Please know amortized analysis (along with all the other stuff).

Amortized analysis:

sum of amortized costs should be ≥ sum of real costs
taxation:

collect "taxos" alongside each operation

pay for part with taxes you collect alreadly

amortized cost:

real cost + tax paid

- how much you pay for with earlier taxes

charging:

amortized cost:

real cost - charge to earlier ops + charges from later ops
potentials

\[ \Phi_i : \text{potential of DS at operation } i \]

keep \( \Phi_i - \Phi_{i-1} \) non-negative

am. cost

\[ a_i = c_i + \Phi_i - \Phi_{i-1} \]

real cost
**Ex:** Fib. heap

\[ \Phi := \# \text{trees}(H) + 2 \# \text{marks}(H) \]

Several heap-ordered trees.

Inserts just add a single node to a tree.

Delete Min:

1) Remove min root, creating new trees from its children.
2) Iteratively combine trees of same rank until all ranks
are distinct.

rank: # of children of root

cost is prop. to # trees, but then # decreases by 2 # trees, so it works out

Decrease key might move many marked nodes, but then # marks decrease so √
Basic Graph Algorithms:

Depth-First search (DFS) - makes a tree rooted at input vertex or forest for dfsAll
- used for directed cycle detection
  - two algos
  - $O(V + E)$ repeatedly grab in-degree $0$ vertex & delete $dfsAll$ then reverse postorder
Shortest path trees
- breadth-first search (BFS)
  \(O(1 + |E|)\)
- Dijkstra's alg
  (non-neg. weights)
  \(O(|E| \log |V|)\)
  or \(O(|V| \log |V| + |E|)\) with Fib. heaps
- Bellman-Ford
  (neg. weight edges)
  \(O(|V||E|)\)
- in a DAG $O(V^3)$

- all-pairs shortest paths
  
  - Floyd-Warshall $O(V^3)$

- minimum spanning trees
  
  - Prim $O(VE\log V)$ or $O(V\log V + E\log E)$ with a heap
  
  - Kruskal $O(VE\log V)$
External memory:

- M internal mem
- mem goes between internal to external
- in blocks of size B

Try to minimize A of block read/writes

B-trees:

- search tree where each internal node has between $B/2 + B$ children
- (root has 2 to B children)
all operations in \[ O(\log_{\frac{N}{B}} \frac{N}{B}) \] I/Os

Can sort in

\[ O(\frac{N}{B} \log_{\frac{N}{B}} \frac{N}{B}) \] I/Os

"external mergesort"
I recommend practicing divide-and-conquer recurrences.

If you use a binary heap runtime for a graph alg, that's fine.

Style like midterm.

Know basic sorting algs.
Dynamic Programming Ex:
Given $x[0...n-1]$ of characters, find length of longest palindrome subsequence.

1st: solve problem recursively

MaxPalSub(i,j): length of longest pal subseq. of $x[i...j]$.

Want to know $MaxPalSub(0,n-1)$
Step 2: Solve subproblems iteratively from big to small.

max \{ \text{MaxPalSub}(i, j) \} \quad \text{m.i.o.}

\begin{align*}
\text{MaxPalSub}(i, j) &= \\
&= \begin{cases} \\
2 + \text{MaxPalSub}(i, j-1) & \text{if } \text{is } \downarrow \\
\text{MaxPalSub}(i+1, j) & \text{if } \text{is } \uparrow \\
\text{MaxPalSub}(i, j-1) & \text{otherwise}
\end{cases}
\end{align*}
public static void longestPalindromeLength(char[] x) {
    int n = x.length;

    // Second base case prevents i = n + 1 or j =
    int[][] maxPalSub = new int[n][n];
    for (int i = n - 1; i >= 0; i--) {
        for (int j = 0; j < n; j++) {
            if (i > j) {
                maxPalSub[i][j] = 0;
            } else if (i == j) {
                maxPalSub[i][j] = 1;
            } else if (x[i] == x[j]) {
                maxPalSub[i][j] = 2 + maxPalSub[i + 1][j - 1];
            } else {
                maxPalSub[i][j] = Math.max(maxPalSub[i][j - 1],
                                            maxPalSub[i + 1][j]);
            }
        }
    }

    return maxPalSub[0][n - 1];
}
Hash tables:
- O(1) expected lookup/insert/remove
- Might be bigger
- Can do chaining or probing
to deal with collisions
- Perfect hashing uses chaining
  with other hash tables
to guarantee $\Theta(1)$ time
  $\Theta(n)$ space
eval.utdallas.edu

Thank you,
