.
  - Custom Data: dictionary of custom data
- Each app limited to 64 scheduled local notifications.

Local notifications work flow

1. Create an instance of UILocalNotification
2. Set the fireDate property.
3. Set the alertBody (message) property, alertAction property(title of button or slider), applicationIconBadgeNumber property, and soundName property.
4. Optionally set any custom data you want with userInfo property
5. Schedule the delivery by calling scheduleLocalNotification: on UIApplication. Or you can fire it immediately by calling presentLocalNotificationNow:

- You can cancel local notifications with cancelLocalNotifcation: or cancel all with cancelAllLocalNotifications:

Reacting to a Notification when your app is not in the foreground.

1. The system presents the notification, displaying the alert, badge, and/or playing the sound.
2. As a result, the user taps the action button of the alert, or taps the applications icon with the badge.
3. If the user tapped the action button, the app is launched and the app calls its delegate's application:DidFinishLaunchingWithOptions: method. It passes in the notification payload in the info dictionary.

Reacting to a Notification when your app is in the foreground.

1. The application calls its delegate application:didReceiveRemoteNotification: method or application:didReceiveLocalNotification method and passes in the notification payload.
2. Remember its up to you notify the user of the event, the system wont play any sounds or show the popup alert.

iOS 8 and notifications

- New with iOS 8, you must call this code before your app can be allowed to use notifications:

```
if ([application respondsToSelector:@selector(registerUserNotificationSettings:)]) {
  UIUserNotificationSettings *settings = [UIUserNotificationSettings
    settingsForTypes:(UIUserNotificationTypeAlert | UIUserNotificationTypeBadge |
    UIUserNotificationTypeSound) categories:nil];
  [application registerUserNotificationSettings:settings];
}
```

Demo

Map Overlays

Map Overlays

- “Overlays offer a way to layer content over an arbitrary region of the map”
- Overlays are usually defined by multiple coordinates, which is different from the single point of an annotation.
- Overlays can be contiguous or noncontiguous lines, rectangles, circles, and other shapes.
- Those shapes can then be filled and stroked with color.

Overlay overview

- Getting an overlay onscreen involves two different objects working together:
  - An overlay object, which is an instance of a class that conforms to the MKOverlay protocol. This object manages the data points of the overlay.
  - An overlay renderer, which is a subclass of MKOverlayRenderer that does the actual drawing of the overlay onto the map.

Overlay workflow

1. Create an overlay object and add it to the map view
2. return an overlay renderer in the delegate method mapView:rendererForOverlay:

Overlay objects

- Overlay objects are typically small data objects that simply store the points that define what the overlay should represent and other attributes.
- MapKit defines several concrete objects you can use for your overlay objects if you want to display a standard shaped overlay.
- Otherwise, you can use any class as an overlay object as long as it conforms to the MKOverlay protocol.
- The map view keeps a reference to all overlay objects and uses the data contained in those objects to figure out when it should be displaying those overlays.

Overlay objects

- Concrete objects: MKCircle, MKPolygon, MKPolyline
- MKTileOverlay: use if your overlay can be represented by a series of bitmap tiles
- Subclass MKShape or MKMultiPoint for custom shapes
- Use your own objects with the MKOverlay protocol

OverlayRenderer

- MapKit provides many standard overlay renderers for standard shapes.
- You don’t add the renderer directly to the map, instead the delegate object provides an overlay renderer when the map view asks for one.

OverlayRenderer Objects

- Standard shapes: MKCircleRenderer, MKPolygonRenderer, MKPolylineRenderer.
- Tiled: MKTileOverlayRenderer
- Custom shapes : MKOverlayPathRenderer
- For the most customization, subclass MKOverlayRenderer and implement your custom drawing code.

Demo

PFUser

Users

- So far in our lectures, we have uploaded data to parse without bothering with the notion of users
- Adding users to your app allows us to implement user owned resources
- For example, on Facebook, you can only view your own posts and people you are friends with.

PFUser

- Parse provides a class called PFUser that represents a user
- PFUser is a subclass of PFObject, so it comes with all of that functionality, plus more

Signing up

- Signing up a user can can be done in two ways:
    1. Writing the code to signup manually
    2. Using Parse's built in PFSignUpViewController (contained in the ParseUI framework, you will need to specifically import that)

Manual Signup

```
- (void)myMethod {
    PFUser *user = [PFUser user];
    user.username = @"my name";
    user.password = @"my pass";
    user.email = @"email@example.com";

    // other fields can be set just like with PFObject
    user[@"phone"] = @"415-392-0202";

    [user signUpInBackgroundWithBlock:^(BOOL succeeded, NSError *error) {
      if (!error) {    // Hooray! Let them use the app now.
      } else {    NSString *errorString = [error userInfo][@"error"];    // Show the errorString somewhere and
let the user try again.
      }
    }];
}
```

PFSignUpViewController

- Instantiate a PFSignUpViewController
- Become its delegate so you can listen for events that happen (like successful signup or canceling)
- Present it!

