Event Queues

- Windows and Mac use GUI
- Mouse movements, mouse button presses and releases, and key presses are "events"
- Typically GUI application will have an event loop that gets next event and figures out what to do
- Events are stored in queue so they are processed in FIFO order
- Some events are important so need to jump ahead in the queue and get processed earlier

Run Queues

- When a process on your machine blocks to do I/O, the O/S selects another process to run
- A FIFO queue can be used to give fair scheduling of the runnable processes
- Some processes are more important and should be run instead of others
  - Particularly interactive processes

Priority Queues

- Collection of items with a priority
- Operations:
  - Create empty priority queue
  - Is priority queue empty?
  - Insert an element
  - Remove element with highest priority
    For equal priorities may want FIFO

Heapsort

- Can use a priority queue to sort an array
- Use keys of items as priority
- Insert all elements of array into priority queue
- Remove elements one at a time and put in array from top down

Heaps

- Complete binary tree
  - All levels completely full except last
  - Last level has all elements from left to somewhere in middle
- Heap property
  - A node's item is always greater than or equal to its children's items
A Heap

Delete Code

```java
Object delete() {
    Object max = head.item;
    TreeNode cur = head;
    Object item = last.item;
    remove last and update last;
    while (item < cur.left.item || item < cur.right.item)
        if (cur.left.item < cur.right.item)
            cur.item = cur.right.item; cur = cur.right;
        else
            cur.item = cur.left.item; cur = cur.left;
    cur.item = item;
    return max;
}
```

Heapify

```java
Object heapify(TreeNode cur) {
    Object item = cur.item;
    while (item < cur.left.item || item < cur.right.item)
        if (cur.left.item < cur.right.item)
            cur.item = cur.right.item; cur = cur.right;
        else
            cur.item = cur.left.item; cur = cur.left;
    cur.item = item;
}
```

Insert

```java
void insert(Object item) {
    create new node next to last and update last;
    TreeNode cur = last;
    while (item > cur.parent.item) {
        cur.item = cur.parent.item;
        cur = cur.parent;
    }
    cur.item = item;
}
```
Heaps With Linked Nodes

- If techniques of last lecture are used:
  - Head pointer, last pointer
  - Nodes with three pointers and data item
  - Dynamically allocate and deallocate
  - Large amount of space and time
- In general trees must be done this way
- But Heaps can be implemented much better

Heaps With Arrays

- Layout the nodes sequentially in an array
- Because the tree is full this the layout will be systematic and it will be easy to calculate child and parent indices from a node's index

Heap Layout

Parent/Left/Right Formulas

- If node \( n \) is located at index \( i \) in the array:
  - \( n \)'s parent is located at index \( i/2 \)
  - \( n \)'s left child is located at index \( 2i \)
  - \( n \)'s right child is located at index \( 2i+1 \)
- These are 1-2 instructions on most machines
- Note that the layout is 1 based; Java is 0 based

Heap Implementation (cont.)

- Representation:
  - Array of items
  - length or max_index field
- To determine if child exists, compare its index with max_index
- Root is always \( 1 \) and has no parent, all other nodes are index \( >1 \) and have parents

Heapsort

```java
void heapsort(Object items) {
    for(int i=items.length/2; i>=1; i--)
        heapify(items, items.length, i);
    for(int i=items.length; i>1; i--)
        items[i] = delete(items, i);
}
```
Heap Layout