Derrick DeRose

3.07)

M_1 M_2 M_3 M_4

10 × 2 2 × 10 10 × 2 2 × 10

cost(M_{12})=200 cost(M_{34})=200

10 × 10 10 × 10

(M_{1234}) total cost = 10000

worst total cost = 10400

M_2 M_1 M_4 M_3

2 × 10 10 × 2 2 × 10 10 × 2

40 40

2 × 2 2 × 2

8

best total cost = 88

88 × 100 = 8800, so the best total cost is 100 × less$ than the worst total cost

Derrick DeBase

3.02) Greedy Algorithm

1)
```
1 while (change count != 0){
2   fit as many of the largest coins not used yet
    in to the change amount.
3   Change count = Change count -( coins that
                                   fit * coin
                                   amount)
  }
```

$1.43
- 1.00 2 half dollars
.43
- .25 1 Quarter
.18
- .10 1 dime 8 total coins
.08
- .05 1 nickel
.03 3 pennies
- .03

2) max change of a dollar occurs @ $.99

Half Dollar ≤ 1
Quarter ≤ 1
Nickels + Dime ≤ 2
Nickels ≤ 1
Pennies ≤ 4

99¢ occurs with 3 Qs, 2 Ds, 4 pennies

There will always be a max of 4 pennies in any solution because 5 pennies will produce 1 nickel. we will never have 2 nickels because then we'd just exchange it for a dime. Nickel + Dime ≤ 2 because 3 nickel, 3 dimes, 2 Nickels + 1 Dime, or 1 Nickel + 2 Dimes can all be expressed in 2 coins. We will never have more than 1 quarter because then we could then use a half dollar and never more than 1 half dollar because that puts us over the change amount

3.02)

3) coin denominations

Derrick Li 2021

1, 20, 80, 100

change 160¢

Greedy - 100, 20, 20, 20

Optimal - 80, 80

4) Let a be the # of coin denominations.
Let K[a] be the smallest coin denomination (1)
for (i=1; i!=a; ++i)
min(GreedyCoinAlgorithm(coin amount, max coin denom. K[i]))

This algorithm uses a modified Greedy Coin algorithm to allow for smaller coin denominations to be the max coin used. This ensures that with any set of coin denominations we can find the minimum number of coins needed to make a change amount

worst-case running time = O(n a)
where n is the number of coins needed to make the change and we have to check a different denominations.

Worst-case space requirements = O(a)
we would a most have to store a different coin amounts from a different denominations.

Derick DeBos2

3.04) $T_2(n) = 2T_2(n-2)$ for $n \geq 2$

Assume $T_2(n) = \Theta(2^{n/2})$

$T_2(n-2) = \quad 2\ T_2(n-4)$

$T_2(n) = 2 * 2T_2(n-4)$

$T_2(n-4) = 2\ T_2(n-6)$

$T_2(n) = 2*2*2\ T_2(n-6)$

$T_2(n) = 2^k T_2(n-2k)$ where $1 \leq k \leq n/2$.

$\boxed{T_2(n) = \Theta(2^{n/2})}$ k is the level of our recursion tree.

$n-2$ subproblem $= n/2$ tree levels.

$c_1 \cdot 2^{n/2} \leq 2^n \leq c_2 \cdot 2^{n/2}$ ✗

$\Theta(2^{n/2}) \neq \Theta(2^n)$ because there is no such constant c_1 or c_2 such that the inequality above becomes true.

3.05)

Derivation DeBase

$$T(n) = T(n-1) + T(n-2) + \dots + T(n-k) \quad \text{for } n \geq k$$

$$T(n) \geq T(n-1) + T(n-2)$$

assuming $T(n-2) < T(n-1)$

$$T(n) \geq T(n-2) + T(n-2)$$

$$\underline{T(n) \geq 2T(n-2)}$$

Same reccurrence relation from 3.04 but with a n inequality to test for Ω

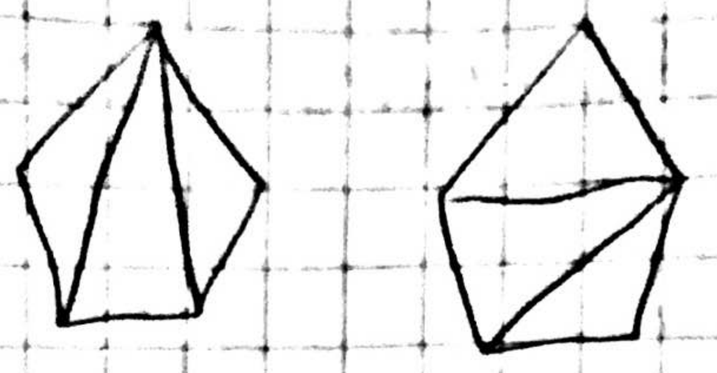
$$T(n) = \Omega(2^{n/2})$$

$$2^{n/2} \neq \Omega(2^n)$$

The lower bounds are not equal because we can't say our $2^{n/2}$ computations is at least 2^n computations.

