Monday, Feb. 19, 2024
Scientific Computing

Announcements:
→ Office Hours canceled today
Tuesday, 11am, Teams
→ HW 2 due on Friday

Topic 5 - Search Spaces + Brute Force
(continued)

Example: Gamestop problem.
What does a possible solution look like?
You have 60 transaction slots and you need to assign a person to each one.
If you start with n people, how many ways can this be done?

60 slots: _ _ _ _ _ .... _
(Not just valid assignments — all assignments)

Slot 1: n people
Slot 2: $n-1$ people
Slot 3: $n-2$
⋮
Slot 60: $n-59$ people

Total # of configs:

$$\underbrace{n\cdot(n-1)\cdot(n-2)\cdot(n-3)\cdots(n-59)} = \frac{n!}{(n-60)!}$$

Search space: all ordered lists of 60 people

Multiply out: n^{60} + [stuff with powers less than 60]

Size: $O(n^{60})$

Good news: polynomial, not exponential

Bad news: still pretty bad

10 people: A B C D E F G H I J

3 PS $\quad \underline{10}\cdot\underline{9}\cdot\underline{8} = 720$

(A, B, C) (A, C, B) (B, A, C)

(C, B, A) (C, A, B) (B, C, A)

NFL Scheduling:

search space <u>per week</u> = all ways of putting 32 teams in pairs

For 17 weeks: do this 17 times

$\approx 6.5 \times 10^{294}$ (ignoring bye weeks)

$10^{80} \approx$ the # atoms in the universe

<u>Summary</u>: (brute force)

Pros: very easy to code — python package called "itertools"

fewer bugs

guaranteed optimal

find <u>all</u> optimal solutions

good for testing other methods against

(1) if you're coding a different guaranteed optimal method, check that it works correctly (for small data)

(2) if you're coding a non-guaranteed

optimal algorithm, testing how close does it get to optimal for small data

Topic 6 - Divide and Conquer (D+C)

D+C is an algorithmic paradigm (a problem solving approach) that roughly goes like:

1) Split the input data in half

2) Solve the problem on each half separately (recursion!)

3) Combine your two answers into one big answer.

Classic Example: Sorting a list

* Let's phrase this as a constraint satifaction problem

* Input: list of n numbers
* Search space: All orderings of n things. These are called "permutations" and the # of them is:

$$n \cdot (n-1) \cdot (n-2) \cdots\cdots 3 \cdot 2 \cdot 1$$

first item   2nd item   3rd item

$$= n!$$

* Goal: Find the rearrangement that puts things in the right order. (smallest to largest)

An "obvious" optimal algorithm: (greedy-ish)

n steps
- Search the whole list for the smallest element, and then put it first
- Find the smallest remaining thing, put it second.
- Repeat until done.

How long does this take? Each step has to go through the whole list.

n steps, go through whole list each step

$\rightarrow n \cdot n \rightarrow O(n^2).$

Fine for ≈ 100k things, but not ≈ 1B

Merge Sort $\quad \underbrace{O(n \cdot \underbrace{\log(n)}))}$

very slightly bigger than $O(n)$