Demo

Login

- Logging in can can be done in two ways:
    1. Writing the code to signup manually
    2. Using Parse's built in PFLoginViewController (contained in the ParseUI framework, you will need to specifically import that)

Manual Login

```
[PFUser logInWithUsernameInBackground:@"myname"
password:@"mypass"
  block:^(PFUser *user, NSError *error) {
    if (user) {
      // Do stuff after successful login.
    } else {
      // The login failed. Check error to see why.
    }
}];
```

PFLogInViewController

- Instantiate a PFLoginViewController
- Become its delegate so you can listen for events that happen (like successful login or canceling)
- Present it!

Demo

Current User

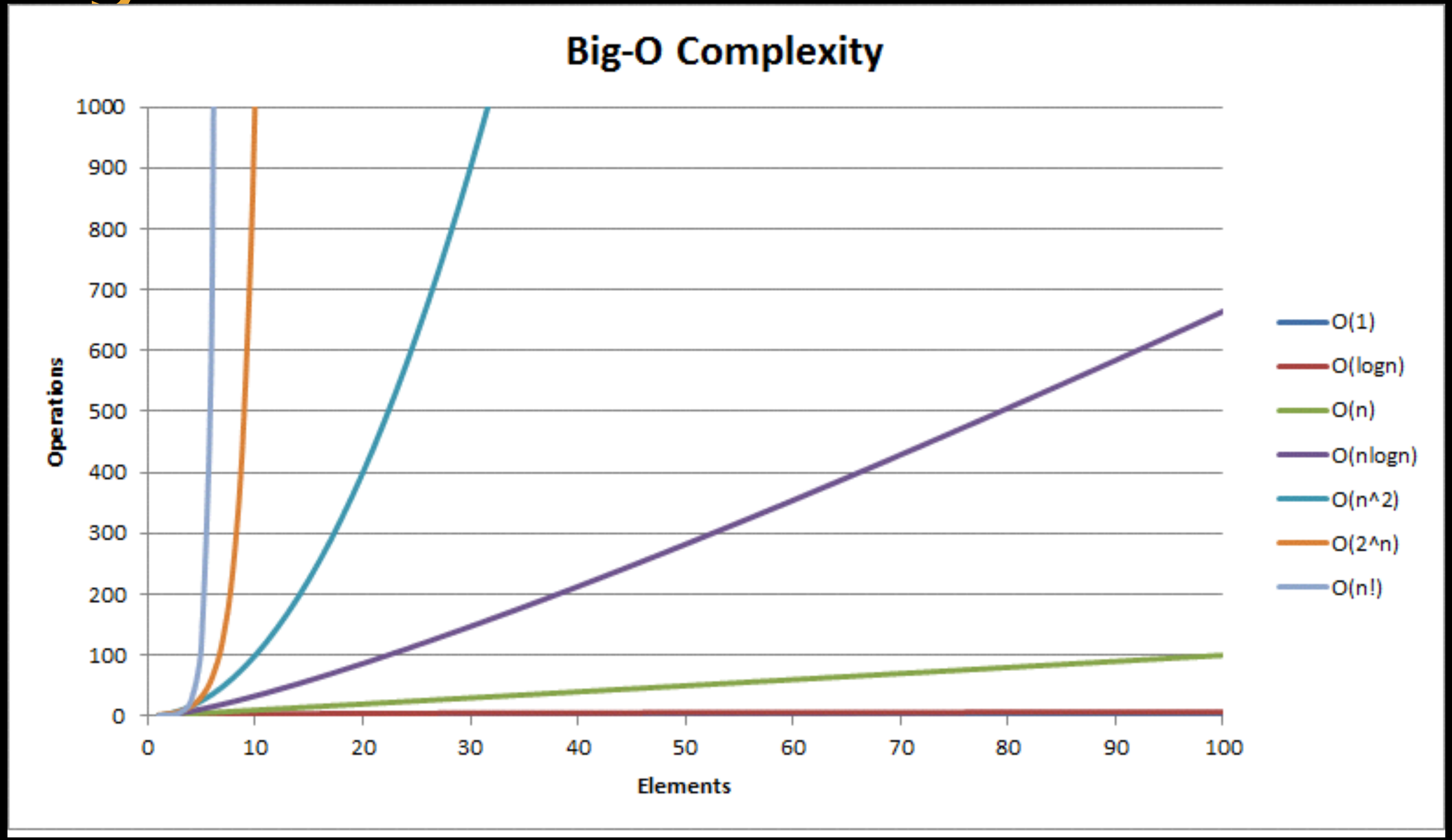
- Luckily, the user does need to go through this process every time they launch the app
- Parse locally caches the user that has signed up or logged in
- There is currentUser class method on PFUser that returns the currently logged in user
- At anytime you can check if this returns non-nil, meaning the user is logged in