3.07)

Derrick DeBoer

1) 

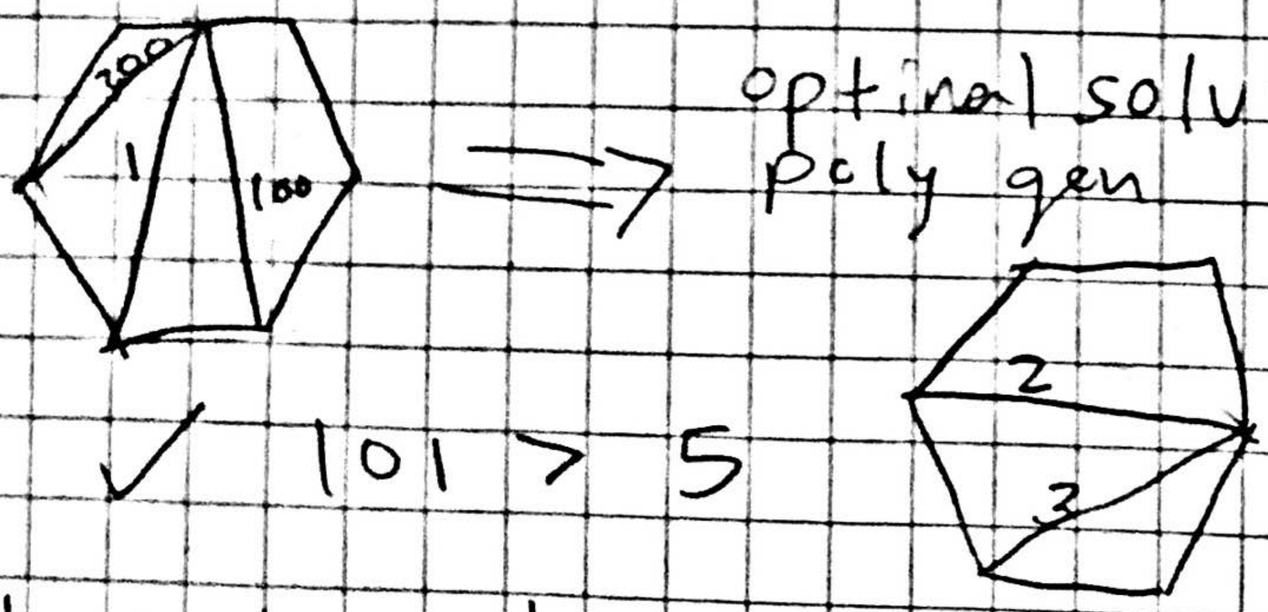
Assume we have 2 non-intersecting chords that share the same point.

If there is only one side between our 2 chords then that last side must be an edge of our polygon.

A non-intersecting chord can be added to split up a polygon into 2 polygons.

If non-intersecting chords continue to added until no more can be added, then we will be left with a polygon consisting of only triangles.

