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 1  What’s the runtime of Heapsort?

Question 2  Is it stable?

Figure 1: 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.

Dijkstra’s Algorithm

int d[n]; //distances from the start node u
int p[n]; //predecessors
int c[n][n]; //edge costs
void Dijkstra (int u) {
    PriorityQueue<int> pq(); //How should we implement this?
    d[u] = 0;
    pq.add(u, d[u]);
    while(!pq.isEmpty()) {
        int v = pq.peek();
        pq.remove();
    }
for all nodes outgoing edges (v,w) from v {
    if (w hasn't been visited || d[v] + c[v][w] < d[w]) {
        d[w] = d[v] + c[v][w];
        p[w] = v;
        if (this is w’s first visit) {
            pq.add(w);
        }
        else pq.update(w, p[w]);
    }
}

**Question 3** How many add/remove/update calls are needed?

**Question 4** What is the runtime of Dijkstra’s using an unsorted array as an implementation for that?

**Question 5** What is the runtime of Dijkstra’s using a sorted array for that?

**Question 6** What is the runtime of Dijkstra’s using a MinHeap/Priority Queue for that structure?

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
};

Note that in Dijkstra’s Algorithm, we only ever bubbleUp.

**Question 7** Is there any type of graph where you’d want to use an unsorted array instead of a heap?