CS3000: Algorithms & Data
Jonathan Ullman

Midterm Info

Lecture 8:

- Dynamic Programming: RNA Folding, Practice

Feb 3, 2020

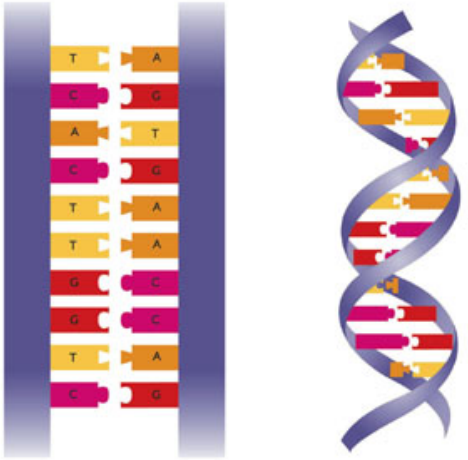
Dynamic Programming Examples

- Interval Scheduling — Choose a subset; One variable recurrence
- Segmented Least Squares — Partition the line into intervals
- Knapsack — Choose a subset; Two variable recurrence
- Edit Distance / Alignments — Choose the last piece of the alignment
- Concert Scheduling — Choose a subset; One variable recurrence
- RNA Folding — Pair up items; Two variable recurrence

RNA Folding

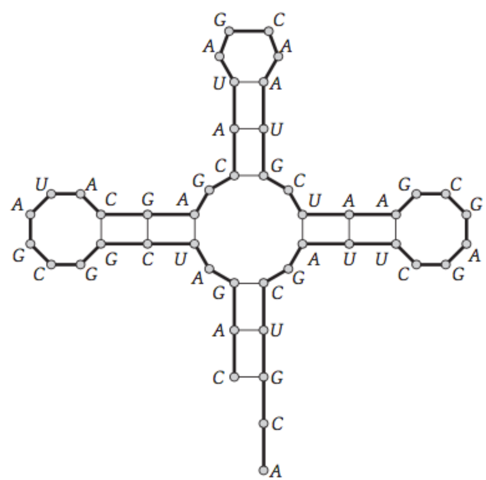
DNA

- DNA is a string of four bases **{A,C,G,T}**
- Two complementary strands of DNA stick together and form a **double helix**
  - **A—T** and **C—G** are complementary pairs

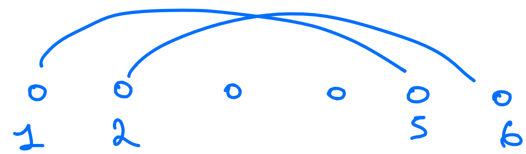


RNA Folding

- RNA is a string of four bases **{A,C,G,U}**
- A single RNA strand sticks to itself and folds into complex structures
  - **A—U** and **C—G** are complementary pairs

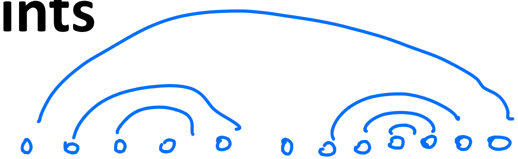


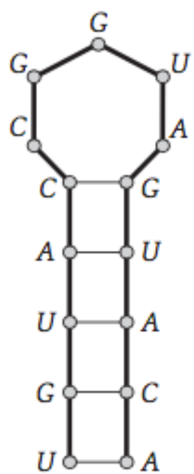
RNA Folding

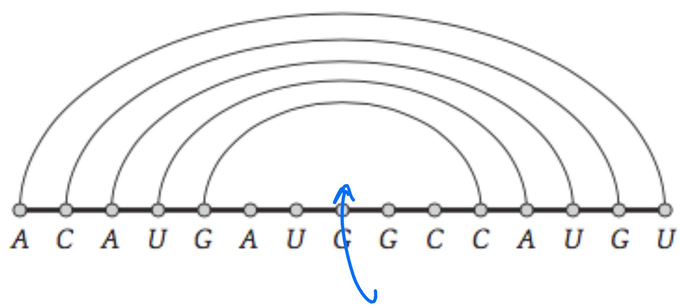


no crossing

- RNA strand will try to **minimize energy** (form the most bonds) subject to **constraints**



