Objectives

- Dynamic Programming: Applications in computational biology
  - Sequence Alignment: space improvement

String Similarity

- How similar are two strings?
  - ocurrance
  - occurrence
- Measurements
  - Gap (-): add a letter
  - Mismatch

Which is the best alignment?

o	c	u	r	r	a	n	c	e	-
o	c	c	u	r	r	e	n	c	e

6 mismatches, 1 gap

o	c	-	u	r	r	a	n	c	e
o	c	c	u	r	r	e	n	c	e

1 mismatch, 1 gap

o	c	-	u	r	r	-	a	n	c	e
o	c	c	u	r	r	e	-	n	c	e

0 mismatches, 3 gaps

Sequence Alignment: Problem Structure

Gaps for remainder of Y

Ran out of 1st string

$$OPT(i,j) = \begin{cases} j\delta & \text{if } i = 0 \\ \min\begin{cases} \alpha_{x_i y_j} + OPT(i-1, j-1) \\ \delta + OPT(i-1, j) \\ \delta + OPT(i, j-1) \end{cases} & \text{otherwise} \\ i\delta & \text{if } j = 0 \end{cases}$$

Ran out of 2nd string

Gaps for remainder of X

Sequence Alignment: Algorithm

Cost parameters

```
Sequence-Alignment(m, n, $x_1x_2...x_m$, $y_1y_2...y_n$, $\delta$, $\alpha$)
   for i = 0 to m
      M[0, i] = i$\delta$
   for j = 0 to n
      M[j, 0] = j$\delta$

   for i = 1 to m
      for j = 1 to n
         M[i, j] = min($\alpha$[$x_i$, $y_j$] + M[i-1, j-1],
                        $\delta$ + M[i-1, j],
                        $\delta$ + M[i, j-1])
   return M[m, n]
```

Costs?

Sequence Alignment: Analysis

```
Sequence-Alignment(m, n, $x_1x_2...x_m$, $y_1y_2...y_n$, $\delta$, $\alpha$)
   for i = 0 to m
      M[0, i] = i$\delta$
   for j = 0 to n
      M[j, 0] = j$\delta$
                                        O(mn)
   for i = 1 to m
      for j = 1 to n
         M[i, j] = min($\alpha$[$x_i$, $y_j$] + M[i-1, j-1],
                        $\delta$ + M[i-1, j],
                        $\delta$ + M[i, j-1])
   return M[m, n]
```

Example

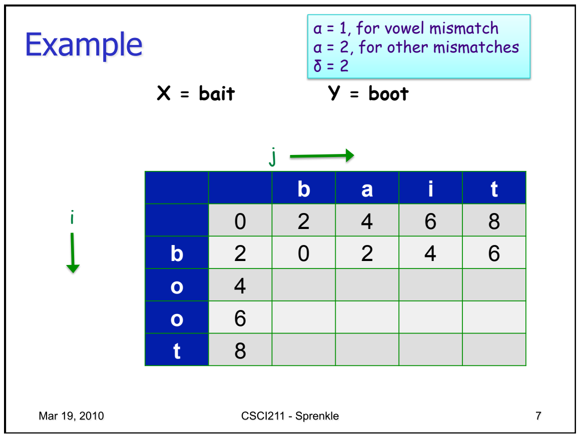
α = 1, for vowel mismatch
α = 2, for other mismatches
δ = 2

