15.3

3) modifying the matrix-Chain-multiplication problem still has optimal substructure. if we split $A_1 \dots A_n$ into $A_1 \dots A_k$ and $A_{k+1} \dots A_n$ there has to be a max cost on each side otherwise we could substitute a more expensive multiplication

where in the original problem we have min (count, minval) we can just replace min with max and it will do the same thing except it will find the max

RECURSIVE-MATRIX-CHAIN(p, i, j)

```
1  if i == j
2      return 0
3  m[i, j] = ∞
4  for k = i to j − 1
5      q = RECURSIVE-MATRIX-CHAIN(p, i, k)
           + RECURSIVE-MATRIX-CHAIN(p, k + 1, j)
           + p_{i−1} p_k p_j
6      if q < m[i, j]
7          m[i, j] = q
8  return m[i, j]
```

Changes

$m[i,j] = -\infty$

Set max val

Check for min

Changes

if $q > m[i,j]$

$m[i,j] = q$

15.4

5) Sequence X longest MIS

Sort X $O(n \log(n))$ and save it to y

Call LCS length on X & y since y contains the longest possible and is guaranteed to increase only increasing vals of X are considered

LCS length has runtime $O(n^2)$

$n \log n + n^2 = n^2$

$O(n^2)$

16.1

3)

Counter example Shortest time

1-3, 2-4, 3-7 Shortest is 2-4 but that conflicts with 1-3 and 3-7 so this doesnt work

Counter example of fewest overlaps

3-5, 0-2, 6-8, 2-4, 4-6, 1-3, 1-3, 5-7, 5-7

Optimal 0-2, 2-4, 4-6, 6-8

Overlap 0-2, 3-5, 6-8

Counter example of earliest start times

{1-10, 2-3, 4-5}

Earliest start time is 1-10 but this isnt optimal 2-3, 4-5 is.

16.2

4) if we choose the furthest stop within m miles we update our position and repeat the choice this has optimal substructure which is required for greedy choice

let O be an optimal solution where the plof stops at positions $o_1 \dots o_k$ let g_1 denote the furthest we can go without stopping then we may replace o_1 by g_2 to make a solution G since $o_2 - o_1 < o_2 - g_1$. since G has the same number of stops we can say that g_1 is contained in the optimal solution

Since we need to look at each stop and decide to take it its runtime is O(n), if we need to sort the stops O(nlogn) is the runtime because sorting takes the longest

5)

We need to contain both ends of the line

So we start from the left so X_1 starts and we remove any point within unit distance of the chosen point we then move to the next point on the line and repeat our choice from there since this is the same choice each time it exhibits optimal substructure and thus greedy choice works

runtime $O(n)$ the list is already sorted

7) Sort both lists smallest to largest

the way to get the biggest number is the biggest number raised to the biggest power

runtime is $O\log(n) * 2$ for both lists adding payouts adds n, in total $O(n\log(n))$

Suppose optimal payout S in which a_q is paired with b_1 and a_1 is paired with b_p

$$\frac{\text{Payoff}(S)}{\text{Payoff}(S')} \frac{(a_q)^{b_1}(a_1)^{b_p}}{(a)^{b_p}(a_1)^{b_1}} = \left(\frac{a_q}{a_1}\right)^{b_1 - b_p}$$

Since $a_q > a_1$ & $b_p > b_1$ because we sorted least to greatest then payoff $S/S' < 1$

this contradicts optimal therefore optimal would pair a_q with b_p