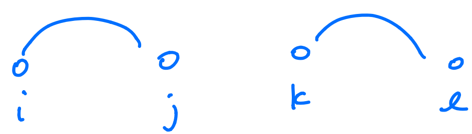




no pairs too close together

RNA Folding

- RNA is a string of bases $\boldsymbol{b_1, \ldots, b_n} \in \{\boldsymbol{A, C, G, U}\}$
- The structure is given by a set of **bonds** S consisting of pairs (i, j) with $i < j$
  - **(Complements)** Only $A - U$ or $C - G$ can be paired
  - **(Matching)** No base b_i is in two pairs in S
  - **(No Sharp Turns)** If $(i, j) \in S$, then $i < j - 4$
  - **(Non-Crossing)** If $(i, j), (k, \ell) \in S$ then it cannot be the case that $i < k < j < \ell$



RNA Folding

- **Input:** RNA sequence $\boldsymbol{b_1}, \dots, \boldsymbol{b_n} \in \{A, C, G, U\}$
- **Output:** A set of pairs $S \subseteq \{1, \dots, n\} \times \{1, \dots, n\}$
  - **Goal:** maximize the size of S

  - **(Complements)** Only $A - U$ or $C - G$ can be paired
  - **(Matching)** No base b_i is in two pairs in S
  - **(No Sharp Turns)** If $(i, j) \in S$, then $i < j - 4$
  - **(Non-Crossing)** If $(i, j), (k, \ell) \in S$ then it cannot be the case that $i < k < j < \ell$

Dynamic Programming

- Let O be the optimal set of pairs for $b_1 \cdots b_n$
- **Case 1:** O does not include any pair involving n

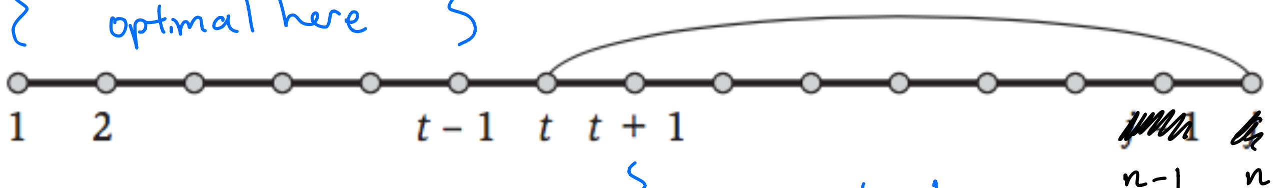
  O is the optimal solution using $b_1, \ldots, b_{n-1}$

- **Case 2:** O has n pair with some $t < n - 4$ in O

  O is (t, n) + the optimal solution using $b_1, \ldots, b_{t-1}$

  + the optimal solution using $b_{t+1}, \ldots, b_{n-1}$

{ optimal here }

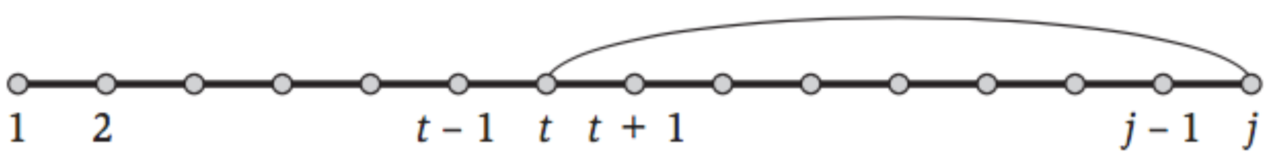


{ optimal here }

Dynamic Programming

- Let $O_{i,j}$ be the optimal set of pairs for $b_i \cdots b_j$
- **Case 1:** $O_{i,j}$ does not include any pair involving j

- **Case 2:** $O_{i,j}$ has j pair with some $t < j - 4$ in O



Dynamic Programming

$1 \le i \le n$ $i+4 < j \le n$

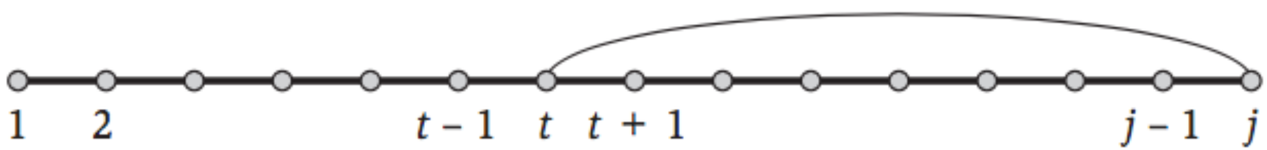
$\binom{n}{2}$ subproblems

- Let $\mathrm{OPT}(i, j)$ be the opt. **number** of ~~pairs~~ bonds for $b_i \cdots b_j$
- **Case 1:** j pairs with nothing

