Algorithms in Bioinformatics

Divide & Conquer

Recap:

- HW due
  (feel free to submit Thurs.)
- Next HW - up by Thursday
- Today: Ch 7

Today: Divide + Conquer

Another variant of recursion:

- Divide: usual recursion, but into halves, or n/t size subproblems
- Conquer:
  - How to recombine

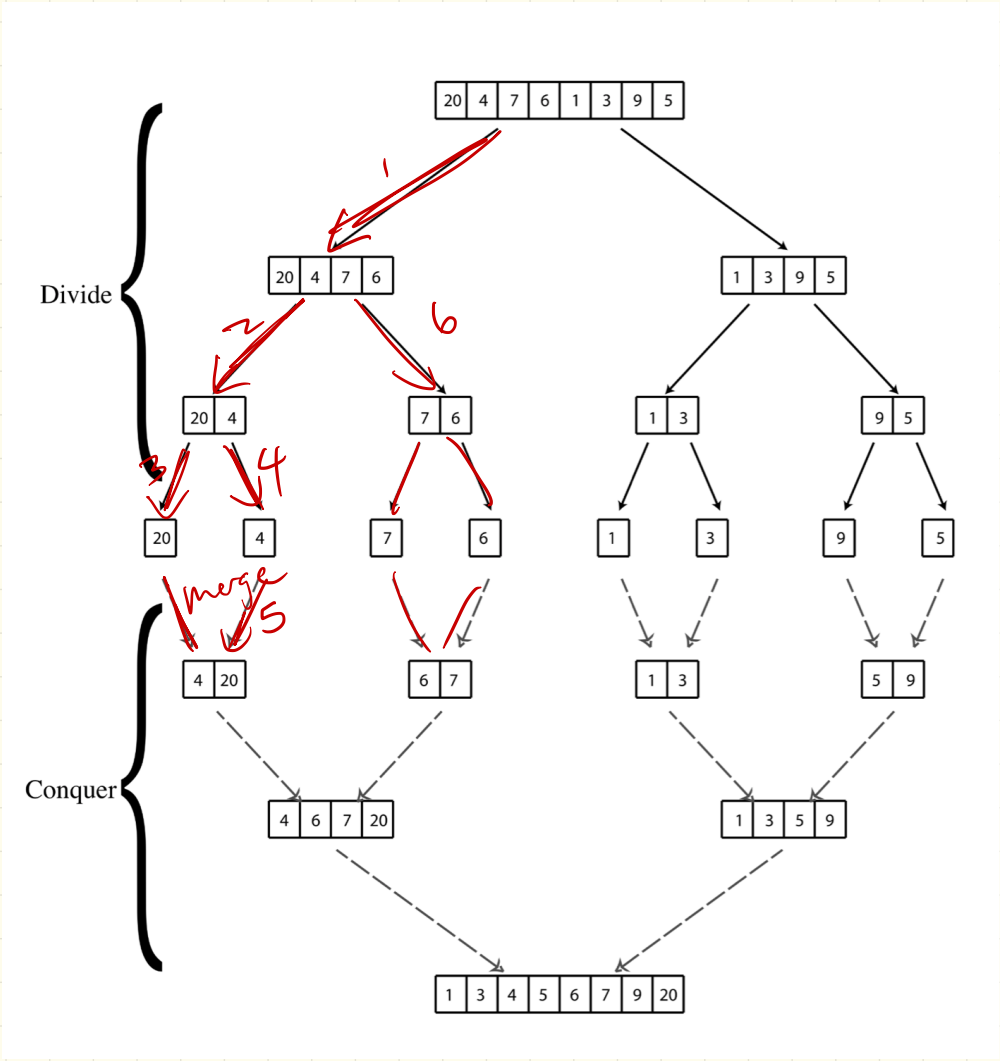
Classic example:

Merge Sort (video)

```
MERGESORT(c)
1  n ← size of c
2  if n = 1
3        return c
4  left ← list of first n/2 elements of c
5  right ← list of last n − n/2 elements of c
6  sortedLeft ← MERGESORT(left)
7  sortedRight ← MERGESORT(right)
8  sortedList ← MERGE(sortedLeft, sortedRight)
9  return sortedList
```

