

DYNAMIC PROGRAMMING

Recall Q1 of Assignment-4 about the factory that produces solar panels for large businesses. Would the following greedy algorithm work for that problem? Assume $R_H=0$ and $R_L=0$ in week $n+1$.

```
for (i=1; i<=n; ) {
    if ( value of  $R_H$  in week i+1 > (value of  $R_L$  in week i +
                                     value of  $R_L$  in week i+1) )
        { Pick priming in week i and  $R_H$  in week i+1
          i = i+2
        }
    else
        { Pick  $R_L$  in week i
          i = i+1
        }
}
```

If YES, then prove it. If NO, then give a counterexample. **NO**

Counter Example:

	Week 1	Week 2	Week 3	Week 4	Week 5
RH	1	5	20	4	0
RL	2	2	1	2	0

Greedy algorithm will pick RH=5 in Week-2, nothing in Week-1 and goes to Week-3. From there it picks RH=4 in Week-4, nothing in Week-3 and goes to Week-5. Total Revenue = 5+4=9

Optimal Algorithm will pick the following: Week-1 RL=2, Week-2 Nothing, Week-3 RH=20, Week-4 RL=2. Total revenue = 2+20+2 = 24

You are given a list of 'n' ordered items that are numbered $\text{item}_1, \text{item}_2, \text{item}_3, \dots, \text{item}_n$. You cannot change the order of the items. There are two things associated with each item: points and penalties, i.e. item_i has points_i and penalty_i . **You have to select a set of items that maximizes the total points.** The rules are as follows: If you select item_i , you will get points_i , however, you won't be allowed to select the following penalty_i items in the list.

Example:

Items	item_1	item_2	item_3	item_4	item_5	item_6
Points	6	8	17	2	22	11
Penalty	2	1	3	2	1	0

In the above example, if you select item_1 , you won't be able to select the **two** items following item_1 (i.e. item_2 and item_3) because $\text{penalty}_1=2$.

Give an **efficient** solution to this problem using **dynamic programming**. Write down the **recurrence relation** for this problem. (**Note:** Clearly explain what each of the terms in your recurrence mean.)

Base case: $\text{fopt}(n) = \text{points}_n$
 $\text{fopt}(n+1) = 0$

Recurrence:

$\text{fopt}(i) = \max (\text{fopt}(i+\text{penalty}_i + 1) + \text{points}_i , \text{fopt}(i+1))$

Show the values your recurrence relation would compute on the above example (**You have to show each step and set of items selected selected by your solution**).

each step and set of items selected selected by your solution.

$$\begin{aligned}
 \text{fopt}(1) &= \max [\text{fopt}(2), \text{fopt}(1+2+1)+6] = \max (30, 26+6) = 30 \\
 \text{fopt}(2) &= \max [\text{fopt}(3), \text{fopt}(2+1+1)+8] = \max (22, 22+8) = 30 \\
 \text{fopt}(3) &= \max [\text{fopt}(4), \text{fopt}(3+1+3)+17] = \max (22, 17) = 22 \\
 \text{fopt}(4) &= \max [\text{fopt}(5), \text{fopt}(4+1+2)+2] = \max (\text{fopt}(4+1+2)+2, 22) = 22 \\
 \text{fopt}(5) &= \max [\text{fopt}(6), \text{fopt}(5+1+1)+22] = \max (11, 0+22) = 22 \\
 \text{fopt}(6) &= 11 \\
 \text{fopt}(7) &= 0
 \end{aligned}$$

Time Complexity **O(n)**

You are given a box containing 'n' spherical balls of different radii. Ball i has a radius r_i and a surface area a_i equal to $4\pi r_i^2$. From the 'n' balls, you have to select a subset of balls such that sum of their areas is **equal** to X . If no such solution exists, then report that it is not possible.

Write down the recurrence relation for this problem. (**Note:** Clearly explain what each of the terms in your recurrence mean.)

Program will call with $\text{FOPT}(n, X)$

Base case: This is similar to Knapsack problem, where you will select a subset of balls to maximize the surface area. If the maximum found by the algorithm is equal to X then we will report success, otherwise "not possible".

Recurrence:

If $x < a_i$ then $\text{FOPT}(i, x) = \text{FOPT}(i-1, x)$

else $\text{FOPT}(i, x) = \max(\text{FOPT}(i-1, x), a_i + \text{FOPT}(i-1, x - a_i))$

Suppose you are given the following 4 balls with the corresponding areas. Value of X is 6. Show the values your recurrence relation would compute on the following example (**You have to clearly show how many boxes there are and how you fill those boxes**).

