Topics: Dynamic Programming

- Dynamic Programming
  - Identify sub-problems
  - Write a recurrence, $(\text{e.g. } OPT(n) = \max\{v_n + OPT(n-6), OPT(n-1)\})$
  - 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: DP

Problem 2. *Dynamic Programming*

The dark lord Sauron loves to destroy the kingdoms of Middle Earth. But he just can't catch a break, and is always eventually defeated. After a defeat, he requires three epochs to rebuild his strength and once again rise to destroy the kingdoms of Middle Earth. In this problem, you will help Sauron decide in which epochs to rise and destroy the kingdoms of Middle Earth.

The input to the algorithm consists of the numbers $x_1, \ldots, x_n$ representing the number of kingdoms in each epoch. If Sauron rises in epoch i then he will destroy all x_i kingdoms, but will not be able to rise again during epochs $i+1, i+2$, or $i+3$. We call a set $S \subseteq \{1, \ldots, n\}$ of epochs *valid* if it satisfies this constraint that $|i-j| \geq 4$ for all $i, j \in S$, and its *value* is $\sum_{i \in S} x_i$. You will design an algorithm that outputs a valid set of epochs with the maximum possible value.

Example: Suppose there are $(1, 7, 8, 2, 6, 3)$ kingdoms of Middle Earth in epochs $1, \ldots, 6$. Then the optimal set of epochs for Sauron to rise up and destroy the kingdoms of Middle Earth is $S = \{2, 6\}$, during which he destroys 10 kingdoms, 7 in the 2nd epoch and 3 in the 6th epoch.

Using DP…

*** describe the set of subproblems you consider**

*** give a recurrence expressing the solution to each subproblem in terms of the solution to smaller subproblems**

***sketch pseudocode of your algorithm & give the runtime**

***describe how you would recover the solution (epochs) if asked**

DP Practice

Longest Increasing Subsequence (LIS): Given a sequence of numbers, find the length of the longest subsequence such that the elements are in increasing order.

Ex. Input: 10, 22, 9, 33, 21, 50, 41, 60, 80, 47

Sol. LIS has length 6 (10, 22, 33, 50, 60, 80)

Let OPT(j) be the length of the LIS ending at the j-th number *(0 <= j <= n)*

Using DP…

* give a recurrence expressing the solution to each subproblem in terms of the solution to smaller subproblems

*sketch pseudocode of your algorithm & give the runtime

*describe how you would recover the LIS if asked

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*
  - Review divide-and-conquer in Kleinberg-Tardos
  - Review sheet on Piazza
  - Review video on Blackboard

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

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 ">"	$n^2 = \omega(\log n)$

Asymptotic Analysis Rules

- **Constant factors can be ignored**
  - $\forall C > 0 \quad Cn = O(n)$
- **Lower order terms can be dropped**
  - E.g. $n^2 + n^{3/2} + n = O(n^2)$
- **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**
  - $\forall\, a > 0, b > 1 \quad n^a = O(b^n)$

Practice Question: Asymptotics

- Put these functions in order so that $f_i = O(f_{i+1})$
  - $n^{\log_2 7}$
  - $8^{\log_2 n}$
  - $2^{3.1 \log_2 n}$
  - $2^{(\log_2 n)^2}$
  - $n^2 \sum_{i=1}^{n} i$
  - $n^2 \log_2 n$

(See HW1 for more examples)

Practice Question: Asymptotics

- Suppose $f_1(n) = O(g(n))$ and $f_2 = O(g(n))$. Prove that $f_1(n) + 4f_2(n) = O(g(n))$.

Topics: Recurrences

- Recurrences
  - Representing running time by a recurrence
  - Solving common recurrences
  - Master Theorem
  - Recursion Trees

- Good way to study:
  - Erickson book
  - Kleinberg-Tardos chapter 5

Practice Question: Recurrences

```
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

- Consder the recurrence $T(n) = \sqrt{n} \cdot T\left(\sqrt{n}\right) + n$ with $T(2) = 1$. Solve using a recursion tree.

Topics: Stable Matching

- Stable Matching
  - Definition of a stable matching
  - The Gale-Shapley algorithm
  - Consequences of the Gale-Shapley algorithm

- Good way to study:
  - Kleinberg-Tardos 1.1

Practice Question: Stable Matching

- Give an example of 3 doctors and 3 hospitals such that there exists a stable matching in which every hospital gets it last choice of doctor

Topics: Divide-and-Conquer

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

Practice Question: Divide-and-Conquer

You are babysitting your niece and before she will go to bed she insists on playing the following guessing game:

1. She picks a number x in $1, 2, \ldots, n$.
2. You make a guess y_1, and she simply says *correct* or *incorrect*.
3. You make a sequence of guesses $y_2, y_3, \ldots$. If your guess $y_i = x$ then your neice says *correct* and goes to bed. If your guess y_i is closer to x than the previous guess y_{i-1}, then she says *warmer* and if y_{i+1} is farther than the previous guess, then she says *colder*.

Your goal is to find x with as few guesses as possible so that your niece will go to bed. Design a divide-and-conquer algorithm that guesses your niece's number using $O(\log n)$ guesses.[1]

- Two versions:

  1. If your guess y_i is the same distance as guess y_{i-1}, your niece stays up forever and you lose

  2. If your guess y_i is the same distance as guess y_{i-1}, your niece says "same" and you win

Practice Question: Divide-and-Conquer

Problem 3. *Divide-and-Conquer*

Suppose you have two sorted arrays $A[1,\ldots,n]$ and $B[1,\ldots,n]$ of equal length, containing $2n$ numbers in total. Design an $O(\log n)$ time divide-and-conquer algorithm that finds the *median* of these $2n$ numbers in A and B. We use the convention that the median of an *odd*-length sorted list $C[1,\ldots,2m+1]$ is $C[m+1]$ and the median of an *even*-length sorted list $C[1,\ldots,2m]$ is $C[m]$.

Example: Suppose $n = 5$, $A = [1,4,7,9,19]$, and $B = [2,5,12,18,20]$, then the combined sorted list is $C = [1,2,4,5,7,9,12,18,19,20]$, whose median is $C[5] = 7$.

- Can assume unique values (no value appears more than once in *C)*

Practice Question: Divide-and-Conquer

- Describe your algorithm in pseudocode

Practice Question: Divide-and-Conquer

- Prove by induction that the algorithm is correct

Practice Question: Divide-and-Conquer

- Analyze your algorithm's running time by writing a recurrence

Practice Question: Alg. Design

- 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: Recurrence Analysis

- Consider the following sorting algorithm

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

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

Practice Question

(challenge ?): Prove the correctness of SillySort

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