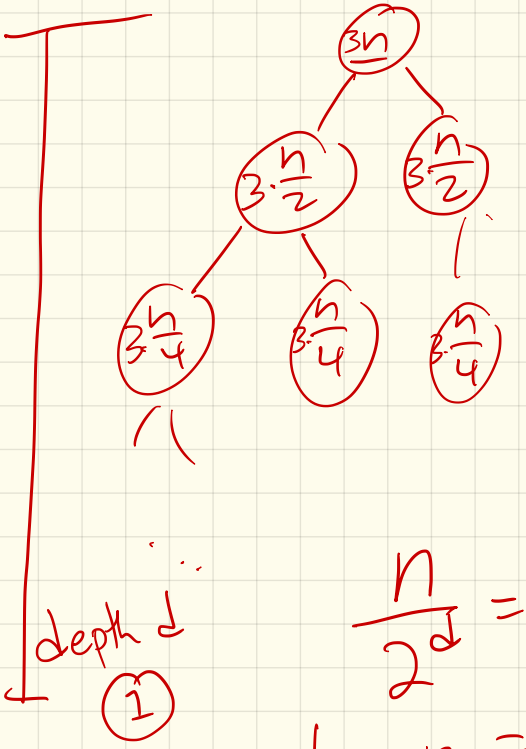
```
MERGE(a, b)
1   n1 ← size of a
2   n2 ← size of b
3   a_{n1+1} ← ∞
4   b_{n2+1} ← ∞
5   i ← 1
6   j ← 1
7   for k ← 1 to n1 + n2
8         if a_i < b_j
9               c_k ← a_i
10              i ←
11        else
12              c_k
13              j ←
14  return c
```

Picture :


Divide
Conquer
20 4 7 6 1 3 9 5
20 4 7 6
1 3 9 5
20 4
7 6
1 3
9 5
20
4
7
6
1
3
9
5
merge
4 20
6 7
1 3
5 9
4 6 7 20
1 3 5 9
1 3 4 5 6 7 9 20

Analysis:

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



amount of work per node on level i

$$\sum_{i=0}^{d} \left(3 \cdot \frac{n}{2^i}\right) \cdot 2^i$$

$$= O(3n \log n)$$

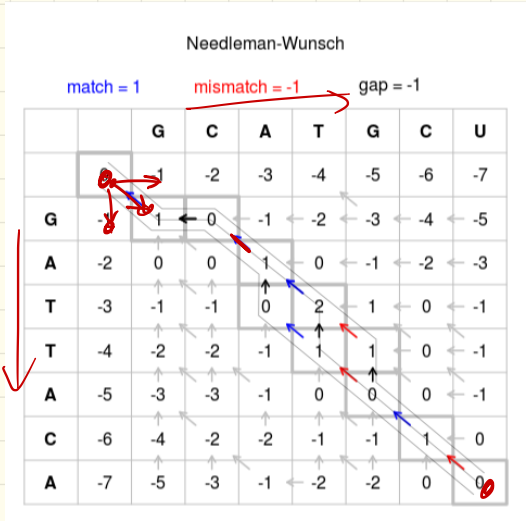
$$\frac{n}{2^d} = 1 \Rightarrow n = 2^d$$

$$\log_2 n = \log_2 2^d$$

$$= d \log_2 2$$

More biological: back to sequence alignment

Recall our alignment structure



(Can use arbitrary scores)

Recursion:

$$S_{i,j} = \max \begin{cases} S_{i-1,j} + \delta(i^{th}\text{ item}, -) \\ S_{i,j-1} + \delta(-, j^{th}) \\ S_{i-1,j-1} + \text{penalty} \end{cases}$$

Runtime & space:

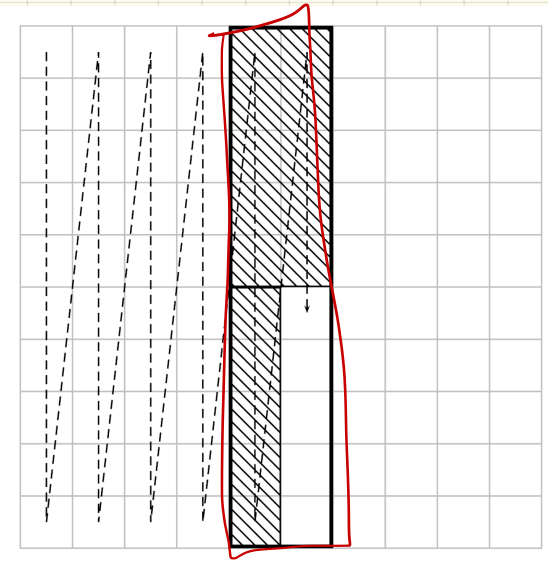
$O(m \cdot n)$

However: If just want score, can get space down to linear.

