Priority Queue ADT & Heaps
Goals

• Introduce the Priority Queue ADT
• Heap Data Structure Concepts
Priority Queue ADT

• Not really a FIFO queue – misnomer!!

• Associates a “priority” with each element in the collection:
  – First element has the highest priority (typically, lowest value)

• Applications of priority queues:
  – To do list with priorities
  – Active processes in an OS
Priority Queue ADT: Interface

- Next element returned has highest priority

```plaintext
void add(newValue);
TYPE getMin();
void removeMin();
```
Heap: has 2 completely different meanings

1. Classic data structure used to implement priority queues
2. Memory space used for dynamic allocation

We will study the data structure (not dynamic memory allocation)
Priority Queue ADT: Implementation

Binary Heap data structure: a complete binary tree in which every node’s value is less than or equal to the values of its children (min heap)

Review: a complete binary tree is a tree in which

1. Every node has at most two children (binary)
2. The tree is entirely filled except for the bottom level which is filled from left to right (complete)
   - Longest path is $\text{ceiling}(\log n)$ for $n$ nodes
Min-Heap: Example

- **Root** = Smallest element

- **Next open spot**

- **Last filled position**
  (not necessarily the last element added)
Maintaining the Heap: Addition

Add element: 4

Place new element in next available position, then fix it by “percolating up”
Maintaining the Heap: Addition (cont.)

Percolating up:
while new value is less than parent,
swap value with parent

After first iteration (swapped with 7)

After second iteration (swapped with 5)
New value not less than parent → Done
Maintaining the Heap: **Removal**

- Since each node’s value is less than or equal to the values of its children, the root is always the smallest element.

- Thus, the operations `getMin` and `removeMin` access and remove the root node, respectively.

- Heap removal (**removeMin**):
  What do we replace the root node with?
  Hint: How do we maintain the completeness of the tree?
Heap removal (*removeMin*):

1. Replace root with the element in the last filled position
2. Fix heap by “percolating down”
Maintaining the Heap: Removal

**removeMin**:  
1. Move element in last filled pos into root  
2. Percolate down

Root = Smallest element

Last filled position
Maintaining the Heap: Removal (cont.)

Percolating down:
while greater than smallest child
swap with smallest child

After first iteration (swapped with 3)

Root value removed
(16 copied to root and last node removed)
Maintaining the Heap: Removal (cont.)

Percolating down:
- while greater than smallest child
- swap with smallest child

After second iteration (moved 9 up)
- Reached leaf node → Stop percolating

After third iteration (moved 12 up)
Maintaining the Heap: Removal (cont.)

Root = New smallest element

New last filled position
Insert the following numbers into a min-heap in the order given:  54, 13, 32, 42, 52, 12, 6, 28, 73, 36
Remove the minimum value from the min-heap
Your Turn

• Complete Worksheet: Heaps Practice