X = bait   Y = boot

$j \longrightarrow$

$i \downarrow$

		b	a	i	t
	0	2	4	6	8
b	2				
o	4				
o	6				
t	8				



Example

α = 1, for vowel mismatch
α = 2, for other mismatches
δ = 2

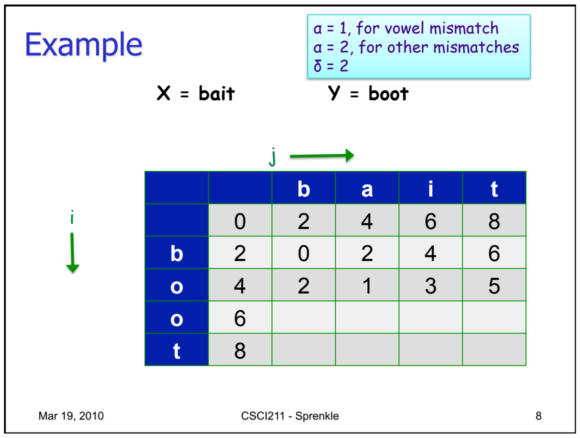
X = bait Y = boot

j →

i ↓

		b	a	i	t
	0	2	4	6	8
b	2	0	2	4	6
o	4				
o	6				
t	8				

Mar 19, 2010 CSCI211 - Sprenkle 7



Example

α = 1, for vowel mismatch
α = 2, for other mismatches
δ = 2

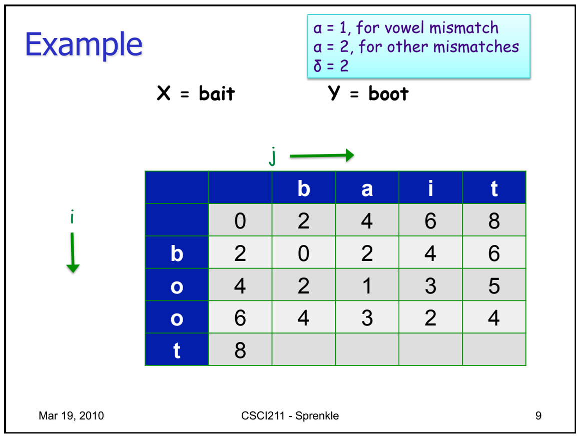
X = bait Y = boot

j →

i ↓

		b	a	i	t
	0	2	4	6	8
b	2	0	2	4	6
o	4	2	1	3	5
o	6				
t	8				

Mar 19, 2010 CSCI211 - Sprenkle 8



Example

α = 1, for vowel mismatch
α = 2, for other mismatches
δ = 2

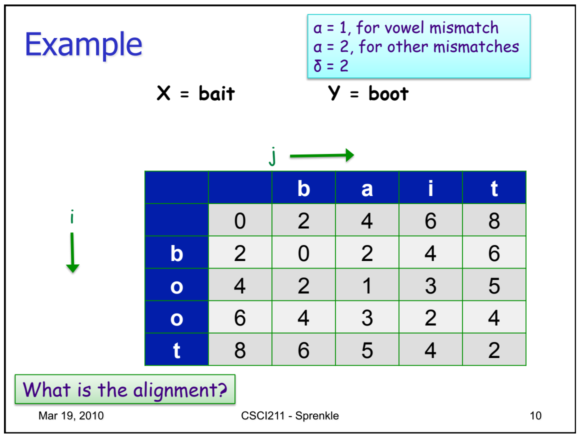
X = bait Y = boot

j →

i ↓

		b	a	i	t
	0	2	4	6	8
b	2	0	2	4	6
o	4	2	1	3	5
o	6	4	3	2	4
t	8				

Mar 19, 2010 CSCI211 - Sprenkle 9



Example

α = 1, for vowel mismatch
α = 2, for other mismatches
δ = 2

X = bait Y = boot

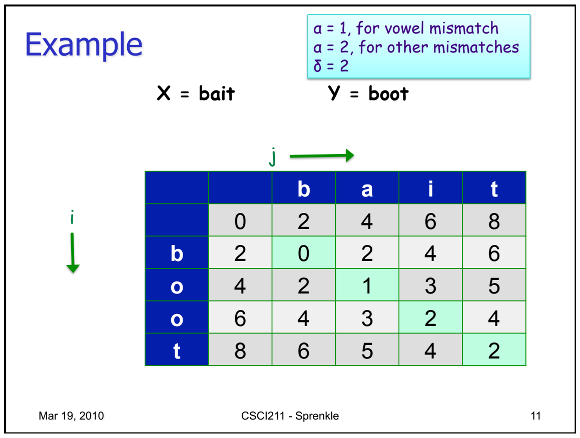
j →

i ↓

		b	a	i	t
	0	2	4	6	8
b	2	0	2	4	6
o	4	2	1	3	5
o	6	4	3	2	4
t	8	6	5	4	2

What is the alignment?

