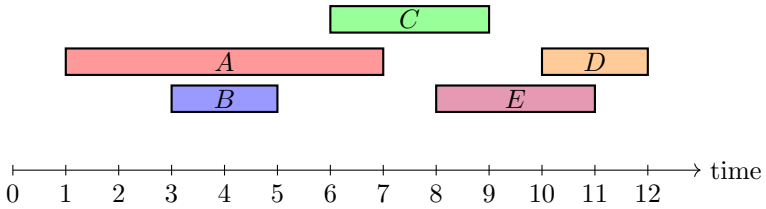
*Note*: Your TA probably will not cover all the problems. This is totally fine, the discussion worksheets are deliberately made long so they can serve as a resource you can use to practice, reinforce, and build upon concepts discussed in lecture, readings, and the homework.

# 1 Greedy Counterexamples

In lecture, we covered the problem of *interval scheduling*. We are given $n$ intervals, where each interval $i$ has a start time $s_i$ and a finish time $f_i$. Two intervals are *compatible* if they do not overlap. Our goal is to select the maximum number of mutually compatible intervals.

**Example:** Consider the following 5 intervals:




The optimal solution is $\{B, C, D\}$ with 3 intervals.

In lecture, we saw that the correct greedy strategy is to always pick the interval with the earliest finish time. Let's see why other greedy strategies don't work. For each of the following, give a counterexample (a set of intervals) that shows the greedy strategy fails to find the optimal solution.

(a) Greedy by most conflicts: always pick the interval that conflicts with the most other remaining intervals. Give a counterexample.

(b) Greedy by earliest start time: always pick the interval with the earliest start time. Give a counterexample.

(c) Greedy by least conflicts: always pick the interval that conflicts with the fewest number of other remaining intervals. Give a counterexample.

(d) Greedy by shortest length: always pick the shortest interval. Give a counterexample.

(e) Greedy by latest start time: always pick the interval that starts last.

## 2 Minimizing Average Completion Time

We have $n$ processes with execution times $t_1, \ldots, t_n$. We need to schedule all $n$ processes sequentially. Given an ordering (i.e. a permutation) $p_1, \ldots, p_n$ of the processes, the *completion time* of process $p_i$ is

$$C_{p_i} = \sum_{j=1}^{i} t_{p_j}.$$

Our goal is to find an ordering that minimizes the average completion time $\frac{1}{n}\sum_{i=1}^{n} C_{p_i}$.

**Example:** Consider 7 processes with execution times $T = [2, 1, 9, 1, 4, 3, 3]$.

In the given order, the completion times are $C = [2, 3, 12, 13, 17, 20, 23]$, so the average completion time is $90/7 \approx 12.86$. Can we do better? Consider the ordering $T' = [1, 2, 1, 3, 4, 3, 9]$. Then $C' = [1, 3, 4, 7, 11, 14, 23]$ and the average is $63/7 = 9$. Much better, but is this optimal?

Describe an efficient algorithm to find the ordering that minimizes the average completion time. Prove that your algorithm is correct and analyze its runtime.

# 3 Huffman Coding Review

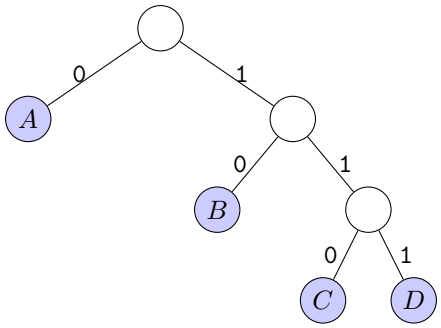
*This section is a walkthrough not*

## Prefix Codes are Binary Trees

Recall that any prefix-free code can be represented by a binary tree where the leaves correspond to the symbols.

- **Left edge** represents a `0`.
- **Right edge** represents a `1`.
- **Code for a symbol** is the path from root to that leaf.

Why does this ensure the prefix-free property? Since symbols are only at the leaves, the path to any symbol cannot be a prefix of the path to another symbol (which would require passing through a leaf to get to another node).




In this example: $A = \texttt{0}, B = \texttt{10}, C = \texttt{110}, D = \texttt{111}$.

## The Optimization Problem

Given frequencies $f_1, \ldots, f_n$ for our $n$ symbols, we want to construct a binary tree $T$ with leaves $1, \ldots, n$ that minimizes the total length of the message:

$$\text{Cost}(T) = \sum_{i=1}^{n} f_i \cdot \text{depth}_T(i)$$

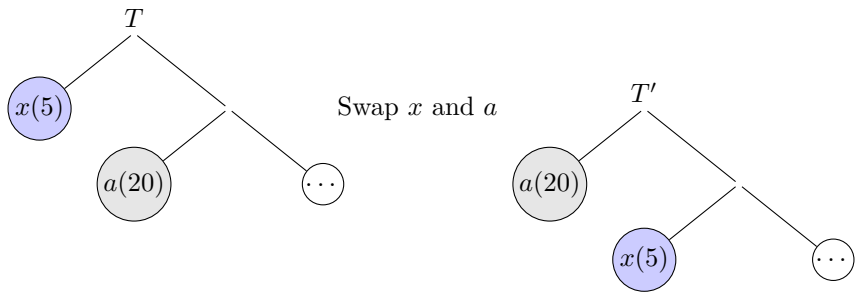
where $\text{depth}_T(i)$ is the length of the path from the root to leaf $i$.

## The Exchange Argument

Huffman's algorithm greedily merges the two nodes with the smallest frequencies. Why is this correct? It relies on the idea that in an optimal tree, the two symbols with the lowest frequencies must be siblings at the deepest level.

Let's see why with a concrete example. Suppose we have a tree where a rare symbol $x$ (frequency 5) is near the root (depth 1), and a frequent symbol $a$ (frequency 20) is deep in the tree (depth 3). This seems inefficient.




If we swap $x$ and $a$:

- $x$ moves from depth $1 \to 3$ (deeper by 2 levels).
  Cost from $x$ increases: $5 \times (+2) = +10$.
- $a$ moves from depth $3 \to 1$ (shallower by 2 levels).
  Cost from $a$ decreases: $20 \times (-2) = -40$.

The total change in cost is $+10 - 40 = -30$. The cost decreased!

This means the original tree $T$ was not optimal. By swapping, we improved the solution. We can keep swapping until the lowest frequency items are at the bottom, which is exactly what Huffman coding produces.

**Note:** This is a *proof by example*, which is generally **not** a valid proof strategy! A rigorous proof would use variables ($f_x, f_a$, etc.) and inequalities to show this holds for *any* frequencies where $f_x < f_a$ and $d_x < d_a$, but the intuition is identical.

# 4 Updating a MST

You are given a graph $G = (V, E)$ with positive edge weights, and a minimum spanning tree $T = (V, E')$ with respect to these weights; you may assume $G$ and $T$ are given as adjacency lists. Now suppose the weight of a particular edge $e \in E$ is modified from $w(e)$ to a new value $\hat{w}(e)$. You wish to quickly update the minimum spanning tree $T$ to reflect this change, without recomputing the entire tree from scratch.

There are four cases. In each, give a description of an algorithm for updating $T$, a proof of correctness, and a runtime analysis for the algorithm. Note that for some of the cases, these may be quite brief. For simplicity, you may assume that no two edges have the same weight (this applies to both $w$ and $\hat{w}$).

(a) $e \in E'$ and $\hat{w}(e) < w(e)$

(b) $e \notin E'$ and $\hat{w}(e) < w(e)$

(c) $e \in E'$ and $\hat{w}(e) > w(e)$

(d) $e \notin E'$ and $\hat{w}(e) > w(e)$