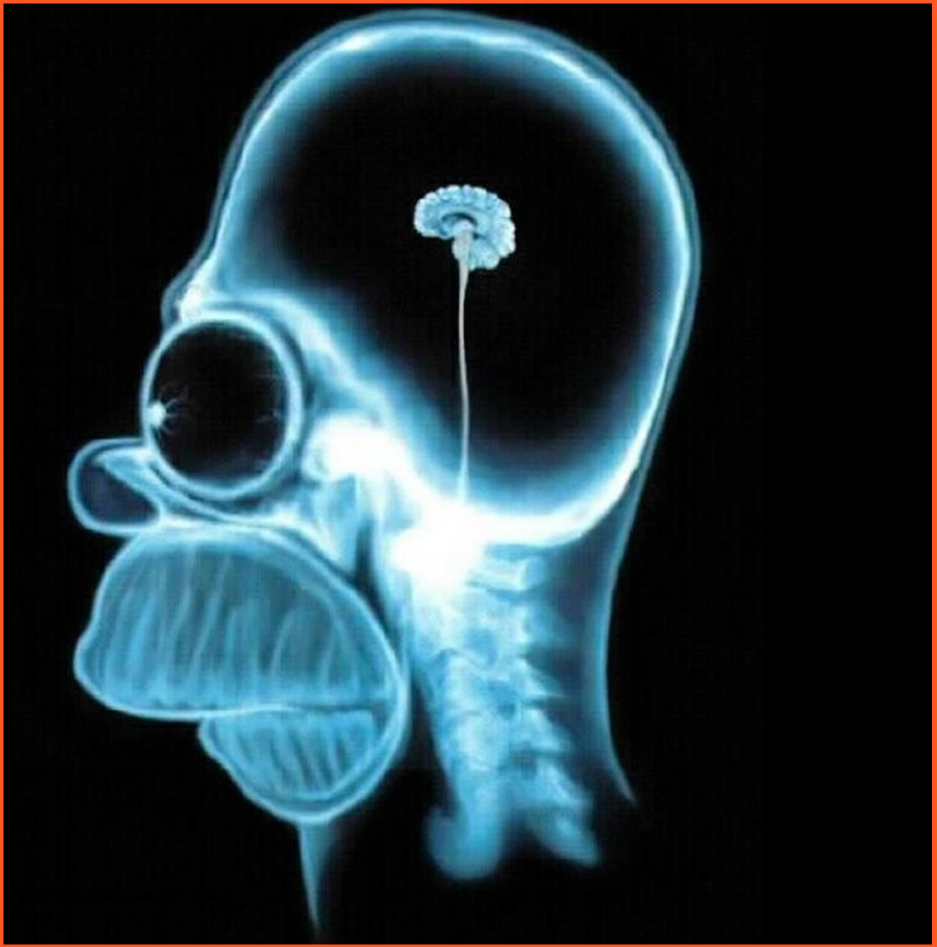
Dynamic Programming Examples

Selected **dynamic programming** problem solutions from *Programming Challanges*

Is Bigger Smarter?



Is Bigger Smarter?

- **Goal:** For two sequences **W** and **S** of integers find the longest sequence $(i_1, i_2, \ldots, i_K)$ such that $W[i_1] < W[i_2] < \ldots < W[i_k]$ and $S[i_1] > S[i_2] > \ldots > S[i_K]$.
- **Idea:** Sort the elements of **W** and **S** according to, say, **W** (remember original position of the element for the output). For this new ordering, we define **L[i]** as **the length of the wanted sequence ending at i-th elephant**. Initially, **L[i]** is 1 for all i. Then we iteratively update **L[j]** as follows: **L[j] = max{L[i] + 1 for i = 0 to j - 1 if S[i] > S[j] and W[i] != W[j]}**
- **Complexity:** $O(N^2)$

Is Bigger Smarter?