Mar 19, 2010 CSCI211 - Sprenkle 10



Example

α = 1, for vowel mismatch
α = 2, for other mismatches
δ = 2

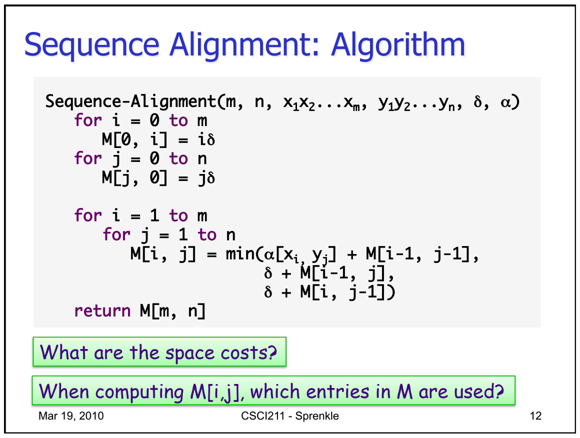
X = bait Y = boot

j →

i ↓

		b	a	i	t
	0	2	4	6	8
b	2	0	2	4	6
o	4	2	1	3	5
o	6	4	3	2	4
t	8	6	5	4	2

Mar 19, 2010 CSCI211 - Sprenkle 11



Sequence Alignment: Algorithm

```
Sequence-Alignment(m, n, x1x2...xm, y1y2...yn, δ, α)
   for i = 0 to m
      M[0, i] = iδ
   for j = 0 to n
      M[j, 0] = jδ

   for i = 1 to m
      for j = 1 to n
         M[i, j] = min(α[xi, yj] + M[i-1, j-1],
                       δ + M[i-1, j],
                       δ + M[i, j-1])
   return M[m, n]
```

What are the space costs?

When computing M[i,j], which entries in M are used?

Mar 19, 2010 CSCI211 - Sprenkle 12

Sequence Alignment: Analysis

```
Sequence-Alignment(m, n, x1x2...xm, y1y2...yn, δ, α)
   for i = 0 to m
      M[0, i] = iδ
   for j = 0 to n
      M[j, 0] = jδ                                  Space Cost: O(mn)

   for i = 1 to m
      for j = 1 to n
         M[i, j] = min(α[xi, yj] + M[i-1, j-1],
                       δ + M[i-1, j],
                       δ + M[i, j-1])
   return M[m, n]
```

Observation: to calculate the current value, we only need the row above us and the entry to the left

SEQUENCE ALIGNMENT IN LINEAR SPACE

Sequence Alignment: O(m) Space

- Collapse into an m x 2 array
  - M[i,0] represents previous row; M[i,1] -- current

```
Space-Efficient-Alignment(m, n, x1x2...xm, y1y2...yn, δ, α)
   for i = 0 to m          # initialize first row
      M[i, 0] = iδ
   for j = 1 to n
      M[0, 1] = jδ         # first gap

    for i = 1 to m
        M[i, 1] = min(α[xi, yj] + M[i-1, 0],
                      δ + M[i, 0],
                      δ + M[i-1, 1])
    for i = 1 to m         # copy current row into previous
        M[i, 0] = M[i, 1]
   return M[m, 1]
```

Any drawbacks?

Sequence Alignment: O(m) Space

- Collapse into an m x 2 array
  - M[i,0] represents previous row; M[i,1] -- current

```
Space-Efficient-Alignment(m, n, x1x2...xm, y1y2...yn, δ, α)
   for i = 0 to m  # initialize first row
      M[i, 0] = iδ
   for j = 1 to n
      M[0, 1] = jδ         # first gap

    for i = 1 to m
        M[i, 1] = min(α[xi, yj] + M[i-1, 0],
                      δ + M[i, 0],
                      δ + M[i-1, 1])
    for i = 1 to m     # copy current row into previous
        M[i, 0] = M[i, 1]
   return M[m, 1]
```

Finds optimal value but will not be able to find alignment

Why Do We Care About Space?

- For English words or sentences, probably doesn't matter
- Matters for Biological sequence alignment
  - Consider: 2 strings with 100,000 symbols each
    - Processor can do 10 billion primitive operations
    - BUT dealing with a 10 GB array

Sequence Alignment: Linear Space

- Can we avoid using quadratic space?
  - Optimal value in O(m) space and O(mn) time.
    - Compute OPT(i, •) from OPT(i-1, •)
    - BUT, no simple way to recover alignment itself
- Theorem. [Hirschberg 1975] Optimal alignment in O(m + n) space and O(mn) time.
  - Clever combination of *divide-and-conquer* and *dynamic programming*
  - Inspired by idea of Savitch from complexity theory

