CSCI 104L Lecture 9: Heaps

Priority Queues

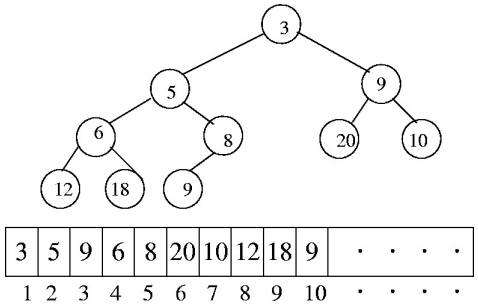
In a **Priority Queue** ADT, you may perform the following operations:

- Add an item (with a priority)
- Return the item of highest priority
- Delete the item of highest priority

Question 1. What would be the runtime of add/peek remove, using a...

- unsorted array / linked list?

- sorted array?



A tree with values in the nodes. Consider this figure when answering the following questions.

Question 2. What kind of tree is this?

Question 3. Do we really need to store it as a tree, or is there a more compact representation?

Question 4. If we are at index x, which index is "above" it in the tree?

Question 5. If we are at index x, which index is "below it to the left"? "below it to the right"?

We say that a complete tree has the **maxHeap property** if, for each item in our "tree," it will have higher (or equal) priority to anything below it.

Here are some of the function implementations, using a maxHeap:

```
T PriorityQueue::peek() const {
  return a[0];
}
void PriorityQueue::add(const T& data) {
  a[size] = data;
  bubbleUp(size);
  size++;
}
void PriorityQueue::bubbleUp(int pos) {
  if (pos > 0 && a[pos] > a[(pos-1)/2]) {
    a.swap(pos, (pos-1)/2);
    bubbleUp((pos-1)/2);
  }
}
void PriorityQueue::remove() {
  a.swap(0,size-1);
  size--;
  trickleDown(0);
}
void PriorityQueue::trickleDown(int pos) {
  int child = 2*pos+1;
  if (child < size) {
    if (child+1 < size && a[child] < a[child+1]) child++;
    if (a[child] > a[pos]) {
      a.swap(child, pos);
      trickleDown(child);
    }
  }
}
```

Question 6. What is the runtime for each of these operations?:

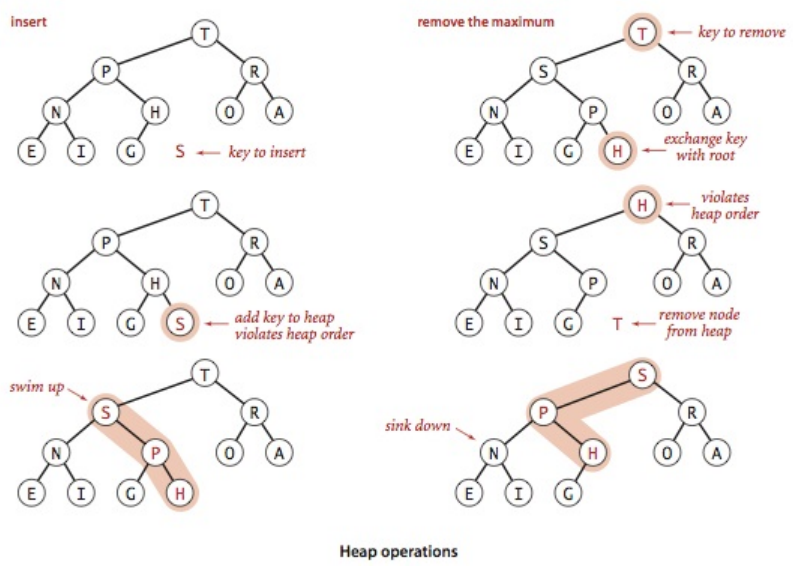


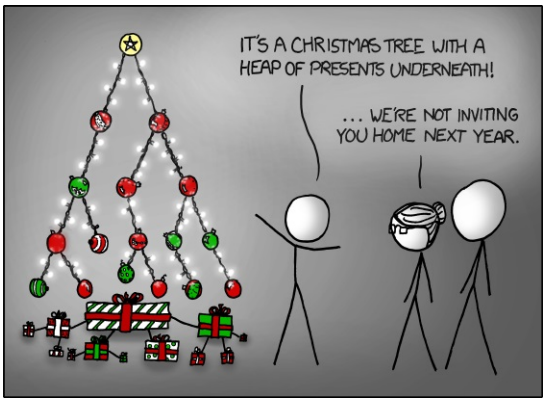
Image produced by Robert Sedgewick and Kevin Wayne

```
void HeapSort(int *a, int size) {
  Heap h;
  for (int i = 0; i < size; i++) {
    h.add(a[i]);
  }
  for (int i = 0; i < size; i++) {
    a[i] = h.peek();
    h.remove();
  }
}
```

Question 7. What's the runtime of Heapsort?

Question 8. Is it stable?

```
void MinHeap::UpdatePriority(int node, int priority) {
  int location = map[node];
  if (a[location] < priority) {
    a[location] = priority;
    trickleDown(location);
  } else {
    a[location] = priority;
    bubbleUp(location);
  }
}
class MinHeap {
  public:
    void UpdatePriority(int node, int priority);
  private:
    int *a; // stores the priorities
    int *map; // stores the locations of each node
};
```



XKCD #835: Not only is that terrible in general, but you just KNOW Billy's going to open the root present first, and then everyone will have to wait while the heap is rebuilt.