Agenda

1. Icebreakers
2. Mini-Assignment
3. Topics
   - Heap and Deque
   - Generics
   - NDS4 Docs
   - Binary Search Trees
   - Coding Conventions
   - Hashing
Mini Assignment - Tree Traversals

Please form groups!

Each team should choose between doing
- Inorder
- Postorder
- Preorder
- Breadth-first Search
Answers!

1. **Inorder**: M-P-A-E-L-P-Y-S-U-R (left, self, right)
2. **Preorder**: P-E-P-M-A-L-U-Y-S-R (Self, left, right)
3. **Postorder**: M-A-P-L-E-S-Y-R-U-P (left, right, self)
4. **Breadth-First**: P-E-U-P-L-Y-R-M-A-S (each child at each level)
Practice Problem
Pseudocode your solution!

Given two lists $A$ and $B$, how can you determine whether any element of $A$ is an element of $B$?
First Approach

$O(A \times B)$ time answer: iterate through both lists in a nested for loop and check if any two elements match

```
for i in range 0 to length of list A
    for j in range 0 to length of list B
        if A[i] == B[j]
            return true
    return false
```
Second Approach

O(A+B) time answer: Add list A to some sort of hash set, and check each element of B to find if it’s in the set.

```python
for element in A:
    hashSet.add(element)
for element in B:
    if hashSet.contains(element):
        return true
return false
```
Topics

- Hashing
- Binary Search Trees
- Heap (required)
- Generics
Hashing
Overview
How do Hashing Functions work?

- Expected one element per bucket but it doesn’t always work.
- We would want to use them because they have \textbf{constant} insert/delete/lookup.
Differences between Hash set and Hash Table

- **Hash set**: has no keys, just values, and there’s no ordering
- **Hash table**: maps keys to values, there’s no ordering
Questions?
Binary Search Trees
How to tell if a tree is a BST?

1- Perform an inorder traversal and check if the returned numbers are in non-decreasing order
2- Recursive solution
   ● Check if each subtree meets the definition of a BST:
     ○ The left subtree of a node contains only nodes with keys less than the node’s key.
     ○ The right subtree of a node contains only nodes with keys greater than the node’s key.
     ○ Both the left and right subtrees must also be binary search trees.
Pseudocode

```python
function isValid(node, max, min)
    if node is null
        return true
    if node.value > max or node.value < min
        return false
    return isValid(node.left, node.value, min) and isValid(node.right, max, node.value)

Original function call: isValid(root, ∞, -∞)
```
What’s the difference between a Binary Tree and a Binary Search Tree?
Difference between BT and BST

**Binary Tree:** each node has at most two children

**Binary Search Tree:** Binary Tree where left child < parent and right child > parent.
Insertion and Deletion

Add 16?
Insertion and Deletion

Delete 7?
Insertion and Deletion

Add 1?
Insertion and Deletion

Delete 12?
Final Solutions
Coding Conventions

- What are some ways in which the ImportanceMethod could be written well?
  - How can we factor this method so that there aren’t a bunch of edge cases?
- ImportanceMethodB.java activity
Heap & Deque
Overview
Heap Clarifications

- Do you have any clarifying questions about the project?
- Things to note:
  - Deleting and adding nodes
    - Upheap and Downheap
  - Using an adaptable priority queue (APQ) instead of a conventional priority queue
    - Able to remove any internal node (not only min or max)
Heap Example

- **Downheap example:**
  - remove 8 from the following heap.

- **Upheap example:**
  - remove 50 from the following heap. (before the modifications above)
Deque Add and Remove

- What is a deque?
  - A deque is a double ended queue (push and pop from front and back)
  - Different from a queue and a stack!