How?

only need up, left, & corner to fill in a cell

⇒ column by column: store 2 of them



Unfortunately – can't recover alignment.

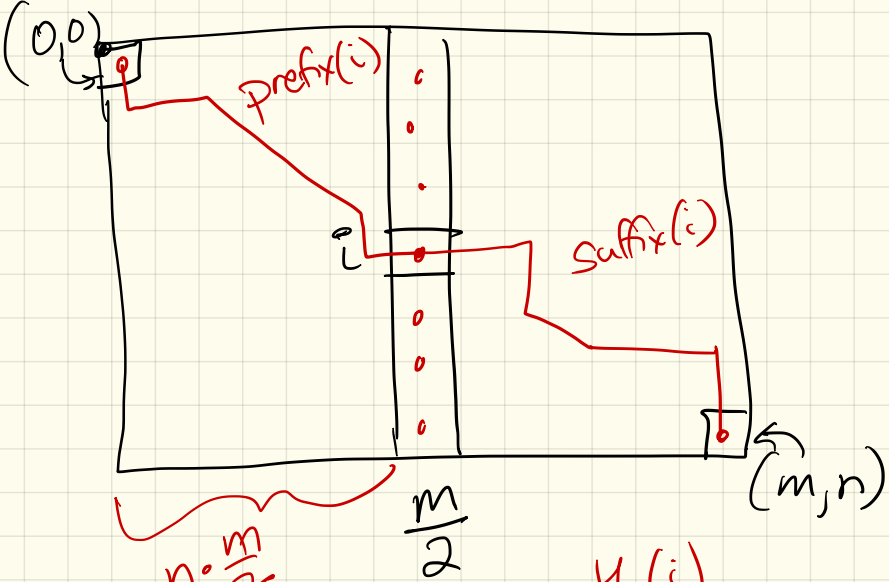
Why?

Think in terms of divide & conquer now:

define length(i) := cost of best alignment through cell $(i, \frac{m}{2})$

One of these must be the best path:

It goes from (0,0) to (m,n)



$\approx$ length(i)

length(i) = prefix(i) + suffix(i)

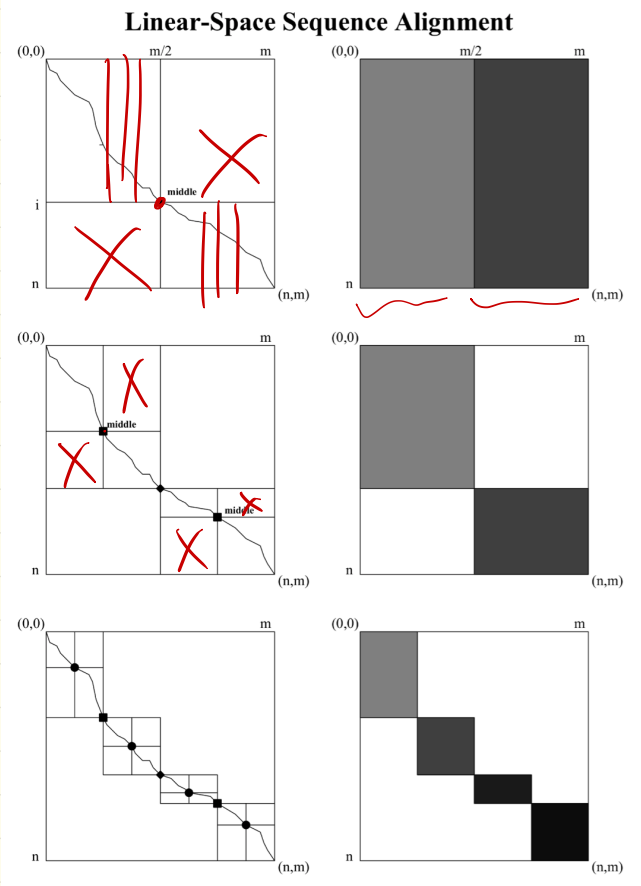
How to get prefix(i)?

chop off right half & run dynmanic pro.