B_1	B_2	B_3	B_4
1	3	4	5

For example:

In the cell marked with *

$\text{FOPT}(3, 5) = \max(\text{FOPT}(2, 5), 4 + \text{FOPT}(2, 5-4))$

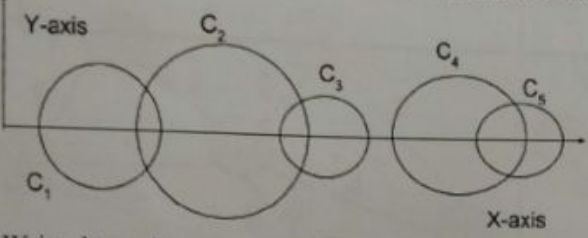
$\text{FOPT}(3, 5) = \max(4, 5)$

$\text{FOPT}(3, 5) = 5$

Output of algorithm: $\text{FOPT}(4, 6) = 6$ which is equal to X , so reports "success"

Time Complexity : $O(nX)$ psuedo-polynomial

You are given a drawing of 'n' circles in the cartesian coordinate system. The centers of each of the circles lie on the x-axis as shown below. Circle i is centered at coordinates $(x_i, 0)$, has radius r_i and circumference c_i equal to $2\pi r_i$. From the 'n' circles, you have to select a subset of circles such that they don't intersect and the sum of their circumferences is maximized. If a circle is completely enclosed within another, that also counts as intersecting circles.



You may approach this problem in the same way as weighted interval scheduling. Define $P(j)$ to be the largest index $i < j$ such that circles i & j do not intersect.

Write down the recurrence relation for this problem. (Note: Clearly explain what each of the terms in your recurrence mean.) [4 marks]

Base case: $f_{OPT}(0) = 0$

Recurrence: $f_{OPT}(i) = \max \left(c_i + f_{OPT}(P(i)), f_{OPT}(i-1) \right)$

In the drawing of five circles above, suppose the centers and radii are as follows. Show the values your recurrence relation would compute on the following example (You have to clearly show how many boxes there are and how you fill those boxes).

C_1	C_2	C_3	C_4	C_5
Center=(3,0)	Center=(10,0)	Center=(17,0)	Center=(22,0)	Center=(24,0)
Radius = 2	Radius = 6	Radius = 1.5	Radius = 2	Radius = 1

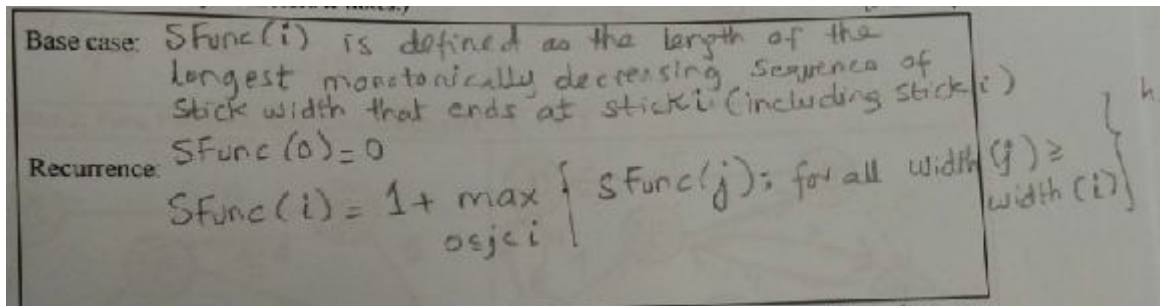
Circumference c_i	4π	12π	3π	4π	2π
$f_{OPT}(1) = \max(4\pi + f_{OPT}(0), f_{OPT}(0)) = 4\pi$ $f_{OPT}(2) = \max(12\pi + f_{OPT}(0), f_{OPT}(1)) = 12\pi$ $f_{OPT}(3) = \max(3\pi + f_{OPT}(1), f_{OPT}(2)) = 12\pi$ $f_{OPT}(4) = \max(4\pi + f_{OPT}(3), f_{OPT}(3)) = 16\pi$ $f_{OPT}(5) = \max(2\pi + f_{OPT}(3), f_{OPT}(4)) = 16\pi$ Circles Selected = C_2 and C_4					

Time Complexity : $O(n)$ Assuming $p(j)$ values are pre-calculated and known.

Consider 'n' sticks that are part of an art piece (see illustration below) fixed to the floor. These sticks have different widths and are arranged in a line. The artist wants to modify the artwork by removing some sticks such that the remaining 'x' sticks are sorted, widthwise, descendingly from left to right. The value of 'x' should be as large as possible. The locations of the remaining sticks cannot be changed.