  $\mathrm{OPT}(i,j) = \mathrm{OPT}(i, j-1)$

- **Case 2:** j pairs with $t < j - 4$

  $\mathrm{OPT}(i,j) = 1 + \mathrm{OPT}(i, t-1) + \mathrm{OPT}(t+1, j-1)$



Dynamic Programming

$\max\{A, B\}$

$\max_t \{f(t)\}$

- Let $\mathrm{OPT}(i,j)$ be the opt. **number** of pairs for $b_i \cdots b_j$
- **Case 1:** j pairs with nothing
- **Case 2:** j pairs with $t < j - 4$

Recurrence:

$1+$

$$\mathrm{OPT}(i,j) = \max\{\mathrm{OPT}(i, j-1), \max\{\mathrm{OPT}(i, t-1) + \mathrm{OPT}(t+1, j-1)\}\}$$

Maximum over all t such that
- $i \le t < j - 4$
- b_t, b_j are compatible bases

Base Cases:

$\mathrm{OPT}(i,j) = 0$ if $i \ge j - 4$

Filling the Table

1:9

Sequence: $ACCGGUAGU$

2:8

Recurrence:

$$\mathrm{OPT}(i,j) = \max\left\{\mathrm{OPT}(i,j-1), \max_{\text{allowable } t}\{\mathrm{OPT}(i,t-1)+\mathrm{OPT}(t+1,j-1)\}\right\}$$

3:7 4:7

ACCGGU

CCGGUA

CGGUAG

GGUAGU

ACCGGUA 1:7

CCGGUAG 2:8

CGGUAGU 3:9

ACCGGUAG 1:8

CCGGUAGU 2:9

	6	7	8	j = 9
4	0	0	0	0
3	0	0	1	1
2	0	0	1	1
i = 1	1	1	1	2

RNA Folding Summary

- Compute the **optimal RNA folding** in time $O(n^3)$ and space $O(n^2)$
- **Dynamic Programming:**
  - Decide on an optimal pair $b_t - b_n$
  - Remaining RNA is two non-overlapping pieces
  - **Adding variables:** one subproblem for each interval
- **Non-crossing** is critical
  - Think about how the dynamic programming algorithm changes if we remove each of the conditions

Midterm I Review

Midterm I Topics

- Fundamentals:
  - Induction
  - Asymptotics
  - Recurrences
- Divide and Conquer
- Dynamic Programming

Last year's midterm will be online.

Cheat Sheets:

One 8x11 page

Double sided

Typed or handwritten

Use the HW template or 11pt font

Topics: Induction

- Proof by Induction:
  - Mathematical formulas, e.g. $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$
  - Spot the bug
  - Solutions to recurrences
  - Correctness of divide-and-conquer algorithms

- Good way to study:
  - Lehman-Leighton-Meyer, *Mathematics for CS* } Link to this on the website
  - Review divide-and-conquer in Kleinberg-Tardos

Practice Question: Induction

- Suppose you have an unlimited supply of 3 and 7 cent coins, prove by induction that you can make any amount $n \geq 12$.

Topics: Asymptotics

- Asymptotic Notation
  - $o, O, \omega, \Omega, \Theta$
  - Relationships between common function types

- Good way to study:
  - Kleinberg-Tardos Chapter 2

Jeff Erickson Book (Also linked online)

