Agenda

1. Icebreaker
2. Mini Assignment
3. Heap & Deque Overview
4. Generics
5. Master Theorem
Icebreaker!
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 _________ _________.

A. 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.

B. 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!

C. 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)

D. Master Theorem is primarily used to solve recurrence relations.
Heap & Deque Overview
Heap Clarifications

● 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)

● Do you have any clarifying questions about the project?
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!
Root: 1

If the deque is empty, set the root to be the new node

Add the node to the back of the deque.
Root: 1

1

If the parent has no left child, add left child

2

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

Deque
Front

1

Back

2
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
DELETE 3

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

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.
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

**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
```

You can actually create ArrayLists that store all types, but then you must cast when getting objects from the ArrayList.
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.
Master Theorem
Master Theorem: Review

Master Theorem Equation

\[ T(n) = a \cdot T\left(\frac{n}{b}\right) + \Theta(n^d) \]

- **a**: number of sub-problems
- **n/b**: size of each sub-problem
- **n^d**: work to prepare sub-problems & combine their solutions

Runtime Interpretation

- \( T(n) = \Theta(n^d) \) if \( a < b^d \)
- \( T(n) = \Theta(n^d \log(n)) \) if \( a = b^d \)
- \( T(n) = \Theta(n^{\log_{b}a}) \) if \( a > b^d \)

*Note: \( b \) should be subscripted*
Master Theorem: Example

Let $a = 2$, $b = 2$, $d = 1$, and $T(n) = 2T(n/2) + O(n^1)$. What is the runtime of this function?

a) $\Theta(n^d)$

b) $\Theta(n^d \log n)$

c) $\Theta(n^{\log_ba})$  *Note: $b$ should be subscripted*
Master Theorem: Example

Let $a = 2$, $b = 2$, $d = 1$, and $T(n) = 2T(n/2) + O(n^1)$. What is the runtime of this function?

a) $\Theta(n^d)$

b) $\Theta(n^d \log n)$

c) $\Theta(n^{\log_b a})$ *(Note: $b$ should be subscripted)*

Solution: B

$a = b^d$ so runtime is $T(n) = \Theta(n^d \log n)$