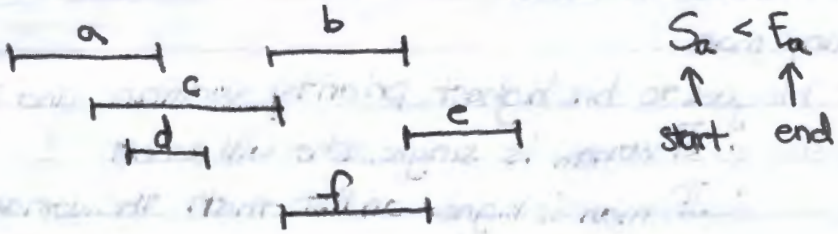
Lecture 3 1/12/16

Suppose we are given a set of intervals:



$S_a < F_a$

start end

Each interval represents a task, but because they overlap we have to select some to perform and others to skip. This is called the scheduling problem.

1. Start and end times are given and cannot be changed.
2. This also means that the lengths of each interval that are fixed.
3. No intersecting intervals are allowed.

Let's say I want to maximize the # of intervals that I can fit in my schedule

- Length doesn't matter
- There are no weights

How can we search exhaustively?

Given n items in the set, there are 2^n possible subsets (each element is either in or out). We can represent this as a n-bit binary string where each bit represents the inclusion of a particular item. (1 = in, 0 = out)

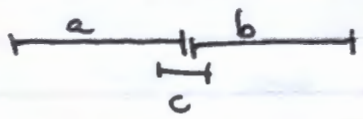
So we find the highest # of 1's that is valid string with

This works, but is very slow.

How do we solve using a greedy algorithm?

We can use length ~~by~~ as a selection mechanism. Begin with the smallest interval and add it, and continue to add the smallest remaining interval until there are no more valid intervals remaining

Does this work? No!



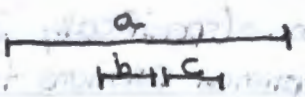
The alg will NOT produce the right answer!

Remember, the smaller a counterexample to an algorithm the better.

Let's now try a plane sweep. (Another greedy algorithm)

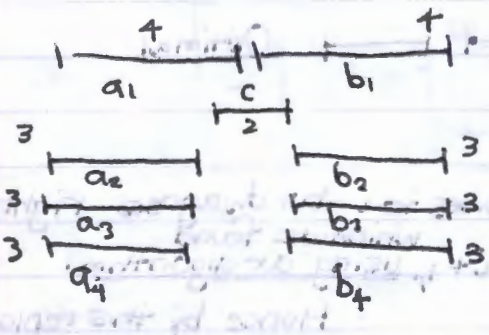
When a problem has geometric flavor, we can begin at one end and sweep to the end, doing something as we encounter certain events.

Does this work? No!



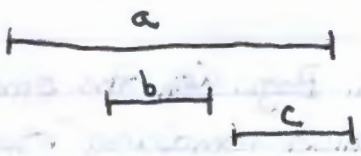
This alg will only select a, which is NOT the right answer!

What if we try the original greedy alg with interval conflicts as our counting criteria? Well, it isn't optimal, but this counterexample doesn't work.



It is encouraging to note that our new answer is harder to disprove.

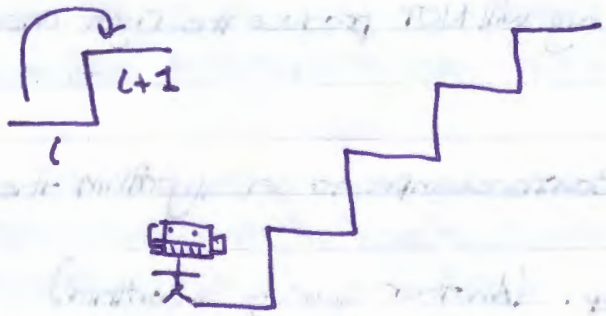
What if we did a greedy plane sweep version, and pick the first one that ends.



✓ Will pick b and c

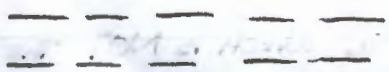
After a few more "failed" counterexamples, we consider trying to prove this.

