Monday, Feb 15, 2023
Lecture # 13
MSSC 6000

Announcements

* HW 2 due Wednesday, Feb 22, 11:59pm
* Office Hours today, 2:30pm - 3:30pm, on Teams

Summary of Brute Force

Pros - very easy to code
fewer bugs
guaranteed optimal
finds all optimal solutions
good to test other methods against

Cons - SLOW, usually can only do
small cases

$\rightsquigarrow$ weighted interval / knapsack
(2^n)

n up to 20-30 in a few minutes

$\rightsquigarrow$ pairs of points $\binom{n}{2} \approx n^2$

$n = 100{,}000$ in a minute

How do we find optimal solutions?

(1) Don't bother — greedy algos * not optimal

(2) Wonder around the search space randomly, keeping track of the best thing you've seen so far. * not opt.

(3) Wander around the search space cleverly, keeping track of the best thing you've seen so far. * not optimal (metaheuristics)

(4) Check everything in the search space one-by-one. (Brute Force)

(5) Check or otherwise rule out everything in the search space. (divide-and-conquer, backtracking, branch-and-bound)

optimal, but sometimes fast and sometimes slow, not flexible

Two python lessons

1) List Slicing   2) Recursion

Topic 6 - Divide and Conquer

"Divide and Conquer" is an algorithmic paradigm that is roughly:

1) Split the input in half
2) Solve the problem on each half separately (recursion)
3) Combine the two answers into one big answer

Classic Example: Sorting a list (easy)

* We can phrase this as a constraint problem.
* Input: n numbers
* Search Space: All orderings of n things
  These are called permutations and the # of them is
  $$n! = n(n-1)(n-2)\cdots 3\cdot 2\cdot 1$$

$n! \approx \frac{n^n}{e^n}$ very big

Goal: Find the permutation of the #s

that is in increasing order.

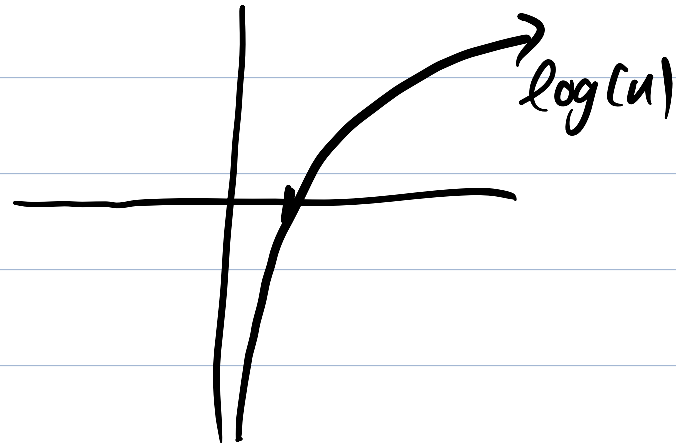
* Obvious optimal algorithm: (greedy-ish)
  - Find the smallest thing, put it first
  - Find the next smallest thing, put it second
  - and so on

How many steps does this take?
* Finding the k^{th} smallest thing takes $\approx n$ steps (have to search the whole list)
* We have to do this n times.
* $(n \text{ steps}) * (n \text{ times}) = O(n^2)$

1000 items
10 s to sort
$\rightsquigarrow$
2000 items
40 s to sort

n^2 vs. $(2n)^2 = 4n^2$

* Divide-and-Conquer can do this in $O(n \cdot \log(n))$.



$O(n \cdot \log(n))$ is <u>way</u> better than $O(n^2)$.

1) Split the input #s in half
2) Sort each half (recursively, by D+C'ing again)
3) Combine the two sorted halves into one big sorted list.

<u>Ex:</u> $\underbrace{3 \quad 19 \quad -7 \quad 2}$ $\underbrace{1 \quad 6 \quad 0 \quad -10}$

$\underbrace{3 \quad 19}$ $\underbrace{-7 \quad 2}$

$\underbrace{3}$ $\underbrace{19}$ $\underbrace{-7}$ $\underbrace{2}$

$\underbrace{3 \quad 19}$ $\underbrace{-7 \quad 2}$

To Be Continued!