Recall Our Example

α = 1, for vowel mismatch
α = 2, for other mismatches
δ = 2

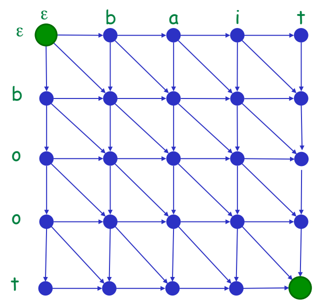
X = bait Y = boot

j →

i ↓

		b	a	i	t
	0	2	4	6	8
b	2	0	2	4	6
o	4	2	1	3	5
o	6	4	3	2	4
t	8	6	5	4	2

Mapping to a Graph Problem

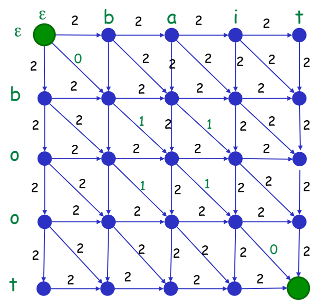


α = 1, for vowel mismatch
α = 2, for other mismatches
δ = 2

- Horizontal and vertical edges cost δ
- Diagonal edges cost α

Goal: Find shortest path from top-left to bottom-right

Mapping to a Graph Problem



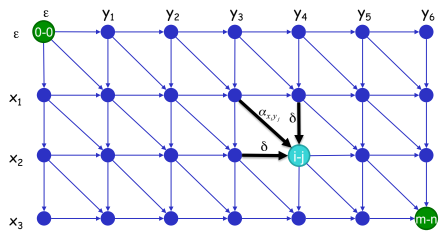
α = 1, for vowel mismatch
α = 2, for other mismatches
δ = 2

- Horizontal and vertical edges cost δ
- Diagonal edges cost α

Goal: Find shortest path from top-left to bottom-right

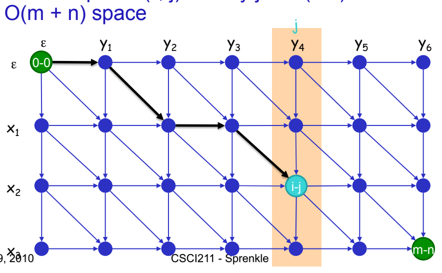
Sequence Alignment: Forward

- Edit distance graph (start)
  - Let f(i, j) be shortest path from (0,0) to (i, j)
  - Observation: f(i, j) = OPT(i, j)



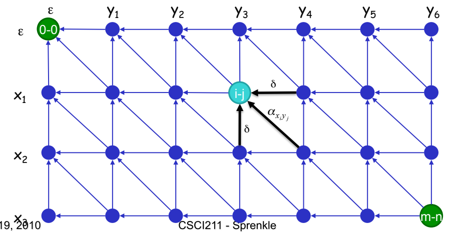
Sequence Alignment: Forward

- Edit distance graph (start)
  - Let f(i, j) be shortest path from (0,0) to (i, j)
  - Can compute f (*, j) for any j in O(mn) time and O(m + n) space



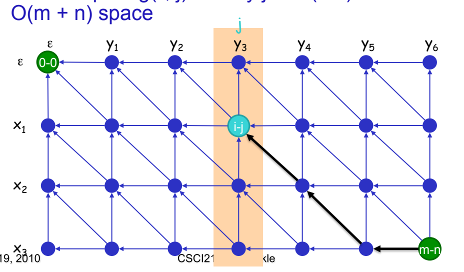
Sequence Alignment: Backward

- Edit distance graph (end)
  - Let g(i, j) be shortest path from (m, n) to (i, j)
  - Can compute by reversing the edge orientations and inverting the roles of (0, 0) and (m, n)



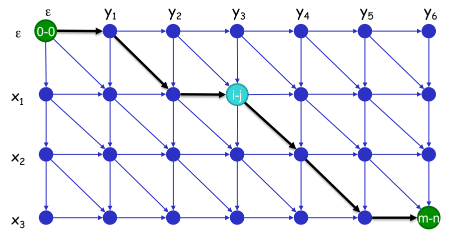
Sequence Alignment: Backward

- Edit distance graph
  - Let g(i, j) be shortest path from (m, n) to (i, j) (end)
  - Can compute g(*, j) for any j in O(mn) time and O(m + n) space



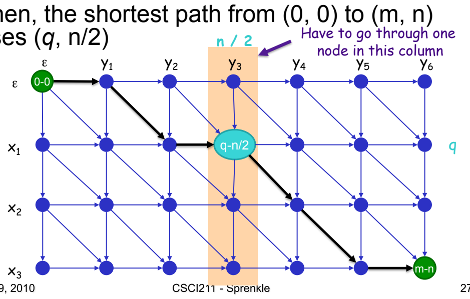
Sequence Alignment: Linear Space

- Observation. The cost of the shortest path that uses *(i, j)* is f(i, j) + g(i, j)