Proof: Consider we have a robot and a staircase:



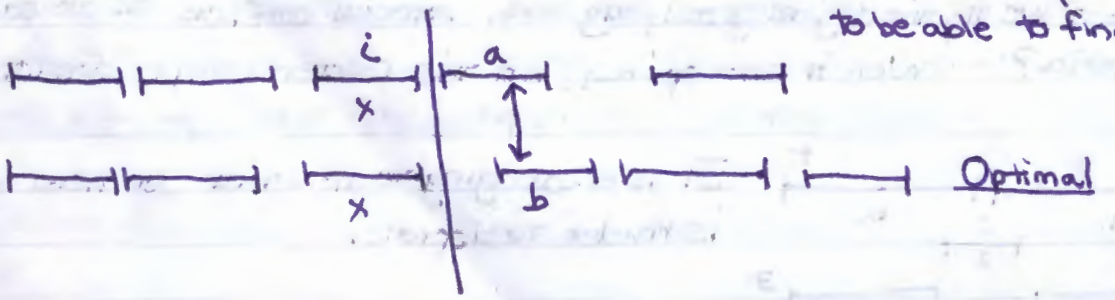
If we show the robot how to get from one step to the next (inductive step) and how to set it up (base case)

Note:



gives ~~$O(n^2)$~~ $O(2^n)$ time (specifically $2^{n/2}$) so we can't find all optimal solutions faster

Given that I have a solution that matches with the optimal up to i, we want to be able to find $i+1$



Say we have an optimal solution that matches to i but disagrees right after. We replace their $i+1$ with our $i+1$, which was found using our algorithm.

$S_a > E_x \quad S_b > E_x \quad E_a \leq E_b$

Hence by this replacement we still have an optimal solution!

Base Case: Before we start, both our interval and the potential optimal have 0 items and so are the same. In other words, adding an ↓ any interval will produce a valid i+1. ↓ with our alg

Runtime Analysis

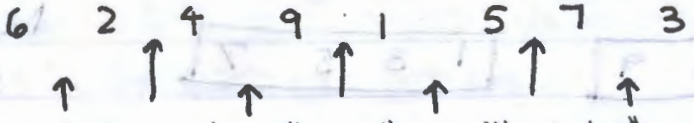
- Want to know what ends first
  Sort by end time $O(n \log n)$ → lower bound for general sorting

Note: store S's and E's in the same array, so when we pick the first one we effectively eliminate the overlapping start times.

Picking them all is $O(n)$. Overall it is $O(n \log n + n)$, → $O(n \log n)$

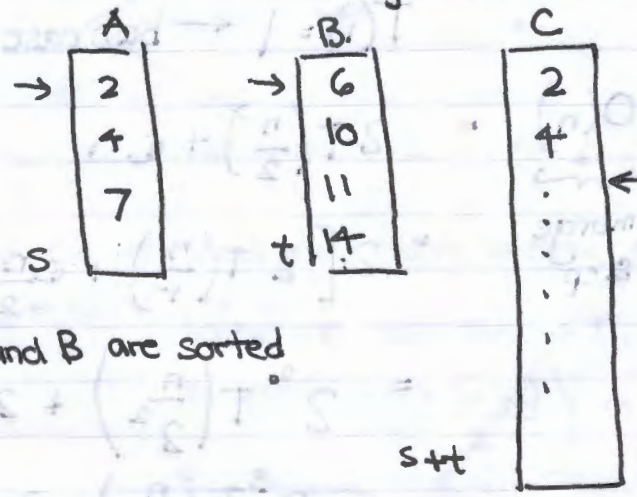
Lecture 4: 1/14/16

Assume you are given a set of integers....



Describe a sorting algorithm that will sort these in nondecreasing order

Now that we can merge these 8 sorted lists



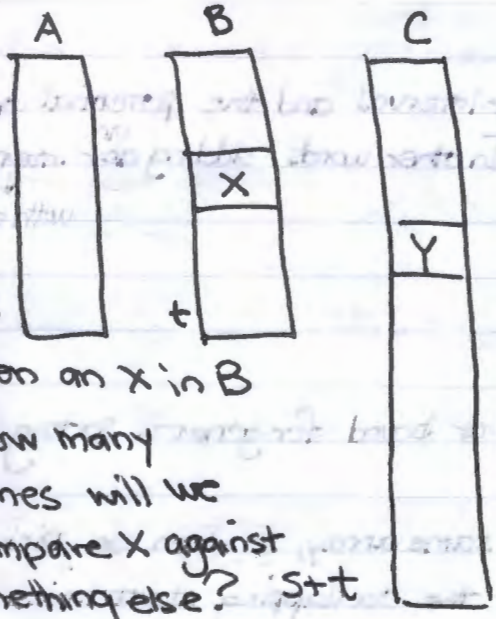
← $\min(\vec{A}, \vec{B})$ "merge step"

increment selected pointer

A and B are sorted.