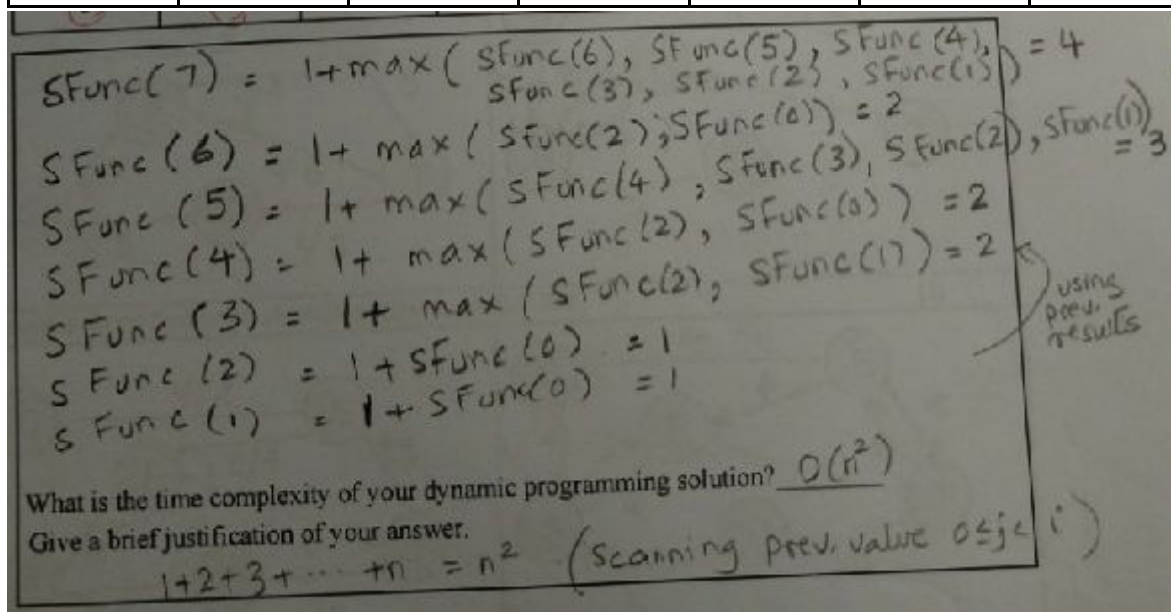


Write down the recurrence relation for this problem. (Note: Clearly explain what your recurrence defines and what parameters it takes.)



You are given the following sticks with the corresponding widths. **Stepwise**, show **all the values** your recurrence relation would compute on the following example (**You have to clearly show how many subproblems there are and how long it takes to compute each subproblem**).

S_1	S_2	S_3	S_4	S_5	S_6	S_7
6	12	5	7	4	8	3



There is a new board game called 'Pointers' which is played by two players. In 'Pointers' each player selects from 'p' different moves during his/her turn. You have collected data of the moves made by two expert players during a game. You want to find the maximum length of the subsequence of moves that are identical in their plays. The ordering of the moves must be preserved when calculating the maximum. Solve this problem using dynamic programming. Assume each of the 'p' moves is represented by a distinct number and 'n' moves are made by each player.

Write the recurrence relation of your dynamic programming algorithm. For full credit, you have to clearly explain what the terms in your recurrence stand for.

Similar to Longest Common Subsequence (LCS). Think about the recurrence relation - try to write it without looking at your class notes.

What is the running time of your algorithm. Explain.

$O(n^2)$

Suppose you are given an implementation of the Edit Distance (ED) algorithm and you want to use it to solve the Longest Common Subsequence problem. Describe how you would solve this problem.

Set the cost of 'Substitution' in Edit Distance to infinity. The cost of 'Match' is zero and cost of 'Insert' is 1 and cost of 'Delete' is 1.

Run ED, it will output the minimum number of changes required to transform one string to another. From that path, determine the number of 'Matches'. This will give you the LCS value.

Your gardener is planting seeds for different flowering plants in **rows** in a rectangular dirt patch. You have decided which seeds should be planted next to each other and you have a seed dispenser which discharges seeds in your specified **order**. The total number of seeds is 'n'. The gardener has to decide how many seeds to plant in each row. If he plants too many, the flowers will not have space to grow properly and if he plants too few he will waste space in the patch. Different plants have different space requirements. Suppose you are given a machine 'M' that displays an integer value 'I' when you place some number of the **ordered** seeds (say x) on it. You have to design a dynamic programming algorithm that tells you how many seeds to plant in each row such that the sum of 'I' values over all the rows is maximized.

Assume machine 'M' gives you the integer value 'I' in constant time.

Write the recurrence relation of your dynamic programming algorithm. For full credit, you have to clearly explain what the terms in your recurrence stand for.

Before planting seeds, the gardener will place 'x' number of seeds on the machine and machine will give the value of 'I'. The actual planting in the ground would be done once the gardener knows the values for all the rows. Note that the gardner cannot change the order of the seeds that are discharged by the seed dispenser.

This is similar to the Segmented Least Squares (line fitting) problem. Think about the recurrence relation - try to write it without looking at your class notes.

What is the running time of your algorithm. Explain.

$O(n^2)$ since we are assuming the machine gives the answer in $O(1)$ time.