↳ $\frac{nm}{2}$ time, $2 \cdot n$ space

Suffix(i): same, but "flip" the rectangle

Picture:



Time: $O(mn)$

Space: $O(n)$

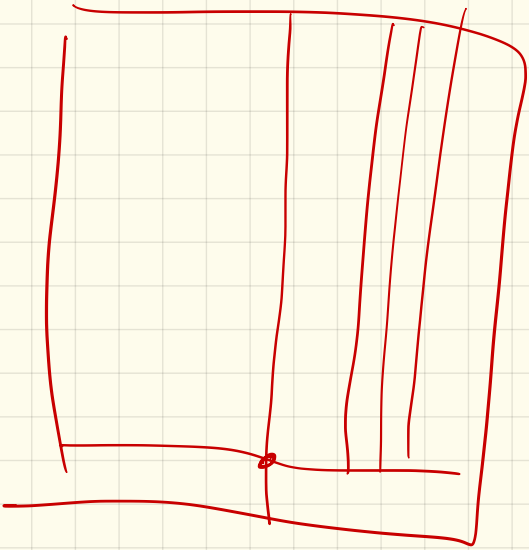
Runtime: mn time, $O(n)$ space

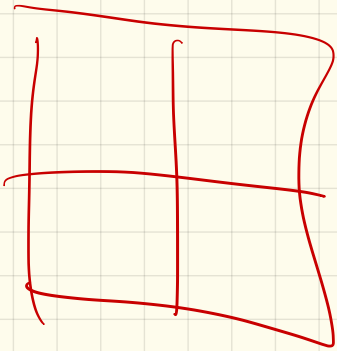
$+\frac{mn}{2}$ ← keep using

$+\frac{mn}{4}$

$< 2mn$

$\sum_{i=0}^{\infty} \frac{1}{2^i} < 2$





Block alignment (7.3)

Consider $\vec{u}$ & $\vec{v}$, 2 DNA sequences, divided into blocks of length t

$u = \underbrace{u_1 u_2 \cdots u_t}\ \underbrace{u_{t+1} \cdots u_{2t}}\ u_{2t+1} \cdots u_{k \cdot t}$

$v = \underbrace{v_1 \cdots v_t}\ \underbrace{v_{t+1} \cdots v_{2t}} \cdots v_{k \cdot t}$

(ie if $t=3$, divide into codons)

We will seek an alignment where every block in $\vec{u}$ is either aligned with another entire block or is inserted/deleted altogether

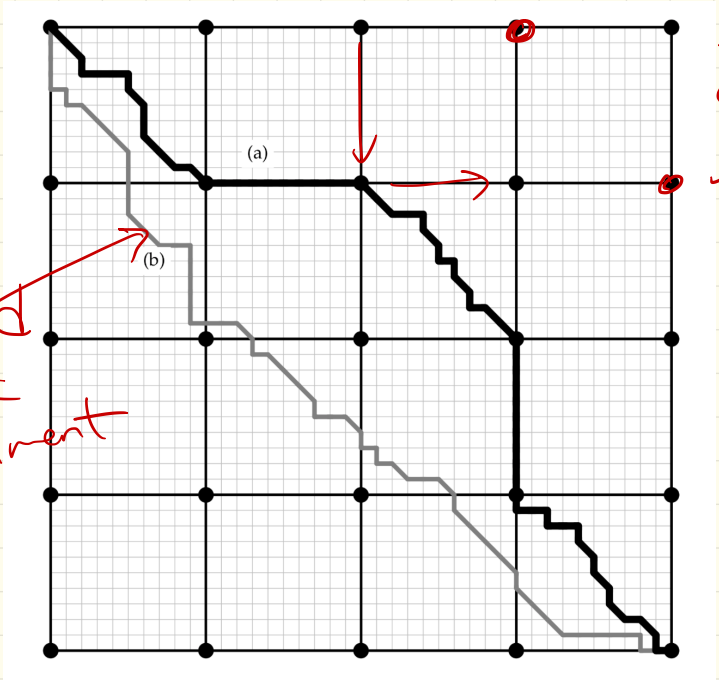
Goal: Find best such alignment.

Block Alignment Problem:
Find the longest block path through an edit graph.

Input: Two sequences, **u** and **v** partitioned into blocks of size t.

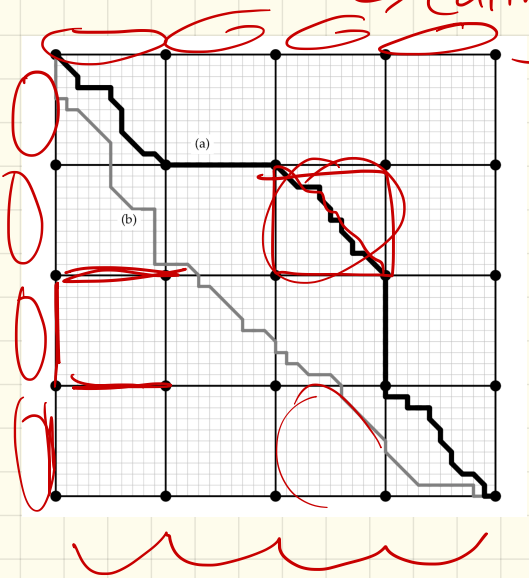
Output: The block alignment of **u** and **v** with the maximum score (i.e., the longest block path through the edit graph).