Topics: Asymptotics

Notation	… means …	Think…	E.g.
f(n)=O(n)	$\exists c > 0, n_0 > 0, \forall n \geq n_0$: $0 \leq f(n) \leq cg(n)$	At most "≤"	$100n^2 = O(n^3)$
f(n)=Ω(g(n))	$\exists c > 0, n_0 > 0, \forall n \geq n_0$: $0 \leq cg(n) \leq f(n)$	At least "≥"	$2^n = \Omega(n^{100})$
f(n)=Θ(g(n))	$f(n) = O\big(g(n)\big)$ and $f(n) = \Omega(g(n))$	Equals "="	$\log(n!) = \Theta(n \log n)$
f(n)=o(g(n))	$\forall c > 0, \exists n_0 > 0, \forall n \geq n_0$: $0 \leq f(n) < cg(n)$	Less than "<"	$n^2 = o(2^n)$
f(n)=ω(g(n))	$\forall c > 0, \exists n_0 > 0, \forall n \geq n_0$: $0 \leq cg(n) < f(n)$	Greater than ">"	$n^2 = \omega(\log n)$

Topics: Asymptotics

- **Constant factors can be ignored**
  - $\forall C > 0 \quad Cn = O(n)$
- **Smaller exponents are Big-Oh of larger exponents**
  - $\forall a > b \quad n^b = O(n^a)$
- **Any logarithm is Big-Oh of any polynomial**
  - $\forall a, \varepsilon > 0 \quad \log_2^a n = O(n^\varepsilon)$
- **Any polynomial is Big-Oh of any exponential**
  - $\forall\ a > 0, b > 1 \quad n^a = O(b^n)$
- **Lower order terms can be dropped**
  - $n^2 + n^{3/2} + n = O(n^2)$

Practice Question: Asymptotics

- Put these functions in order so that $f_i = O(f_{i+1})$
  - $n^{\log_2 7}$ $n^{2.82}$ $n^{2.804}$
  - $8^{\log_2 n}$
  - $2^{3 \log_2 \log_2 n}$ n^2 n^3
  - $2^{(\log_2 n)^2}$
  - $\sum_{i=1}^{n} i$
  - $n^2 \log_2 n$

Practice Question: Asymptotics

- Suppose $f_1 = O(g)$ and $f_2 = O(g)$.
  Prove that $f_1 + f_2 = O(g)$.

Topics: Recurrences

$$T(n) = T\left(\frac{7n}{10}\right) + T\left(\frac{2n}{10}\right) + n$$

- Recurrences
  - Representing running time by a recurrence
  - Solving common recurrences → Drawing the recursion tree
  - Master Theorem

- Good way to study:
  - Erickson book
  - Kleinberg-Tardos divide-and-conquer chapter

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

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

Practice Question: Recurrences

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

```
F(n):
  For i = 1,…,n²: Print "Hi"
  For i = 1,…,3: F(n/3)
```

- Write a recurrence for the running time of this algorithm. Write the asymptotic running time given by the recurrence.

Topics: Recurrences

$T(n) = \Theta(n \log\log n)$

- Consder the recurrence $T(n) = \sqrt{n} \cdot T(\sqrt{n}) + n$ with $T(1) = 1$. Solve using a recursion tree.

$T(2^n) = 2^{n/2}\ T(2^{n/2}) + \log(2^n)$

Topics: Divide-and-Conquer

- Divide-and-Conquer
  - Writing pseudocode
  - Proving correctness by induction
  - Analyzing running time via recurrences
- Examples we've studied:
  - Mergesort, Binary Search, Karatsuba's, Selection
- Good way to study:
  - Example problems from Kleinberg-Tardos or Erickson

    Good discussion of pseudocode
  - Practice, practice, practice!

Topics: Dynamic Programming

- Dynamic Programming
  - Identify sub-problems
  - Write a recurrence, $OPT(n) = \max\{v_n + OPT(n-6), OPT(n-1)\}$
  - Fill the dynamic programming table
  - Find the optimal solution
  - Analyze running time

- Good way to study:
  - Example problems from Kleinberg-Tardos or Erickson
  - Practice, practice, practice!

