**Topic 24**  
**Heaps**

"You think you know when you can learn, are more sure when you can write even more when you can teach, but certain when you can program."

- Alan Perlis

---

**Priority Queue**

- Recall priority queue
  - elements enqueued based on priority
  - dequeue removes the highest priority item

- Options?
  - List? Binary Search Tree?

<table>
<thead>
<tr>
<th>Linked List enqueue</th>
<th>BST enqueue</th>
</tr>
</thead>
<tbody>
<tr>
<td>O(N)</td>
<td>O(1)</td>
</tr>
<tr>
<td>O(N)</td>
<td>O(logN)</td>
</tr>
<tr>
<td>O(N)</td>
<td>O(N)</td>
</tr>
<tr>
<td>O(logN)</td>
<td>O(logN)</td>
</tr>
<tr>
<td>O(1)</td>
<td>O(logN)</td>
</tr>
</tbody>
</table>

---

**Another Option**

- A heap
  - not to be confused with the runtime heap (portion of memory for dynamically allocated variables)

- A complete binary tree
  - all levels have maximum number of nodes except deepest where nodes are filled in from left to right

- Maintains the *heap order property*
  - in a min heap the value in the root of any subtree is less than or equal to all other values in the subtree

---

**Clicker Question 2**

- In a max heap with no duplicates where is the largest value?

  A. the root of the tree
  B. in the left-most node
  C. in the right-most node
  D. a node in the lowest level
  E. None of these
Example Min Heap

Enqueue Operation

- Add new element to next open spot in array
- Swap with parent if new value is less than parent
- Continue back up the tree as long as the new value is less than new parent node

Enqueue Example

- Add 15 to heap (initially next left most node)

Enqueue Example

- Swap 15 and 52
Enqueue Example

Swap 15 and 17, then stop

Internal Storage

Interestingly heaps are often implemented with an array instead of nodes

PriorityQueue Class

```java
public class PriorityQueue<E extends Comparable<E>> {

    private int size;
    private E[] con;

    public PriorityQueue() {
        heap = getArray(2);
    }

    private E[] getArray(int size) {
        return (E[]) (new Comparable[size]);
    }

    public void enqueue(E val) {
        if (size >= con.length - 1)
            enlargeArray(con.length * 2 + 1);
        size++;
        int indexToPlace = size;
        while (indexToPlace > 1
            && val.compareTo(con[indexToPlace / 2]) < 0) {
            con[indexToPlace] = con[indexToPlace / 2]; // swap
            indexToPlace /= 2; // change indexToPlace to parent
        }
        con[indexToPlace] = val;
    }

    private void enlargeArray(int newSize) {
        E[] temp = getArray(newSize);
        System.arraycopy(con, 1, temp, 1, size);
        con = temp;
    }
```

PriorityQueue enqueue

for element at position i:
- parent index: \( i / 2 \)
- left child index: \( i \times 2 \)
- right child index: \( i \times 2 + 1 \)
Dequeue

- min value / front of queue is in root of tree
- swap value from last node to root and move down swapping with smaller child unless values is smaller than both children

Dequeue Example

- Swap 35 into root (save 12 to return)
Dequeue Example

- Swap 35 with smaller child (17)

```java
public E dequeue() {
    E top = con[1];
    int hole = 1;
    boolean done = false;
    while (hole * 2 < size && !done) {
        int child = hole * 2;
        // see which child is smaller
        if (con[child].compareTo(con[child + 1]) > 0) {
            child++;
            // child now points to smaller
        }
        // is replacement value bigger than child?
        if (con[hole].compareTo(con[child]) > 0) {
            con[hole] = con[child];
            hole = child;
        } else {
            done = true;
        }
        con[hole] = con[size];
        size--;
    }
    return top;
}
```

Dequeue Example

- Swap 35 with smaller child (21)

PriorityQueue Comparison

- Run a Stress test of PQ implemented with Heap and PQ implemented with BinarySearchTree

- What will result be?
  A. Heap takes half the time or less of BST
  B. Heap faster, but not twice as fast
  C. About the same
  D. BST faster, but not twice as fast
  E. BST takes half the time or less of Heap
Data Structures

- Data structures we have studied:
  - arrays, array based lists, linked lists, maps, sets, stacks, queue, trees, binary search trees, graphs, hash tables, red-black trees, priority queues, heaps
- Most program languages have some built in data structures, native or library
- Must be familiar with performance of data structures
  - best learned by implementing them yourself

We have *not* covered every data structure:


- deque, b-trees, quad-trees, binary space partition trees, skip list, sparse list, sparse matrix, union-find data structure, Bloom filters, AVL trees, trie, 2-3-4 trees, and more!
- Must be able to learn new and apply new data structures