Picture: $n = m = 40$
$t = 10$



Solution: Try each pair of blocks & compute alignment score $\beta_{i,j}$

↳ calling global alignment



$\frac{n}{t} \times \frac{n}{t}$ time $= O\left(\frac{n^2}{t^2}\right)$

Then $S_{i,j} = \max$ $\begin{cases} S_{i-1,j} - G_{block} \\ S_{i,j-1} - G_{block} \\ S_{i-1,j-1} - \beta_{i,j} \end{cases}$

↑ goes from 1 to $\frac{n}{t}$

Runtime:

each block $B_{i,j}$: $t \times t$ block

$\Rightarrow O(t^2)$

$\frac{n}{t} \times \frac{n}{t}$ pairs

$\Rightarrow O(\frac{n^2}{t^2})$

$\Rightarrow O(\frac{n^2}{t^2}) \cdot O(t^2)$

$\Rightarrow O(n^2)$

Interesting speedup: (4-Russians technique)

Let $t = \log_4 n$

Don't compute all $\frac{n}{t} \times \frac{n}{t}$ subproblems

Instead, consider all possible length t nucleotides

↳ $4^t \times 4^t$ total pairs

Store the cost of aligning in a table

Runtime: If $t \approx \frac{\log n}{4}$,

$4^t \times 4^t = n^{1/2} \times n^{1/2} = O(n)$

Resulting table, Score, has only $4^t \times 4^t$ entries:

$O(n)$ size

Runtime:

- $O(n)$ entries in table
- each table entry takes $O(\log^2 n)$ time

Then same dynamic program:

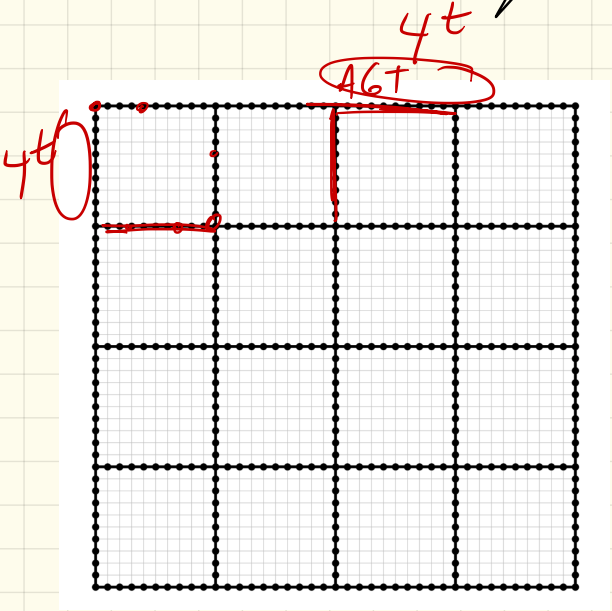
$$s_{i,j} = \max \begin{cases} s_{i-1,j} - \sigma_{block} \\ s_{i,j-1} - \sigma_{block} \\ s_{i-1,j-1} + Score(i\text{th block of } \mathbf{v},\ j\text{th block of } \mathbf{u}) \end{cases}$$

Then $\frac{n}{t} \times \frac{n}{t}$ time for dynamic program, plus $O(\log n)$ lookup time in table (since need to find where it is stored)

Final runtime: $O\left(\frac{n^2}{\log n}\right)$

This can actually be used for LCS problem, also.

Idea: break up into $t \times t$ squares again



n/t rows

n/t columns

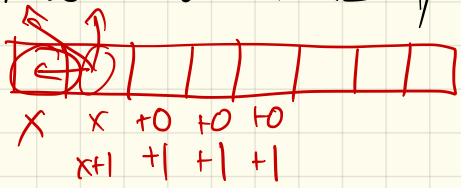
total: n^2/t

Can't just use top left & bottom right corner on each

Brute force:

$4^t \times 4^t$ sequences, and all possible scores for 1st column + row

1st row: what is possible?



So: encode as binary vector instead $\Big] 2^t$

Size of table:

$4^{t-\log n} \times 4^t \times 2^t \times 2^t = O(n)$

$\hookrightarrow O\left(\frac{n^2}{\log n}\right)$ ✗