Practice Question

- Design an $O(n)$-time algorithm that takes an array $A[1{:}n]$ and returns a sorted array containing the smallest $\sqrt{n}$ elements of A

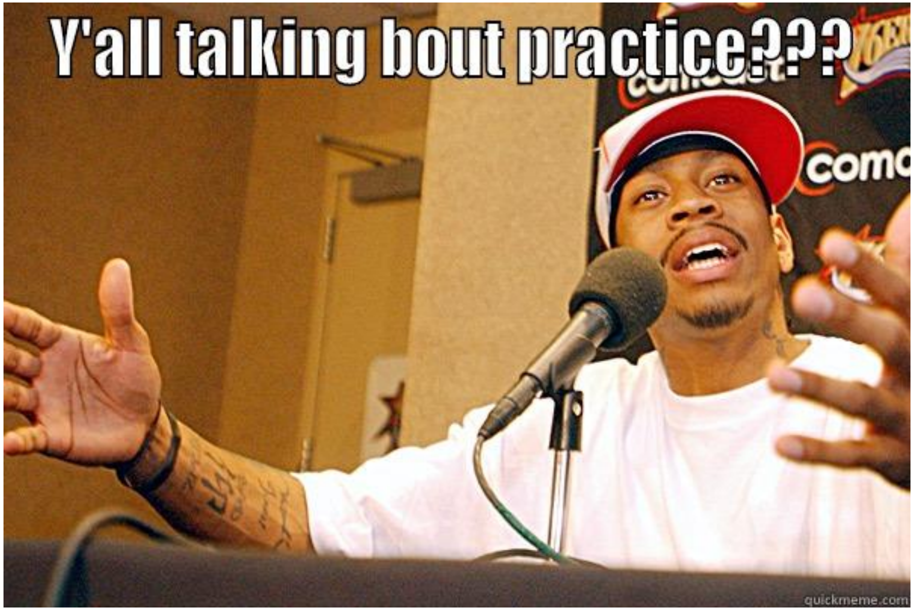
Practice Question

- Consider the following sorting algorithm

```
A[1:n] is a global array
SillySort(1,n):
  if (n <= 2): put A in order
  else:
    SillySort(1,2n/3)
    SillySort(n/3,n)
    SillySort(1,2n/3)
```

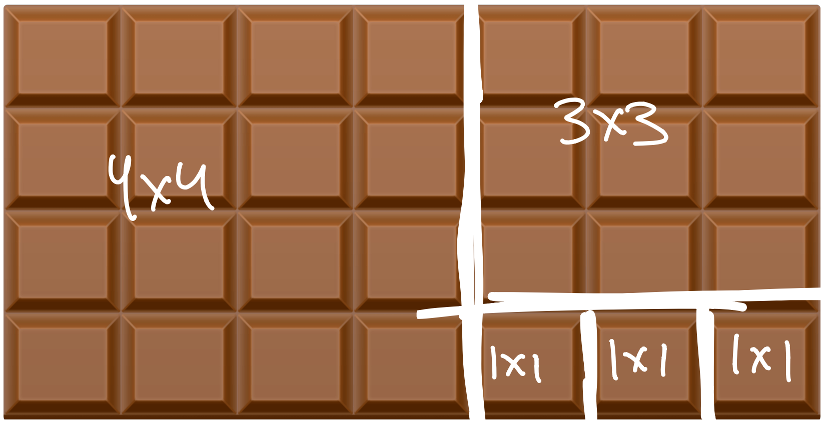
- Prove that it is correct
- Analyze its running time

Dynamic Programming Practice

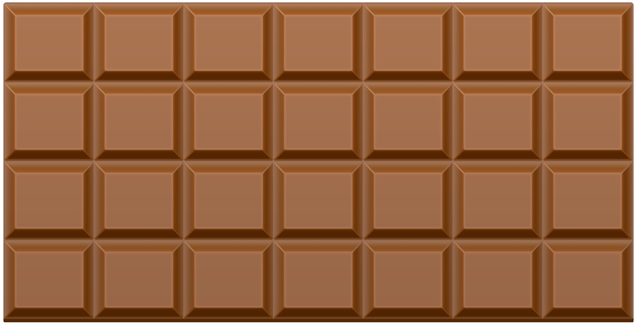


Chocolate Bar Splitting

- **Input:** A chocolate bar with $n \times m$ pieces
- **Output:** The minimum number of cuts needed to divide the block into perfect squares



Chocolate Bar Splitting



Vankin's Mile

-1	7	-8	10	-5
-4	-9	8	-6	0
5	-2	-6	-6	7
-7	4	7	-3	-3
7	1	-6	4	-9

score = 10

- **Input:** An $n \times n$ board of numbers
- **Rules:**
  - Place a chip on the board
  - Keep moving the tile **down** or **right** until you fall off
  - Score = sum of the numbers your chip visited
- **Output:** The best possible strategy