CS3230
AY21/22 SEM 2
github/jovyntls

01. COMPUTATIONAL MODELS

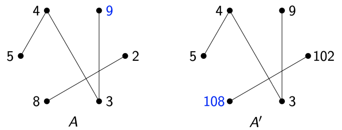
- **algorithm** $\rightarrow$ a well-defined procedure for finding the correct solution to the input
- **correctness**
  - **worst-case correctness** $\rightarrow$ correct on *every valid input*
  - other types of correctness: correct on random input/with high probability/approximately correct
- **efficiency** / **running time** $\rightarrow$ measures the number of steps executed by an algorithm as a function of the *input size* (depends on computational model used)
  - number input: typically the length of binary representation
  - **worst-case** running time $\rightarrow$ *max* number of steps executed when run on an input of size n

Comparison Model

- algorithm can **compare** any two elements in one time unit ($x > y, x < y, x = y$)
- running time = number of comparisons made
- array can be manipulated at no cost

Maximum Problem

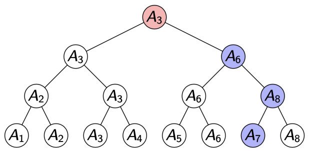
- *problem*: find the largest element in an array A of n distinct elements
- *proof that $n-1$ comparisons are needed*:
  - fix an algorithm M that solves the Maximum problem on all inputs using $< n-1$ comparisons. construct graph G where nodes i and j are adjacent iff M compares i and j.



  - M cannot differentiate A and A'.
- **adversary argument** $\rightarrow$ inputs are decided such that they have different solutions

Second Largest Problem

- *problem*: find the second largest element in $< 2n-3$ comparisons (2x Maximum $\Rightarrow (n-1)+((n-1)-1) = 2n-3$)
- *solution*: **knockout tournament** $\Rightarrow n + \lceil \lg n \rceil - 2$



1. bracket system: $n-1$ matches
   - every non-winner has lost exactly once
2. then compare the elements that have lost to the largest
   - the second-largest element must have lost to the winner
   - compares $\lceil \lg n \rceil$ elements that have lost to the winner using $\lceil \lg n \rceil - 1$ comparisons

Sorting

- there is a sorting algorithm that requires $\leq n \lg n - n + 1$ comparisons.
- *proof:* every sorting algorithm must make $\geq \lg(n!)$ comparisons.
  1. let set $\mathcal{U}$ be the set of all permutations of the set $\{1, \ldots, n\}$ that the adversary could choose as array A. $|\mathcal{U}| = n!$
  2. for each query "is $A_i > A_j$?", if $\mathcal{U}_{yes} = \{A \in \mathcal{U} : A_i > A_j\}$ is of size $\geq |\mathcal{U}|/2$, set $\mathcal{U} := \mathcal{U}_{yes}$. else: $\mathcal{U} := \mathcal{U} \backslash \mathcal{U}_{yes}$
  3. the size of $\mathcal{U}$ decreases by at most half with each comparison
  4. for $> \lg(n!)$ comparisons, $\mathcal{U}$ will still contain at least 2 permutations

$$n! \geq (\tfrac{n}{e})^n$$
$$\Rightarrow \lg(n!) \geq n \lg(\tfrac{n}{e}) = n \lg n - n \lg e$$
$$\approx n \lg n - 1.44n$$

$\Rightarrow$ roughly $n \lg n$ comparisons are **required** and **sufficient** for sorting n numbers

String Model

- input: string of n bits
- each query: find out **one bit** of the string
- n queries are **necessary** and **sufficient** to check if the input string is all 0s.

Graph Model

- input: (symmetric) adjacency matrix of an n-node undirected graph
- each query: find out if an edge is present between two chosen nodes
- *proof*: $\binom{n}{2}$ queries are necessary to decide whether the graph is connected or not
  1. suppose M is an algorithm making $\leq \binom{n}{2}$ queries.
  2. whenever M makes a query, the algorithm tries not adding this edge, but adding all remaining unqueried edges.
     - 2.1. if the resulting graph is connected, M replies 0 (i.e. edge does not exist)
     - 2.2. else: replies 1 (edge exists)
  3. after $< \binom{n}{2}$ queries, at least one entry of the adjacency matrix is unqueried.