Input Sequence:

W:	6008	6000	500	1000	1100	6000	8000	6000	2000
S:	1300	2100	2000	4000	3000	2000	1400	1200	1900

Ordered by W:

O:	3	4	5	9	8	6	2	1	7
W:	500	1000	1100	2000	6000	6000	6000	6008	8000
S:	2000	4000	3000	1900	1200	2000	2100	1300	1400

Updating L:

L[1]:	1	1	1	1	1	1	1	1	1
L[2]:	1	1	1	1	1	1	1	1	1
L[3]:	1	**1**	**2**	1	1	1	1	1	1
L[4]:	1	1	**2**	**3**	1	1	1	1	1
L[5]:	1	1	2	**3**	**4**	1	1	1	1
L[6]:	1	1	**2**	3	4	**3**	1	1	1
L[7]:	1	1	**2**	3	4	3	**3**	1	1
L[8]:	1	1	2	3	4	3	**3**	**4**	1
L[9]:	1	1	2	**3**	4	3	3	4	**4**

Weights and Measures

Weights and Measures

- **Goal:** Given a list of turtles described by their weights and strengths, you should determine the highest tower you can build out of them, such that each turtle in the tower has enough strenght to carry the turtles on its back (including itself).
- **Idea:** Sort the turtles by their strength. Keep the height of the heighest tower so far and for each tower store its weight, starting with 0/0 tower (height/weight). Keep trying to enlarge the existing towers using the turtles (in order) until trying all of them.
- **Complexity:** O(Nh), where h is the height of the tallest tower -- potentially N.

Weights and Measures

Input Sequence (weights and strenght):

```
W:  300 1000 200 100
S: 1000 1200 600 101
```

Ordered by S:

```
W: 100 200  300 1000
S: 101 600 1000 1200
```

Updating towers:

```
T[0]:  0  inf   inf   inf   inf   (initial “towers”)
T[1]:  0  100   inf   inf   inf
T[2]:  0  100   300   inf   inf
T[3]:  0  100   300   600   inf
T[4]:  0  100   300   600   inf
```

Weights and Measures

- **Correctness of this algorithm is in question!?**
  - The order in which the turtles are tried does make a difference! For example, random ordering or ordering using the remaining strength (S[i] - W[i]) does not work, consider turtles (17, 20), (3, 10).
  - Befare! uDebug shows an incorrect answer in this case (solution by: forthright48)!
  - Good idea for a presentation?

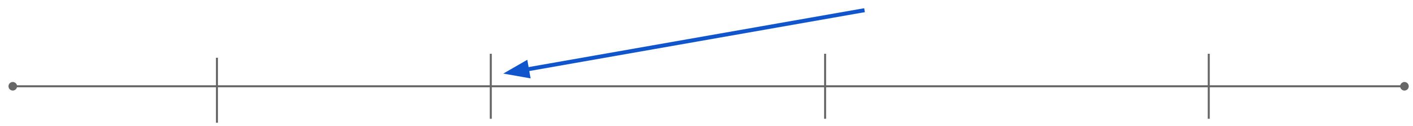
Cutting Sticks

Cutting Sticks

- **Goal:** Given a stick of length L and places where to cut it (in distance from the left end), determine the order in which to make the cuts such that it is cheapest possible. The price of the cut is equal to the length of the stick being cut.
- **Idea:** Lets define C[i][j] as the cheapest price of cutting the stick from the i-th to j-th cut position. Define L[0] = 0 and L[N + 1] = L, where N is the number of cuts. C[i][j] can be (recursively) determine using the following formula: **C[i][j] = min{C[i][k] + C[k][j], i < k < j + L[j] - L[i]} for i < j + 1**
C[i][j] = 0 for i = j + 1 (the “i-j stick” is just a single piece)

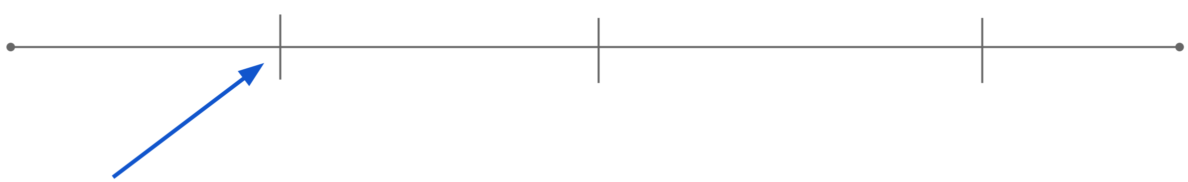
Cutting Sticks

In order to know the optimal price of the cut here...



… you need to know the price of the following cuting...

… the trick is in avoiding recomputing the prices of possible cuts...



… look that the answer to the red stick is used in finding the price of a cut of this smaller stick. And not recomputing these really makes a difference!