- What is the relationship between a deque and heap?
  - Brainstorm with a partner and then share out loud!
If the deque is empty, set the root to be the new node. Add the node to the back of the deque.
Root: 1

If the parent has no left child, add left child:

Add the node to the back of the deque, and check the front of the deque to get the parent node.
ADD 3

Root: 1

If the parent has a left child, add right child

Add the node to the back of the deque, and check the front of the deque to get the parent node, and then pop the parent node off front of the deque because it has no space for new child nodes
Root: 1

If the parent has a right child, remove right child

Pop the node off the back of the deque, and check the parent of the removed node
Root: 1

If the parent has no right child, remove left child

Pop the node off the back of the deque, and check the parent of the removed node.
Root: Null

If the deque is empty, set the root to null.

Pop the node off the back of the deque, the deque will now be empty.
Deque Add and Remove cont.

Now that you know how a deque works when removing and adding a node:

- Try to write pseudocode for the add and remove functions!
  - Get into partners of 2 and start working!
  - Ask TAs for help/guidance
Generics
Generics

**Definition (from javadocs):** generics enable *types* (classes and interfaces) to be parameters when defining classes, interfaces and methods.

**How generics are helpful:**

1) By providing types as a parameter, type checking at compile-time will be more reliable. Fixing errors at compile time is easier than fixing errors at run-time, in which an error may occur anywhere within the lifecycle of the program

2) Creating generic algorithms - algorithms can be used on collections of types

3) Eliminate Casts (example on next slide)

Read more @: [https://docs.oracle.com/javase/tutorial/java/generics/why.html](https://docs.oracle.com/javase/tutorial/java/generics/why.html)
Generics

Eliminate Casts Example:

The following code snippet without generics requires casting:

```java
List list = new ArrayList();
list.add("hello");
String s = (String) list.get(0);
```

When re-written to use generics, the code does not require casting:

```java
List<String> list = new ArrayList<String>();
list.add("hello");
String s = list.get(0);  // no cast
```
Benefits of Generics

Let’s try an exercise! Look at this code and try to make it generic:

```java
public class Box {
    private Object _object;

    public void set(Object object) { _object = object; }
    public Object get() { return _object; }
}
```
Benefits of Generics

/**
 * Generic version of the Box class.
 * @param <T> the type of the value being boxed
 */

public class Box<T> {
    // T stands for "Type"
    private T _t;

    public void set(T t) { _t = t; }
    public T get() { return _t; }
}

Quick question!
How would we instantiate something of type Box?

Solution:
Box<Integer> box = new Box<Integer>();
Application of Generics in Heap

- Examples in Heap:
  - `MyLinkedHeapTree<E> uses Position<E>`
  - `MyHeap creates a MyLinkedHeapTree<MyHeapEntry<K,V>>`
  - `MyHeapEntry has a reference to its position in a variable of type Position<MyHeapEntry<K,V>>`

- So what’s the point?
  - Generics allow us to have one set of code that works for any type and guarantees that the different classes’ types match.
  - In the visualizer we use integers/strings, but the point of making our heap is to allow it to be used in the future, and therefore we don’t want to restrict the types that our heap can be used by.
Mini-Assignment
Mini Assignment

A. Name three properties of a heap and explain the differences between a position and entry.

B. The upheap method is called when you ______ a node and downheap is called when ______ a node? Does the position or the entry move when you upheap?

C. Give a real world application of a priority queue. What would the key/value pair types be?

D. Master Theorem is primarily used to solve _________ _________.

Valid Properties: Binary Tree, Each node has a priority (key), A heap has an order: Min-Heap vs Max-Heap, Left-complete, Height log(n), Run Time: Deletion $O(\log n)$ Insertion $O(\log n)$.

Position is the tree node, and it contains an entry. The entry is the data (key, value pair) stored in that position.

Upheap is called when you **insert** a node and down heap is called when you **remove** a node. When you upheap the entry moves not the position!

Emergency room. Key would be the severity of the issue and the value would be the patient’s name. Other examples: airports and flights/departure times, bandwidth management (prioritizing information)

Master Theorem is primarily used to solve **recurrence relations**.