Max, min, and priority queues

An important data structure is the priority queue: A priority queue has two operations: (1) inserting an item in the queue and (2) extracting the highest priority item. Conventionally the highest priority item has the lowest value and the extraction function is called `extract_min()`.

Operating systems use priority queues to manage timesharing of running jobs. Web services use priority queues to manage service delivery timeliness. Algorithms use priority queues to manage searches through data (shortest path problems), etc., etc..

arrayMax could be thought of as a step toward a priority queue design: Insertion would be to add an element to the array, extraction would be arrayMax (modified to remove the item). Turns out: *we can do much better than this*.

Strawman design for priority queues

```
template <class T>
class PriorityQueue {
    int n; // current size
    T p[?]; // unsorted array of items
public:
    void add(T x) { P[n] = x; n = n + 1; }
    T extract_min() {
        T m = min(P, n); // also get index i of m.
        copy p[i+1]..p[n-1] to p[i]..p[n-2].
        n = n - 1;
        return m;
    }
};
```

Issues: 1. extract)min is expensive 2. how big should array p be?

Second strawman design for priority queues

```
template <class T>
class PriorityQueue {
    int n; // current size
    T p[?]; // sorted array of items
public:
    void add(T x) {
        use binary search to find i where x should be inse
        copy p[i..n-1] to p[i+1..n]
        p[i] = x;
        n = n + 1;
    }
    T extract_min() { n = n - 1; return p[n]; }
};
```

Issues: 1. add is expensive 2. how big should array p be?

files that tell the story

Have a look at:

1. call-stack.cpp
2. call-stack2.cpp
3. dynamic-array.h
4. example situation
5. mergeSort.cpp