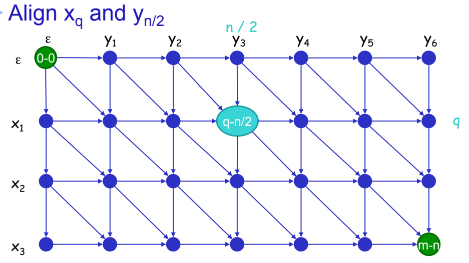
Sequence Alignment: Linear Space

- Let *q* be an index that minimizes f(*q*, n/2) + g(*q*, n/2)
- Then, the shortest path from (0, 0) to (m, n) uses (*q*, n/2)



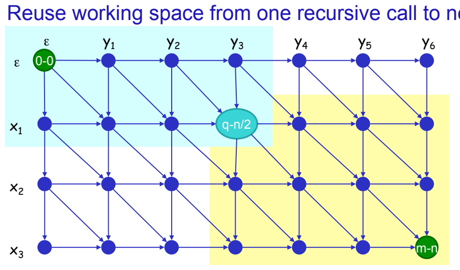
Sequence Alignment: Linear Space

- Divide: find index q that minimizes f(q, n/2) + g(q, n/2) using DP
  - Align x_q and $y_{n/2}$



Sequence Alignment: Linear Space

- Conquer: recursively compute optimal alignment in each piece
  - Reuse working space from one recursive call to next

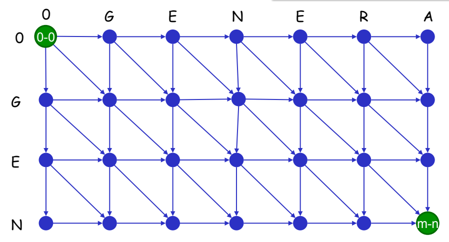


Divide and Conquer Sequence Alignment

```
Create graph, label edges with weights

P contains node on shortest corner-to-corner path

Divide-and-Conquer-Alignment(X, Y)

Divide-and-Conquer-Alignment (X, Y):
    m = length of X
    n = length of Y
    if m <= 2 or n <= 2
        compute optimal alignment using Alignment(X, Y)
        return
    Space-Efficient-Alignment(X, Y[1:n/2])
    Backward-Space-Efficient-Alignment(X, Y[n/2+1:n])
    q = index that minimizes f(q, n/2) + g(q, n/2)
    add(q, n/2) to P
    Divide-and-Conquer-Alignment(X[1:q],Y[1:n/2])
    Divide-and-Conquer-Alignment(X[q:m],Y[(n/2):n])
    return P
```

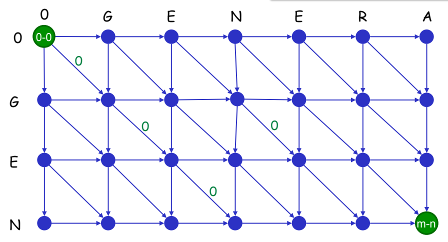
Example

α = 1, for vowel mismatch
α = 2, for other mismatches
δ = 2



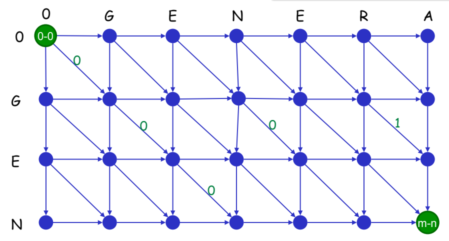
Example

α = 1, for vowel mismatch
α = 2, for other mismatches
δ = 2

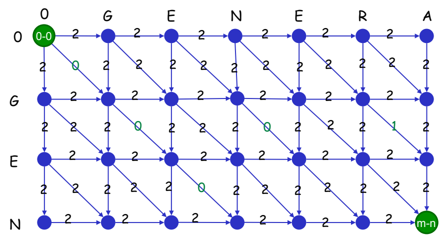


Example

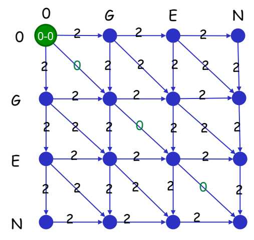
α = 1, for vowel mismatch
α = 2, for other mismatches
δ = 2