Demo

Big O notation

- In computer science, big o notation is used to measure the efficiency of algorithms.
- The O stands for order, 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 containing constant time computations.
- 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 everything but the biggest part, so the Big-O of a regular for loop is just O(N)

O(N^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 for loop nested inside of another 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.

Demo

Linked Lists

A Linked List is…

A group of nodes linked together in a sequence.

Can think of it as a line of railroads cars

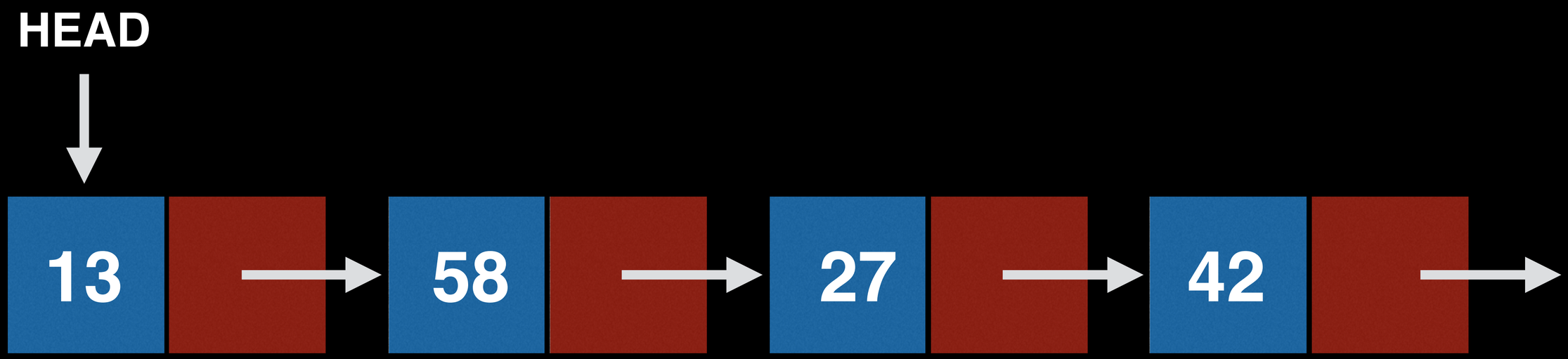
Each node stores some piece of **data**

Each node has a pointer to the **next** node

A favorite topic in Interviews… (and Queues and Stacks!)

To Find a Node…

Use the next pointer to walk the List



Data Access

- No direct access to all the nodes, must look through nodes
- Operations at the front are quick, at the back are slow
- Accessing data further down the list requires a longer walk
- This “linear” time cost is more expensive than “constant” time of an Array
- Basically, Linked Lists slower than Arrays, $O(n) > O(1)$

Traversing the List

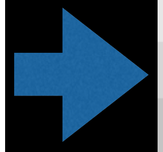
- Traversing a linked list is an important skill to have.
- It's easy once you know how to do it.
- There are two ways commonly used:
  - Recursion (maybe easier to read, but more overhead)
  - Iteration (less overhead, uses a while loop)

Iteratively Traversing

```
class LinkedList {

  var head : Node?

  func addValue(value : Int) {
    if self.head == nil {
      self.head = Node(value: value)
    } else {
      var currentNode = self.head
      while currentNode!.next != nil {
        currentNode = currentNode!.next
      }
      currentNode?.next = Node(value: value)
    }
  }
}
```

Recursively Traversing

```
class LinkedList {

  var head : Node?

  func addValue(value : Int) {
    if self.head == nil {
      self.head = Node(value: value)
    } else {
      self.head?.addValue(value)
    }
  }
}
```



```
class Node {
  var value : Int
  var next : Node?
  init(value: Int) {
    self.value = value
  }

  func addValue(value : Int) {
    if self.next == nil {
      self.next = Node(value: value)
    } else {
      self.next?.addValue(value)
    }
  }
}
```

Demo