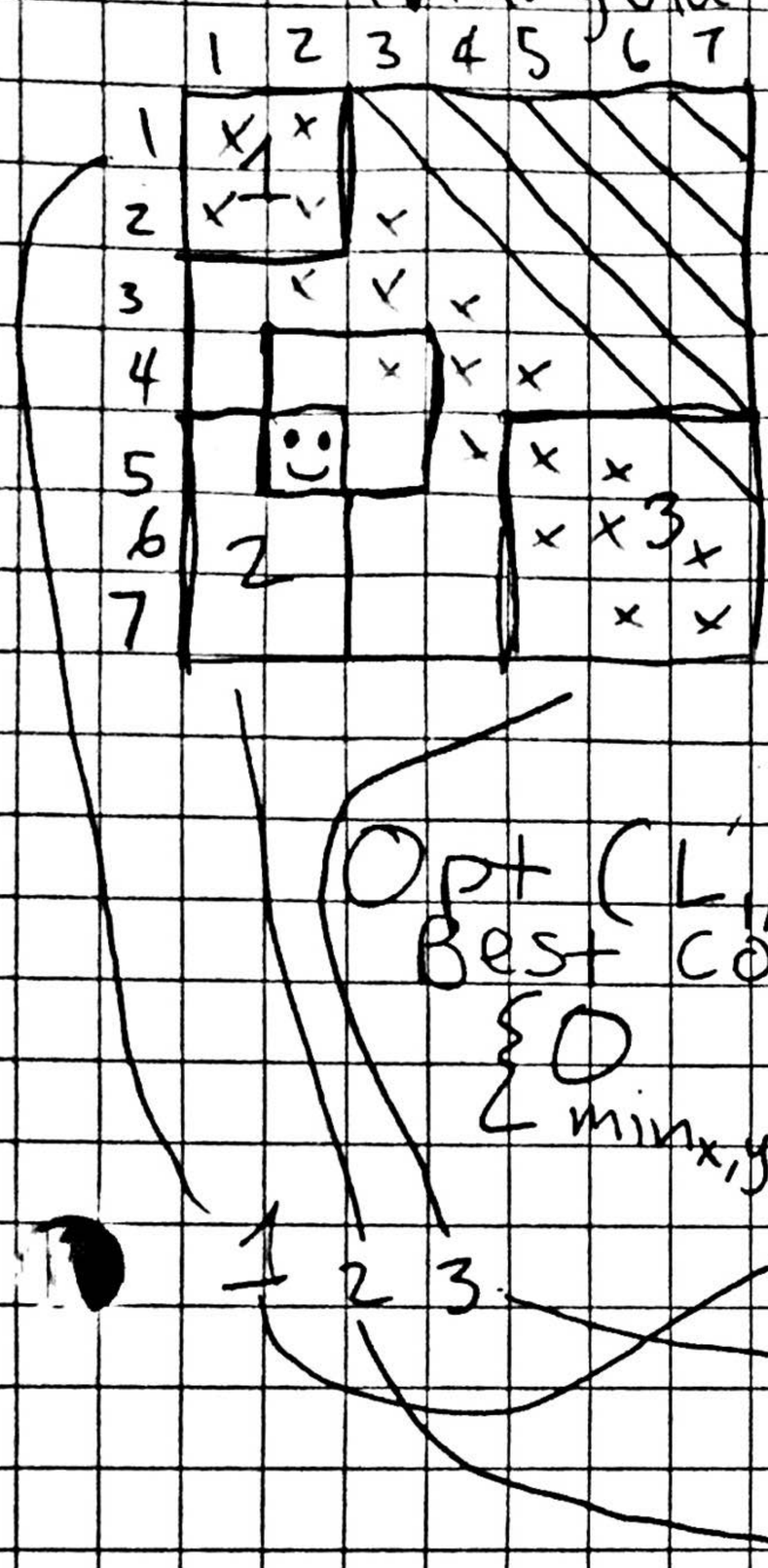
2) 1) choose the shortest chord first.
2) then choose the shortest of the adjacent chords



Let n be all possible chords of a polygon. to find min of a set is O(n). with an O(1) comparison of the 2 adjacent chords. worst case running time is O(n).

Derick DeBoce

3) In order to make an Opt function we must first assume we have an edge that will be apart of our triangulation. $A_{x,y}$



x not a chord duplicate answer

☺ $A_{x,y}$

This algorithm eliminates intersecting chords based on our given chord that is apart of the minimum triangulization

$Opt(L_1, L_2, H_1, H_2) =$

Best cost =

$$\begin{cases} 0 \\ \min_{x,y} \{ A_{x,y} + Opt(x, y, x, y) \end{cases}$$

found in center box with ☺ in the bottom left.

$$+ \min \begin{cases} Opt(L_1, x, H_1, x) \\ Opt(y, L_2, y, H_2) \\ Opt(L_1, x, y, H_2) \end{cases}$$

min function used to find min non intersecting chord.

L_1 - top left point of 2D array
L_2 - bottom left point
H_1 - top right point
H_2 - bottom right point.
4 points create a box on the 2D array of possible non intersecting chords.

Worst case runtime - (2^n) because each n-element is broken up into 2 subproblems of 2 polygons.

Low & high side of the polygon.

Derrick DeBose

4) Opt(L_1, L_2, H_1, H_2) {
Best cost = 0
cost = min{ $A_{x,y}$, Opt(x, y, x, y)

+ min { Opt(L_1, x, H_1, x)
Opt(y, L_2, y, H_2)
Opt(L_1, x, y, H_2) }

bCost = retrieve(bestCost)
if bCost > cost
bestCost = cost
Store (bestCost)

worst case run-time — $O(n^5)$.

DP is polynomial time because we have 2 subproblems and 3 cases $O(n^2)$ subproblems with $O(n^3)$ cases.
So the worst time is $O(n^5)$

Help on this problem was given at LSS tutoring.