$O(s+t)$

can we improve this time analysis?



for any Y in C, we spent 1 operation to obtain it, and so since we have a total of s+t items in C we have order

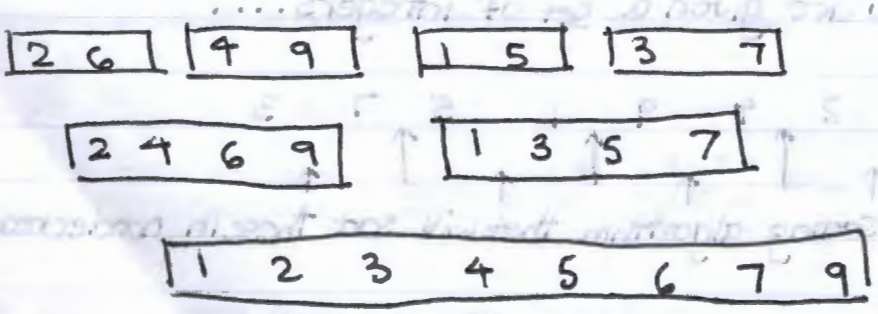
$O(s+t)$

Given an X in B

How many times will we compare X against something else? s+t

The whole set of numbers in A! (in the worst case)

Go back to the 8 lists... after a merge we get



Merge/Divide and Conquer Sort

Runtime analysis:

$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$ ← sorting halves, merge step

$T(1) = 1$ ← base case

$= 2T\left(\frac{n}{2}\right) + cn$

$= 2\left[2T\left(\frac{n}{4}\right) + c\frac{n}{2}\right] + cn$

$= 2^2 T\left(\frac{n}{2^2}\right) + 2cn$

$= 2^3 T\left(\frac{n}{2^3}\right) + 3cn$

$$= 2^i \, T\left(\frac{n}{2^i}\right) + i \, cn$$ ← generic step

We want to force

$$\frac{n}{2^i} \stackrel{?}{=} 1 \rightarrow 2^i = n$$

$$i = \log n$$

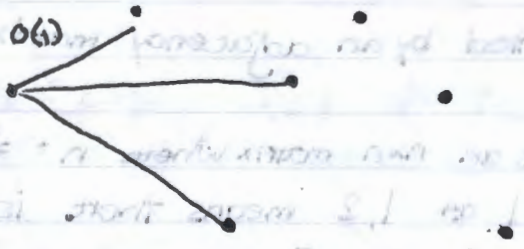
Plug in

$$2^{\log n} \cdot T\left(\frac{n}{2^{\log n}}\right) + \log n \cdot cn$$

$$= n + cn\log n$$

and so the runtime of our algorithm is $O(n \log n)$

Now consider a 2-D set of points. We want to find the closest 2 points. This is the closest pair problem.



Find the distances of a point to every other point, tracking the minimum as we go. Note we can account for repeated ops and eliminate them to get $O\left(\frac{n(n+1)}{2}\right)$ or we can keep it lose and end up with $O(n^2)$. The answer comes out to be the same.

12	4		1	9		11	6		18	3

min 12 move down list and compare each one, replacing as we find new min. Takes $n-1$ time

max same as min. $n-1$ time.

How do we perform better than $2n$?

Seems like we can't beat this, right? but this isn't true

Group in groups of 2, smaller is compared with min, larger is compared with larger.

Perform $\frac{n}{2}$ comparisons to find min and max in each group.

Then every single one is compared against something; n

Hence a better answer is $\frac{3}{2}n$

Lecture 5 Graphs 1/19/18

Graphs are defined by a set of vertices V and a set of edges E.

A graph can be represented by an adjacency matrix

	1	2	3
1			
2	□		
3			

It's an $n \times n$ matrix where n = # edges.
a 1 at 1,2 means there is an edge there.
Otherwise it is 0.

In an <u>undirected</u> graph this adjacency matrix is symmetric.