

1 Sixty-two

- 1.1 Each of the following sequences represent an array being sorted at some intermediate step. Match each sample with one of the sorting algorithms: **insertion sort**, **selection sort**, **heapsort**, **merge sort**, **quicksort**. The original array is below.

5103 9914 0608 3715 6035 2261 9797 7188 1163 4411

- (a) 5103 9914 0608 3715 2261 6035 7188 9797 1163 4411
0608 2261 3715 5103 6035 7188 9797 9914 1163 4411

Merge sort

- (b) 0608 1163 5103 3715 6035 2261 9797 7188 9914 4411
0608 1163 2261 3715 6035 5103 9797 7188 9914 4411

Selection sort

- (c) 9797 7188 5103 4411 6035 2261 0608 3715 1163 9914
4411 3715 2261 0608 1163 5103 6035 7188 9797 9914

Heapsort

- (d) 5103 0608 3715 2261 1163 4411 6035 9914 9797 7188
0608 2261 1163 3715 5103 4411 6035 9914 9797 7188

Quicksort

- (e) 0608 5103 9914 3715 6035 2261 9797 7188 1163 4411
0608 2261 3715 5103 6035 9914 9797 7188 1163 4411

Insertion sort

- 1.2 Give the *amortized runtime analysis* for push and pop for the priority queue below.

```
class TwinListPriorityQueue<E implements Comparable> {
    ArrayList<E> L1, L2;
    void push(E item) {
        L1.push(item);
        if (L1.size() >= Math.log(L2.size())) {
            L2.addAll(L1);
            mergeSort(L2);
            L1.clear();
        }
    }
    E pop() {
        E min1 = getMin(L1);
        E min2 = L2.poll();
        if (min1.compareTo(min2) < 0) {
            L1.remove(min1);
            return min1;
        }
    }
}
```

2 Final Review

```
    } else {  
        L2.remove(min2);  
        return min2;  
    }  
}  
}
```

Let N be the number of elements in the priority queue. Then the amortized runtime for push is in $O(N)$ as the cost for every $\log N$ insertions is in $O(\log N \cdot 1 + 1 \cdot N \log N)$ which simplifies to $O(N)$. Note that the size of L1 is always constrained to be in $O(\log N)$.

The amortized runtime for pop is also in $O(N)$. `getMin` on the unsorted list, L1, is in $O(\log N)$, as with `L1.remove(min1)`. Polling from the front of L2 is in $\Theta(1)$. The most expensive component is `L2.remove(min2)` which is in $O(N)$.

- 1.3 Briefly describe an efficient algorithm (and report the running time) for finding a minimum spanning tree in an undirected, connected graph $G = (V, E)$ when the edge weights satisfy:

(a) For all $e \in E$, $w_e = 1$.

The key idea here is that any tree which connects all nodes is an MST. We can run DFS and take the DFS tree. You could also take a BFS tree, or run Prim's algorithm with a queue or stack instead of a priority queue (this would be equivalent to BFS/DFS). The runtime of this algorithm is in $\Theta(|V| + |E|) \in \Theta(|E|)$ for simple graphs.

(b) For all $e \in E$, $w_e \in \{1, 2\}$. (In other words, all edge weights are either 1 or 2.)

Run Prim's algorithm with a specialized priority queue, comprised of 2 regular queues. When we add items to the priority queue, we add it to the first queue if the weight is 1 and otherwise add it to the second queue (the weight must be 2). When we pop from the priority queue, we take from the first queue, unless it is empty, in which case we take from the second. The runtime of this algorithm is in $\Theta(|E|)$ as priority queue operations are now constant time.

- 1.4 Find the Huffman encoding for the following alphabet and set of frequencies.

$\{(a, 0.12), (b, 0.38), (c, 0.1), (e, 0.25), (f, 0.06), (d, 0.05), (g, 0.01), (h, 0.03)\}$

When you build up your Huffman tree, you should place the branch of lower weight on the left. A left or right branch should respectively correspond to a 0 or 1 in the codeword.

$$a = 1111$$

$$b = 0$$

$$c = 1110$$

$$d = 11011$$

$$e = 10$$

$$f = 1100$$

$$g = 110100$$

$$h = 110101$$