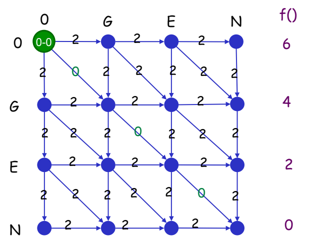
Example



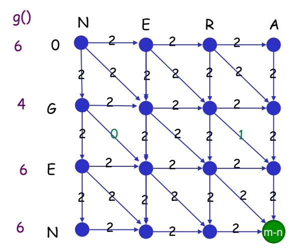
Space-efficient alignment: Left



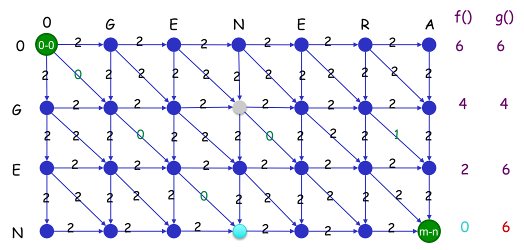
Space-efficient alignment: Left



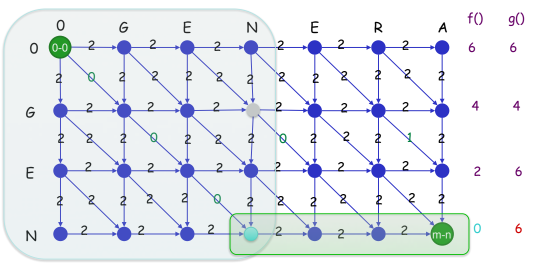
Backwards Space Efficient



Example



Example



Divide and Conquer Sequence Alignment: Analysis

```
P contains node on shortest corner-to-corner path
Divide-and-Conquer-Alignment (X, Y)
    m = length of X
    n = length of Y
    if m <= 2 or n <= 2
        compute optimal alignment using Alignment(X, Y)
        return
    Space-Efficient-Alignment(X, Y[1:n/2])
    Backward-Space-Efficient-Alignment(X, Y[n/2+1:n])
    q = index that minimizes f(q, n/2) + g(q, n/2)
    add(q, n/2) to P
    Divide-and-Conquer-Alignment(X[1:q],Y[1:n/2])
    Divide-and-Conquer-Alignment(X[q:m],Y[(n/2):n])
    return P
```

What is the recurrence relation?

Sequence Alignment: Running Time Analysis Warmup

- Theorem. Let T(m, n) = max running time of algorithm on strings of length at most m and n. T(m, n) = O(mn log n).

$$T(m,n) \le 2T(m, n/2) + O(mn) \Rightarrow T(m,n) = O(mn \log n)$$

- Remark. Analysis is not tight because sub-problems are of size (q, n/2) and (m - q, n/2).

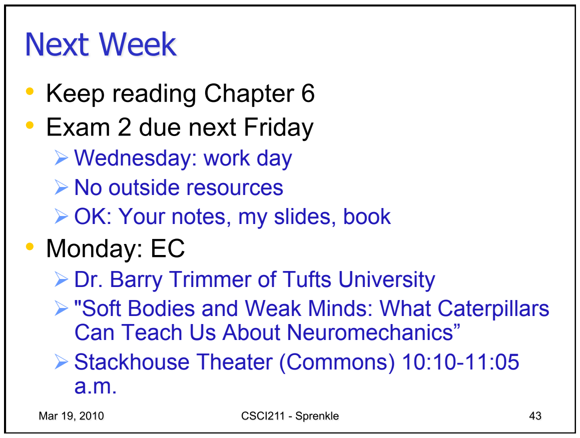
Sequence Alignment: Running Time Analysis

- Theorem. Let T(m, n) = max running time of algorithm on strings of length m and n. T(m, n) = O(mn)
- Recurrence Relation:

$$\begin{aligned} T(m, 2) &\le cm \\ T(2, n) &\le cn \\ T(m, n) &\le cmn + T(q, n/2) + T(m-q, n/2) \end{aligned}$$

- Solve using substitution:

$$\begin{aligned} T(m,n) &\le T(q,n/2) + T(m-q,n/2) + cmn \\ &\le 2cqn/2 + 2c(m-q)n/2 + cmn \\ &= cqn + cmn - cqn + cmn \\ &= 2cmn \end{aligned}$$



Next Week

- Keep reading Chapter 6
- Exam 2 due next Friday
  - Wednesday: work day
  - No outside resources
  - OK: Your notes, my slides, book
- Monday: EC
  - Dr. Barry Trimmer of Tufts University
  - "Soft Bodies and Weak Minds: What Caterpillars Can Teach Us About Neuromechanics”
  - Stackhouse Theater (Commons) 10:10-11:05 a.m.

Mar 19, 2010 CSCI211 - Sprenkle 43