Example Title

Peder Brandstorp Sanden

Section

Some math

$$T(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ T(n-1) + T(n-2) + 1 & \text{if } n \geq 2 \end{cases} \tag{1}$$

$$\begin{aligned} T(n) &= T(n-1) + T(n-2) + 1 \\ T(n) &= T(n-1) + T(n-2) \\ T(n) &= 2 \cdot T(n-1) \\ &\Downarrow \\ T(n) &= 2 \cdot 2 \cdot T(n-2) \\ T(n) &= 2^{k-1} \cdot T(n-k) \quad \{k = n-1\} \\ T(n) &= 2^{k-1} \cdot T(1) \\ T(n) &= O(2^n) \end{aligned} \tag{2}$$

A List

The two properties a problem must have to be solved with dynamic programming are:

- Optimal substructure
- Overlapping sub-problems

A Table

Done after two iterations as there are no changes between iteration *1.5* and *2*, see table 1. The shortest path from s to t is 6 and goes: s-A-B-D-t.

Iteration	**s**	**A**	**B**	**C**	**D**	**t**
0	0	∞	∞	∞	∞	∞
1	0	1	3	3	4	6
2	0	1	3	3	4	6

Table 1: Iteratioins for finding the shortest path from s to t

Task 3: Maximum sum sub-array

```python
def max_subarray_sum(A):
  max_ending_here = max_so_far = A[0]

  for num in A[1:]:
    max_ending_here = max(num, max_ending_here + num)
    max_so_far = max(max_so_far, max_ending_here)

  return max_so_far

# Example usage
example_array = [-2, 1, -3, 4, -1, 2, 1, -5, 4]
print(max_subarray_sum(example_array))
```

The time complexity of this is $O(n)$ as it only iterates through the array once.

Task 4: Grid traveling

Traverse the grid from S to F by only moving right or down.

a)

S	
	F

When the grid is 2x2 there are 2 ways to get from S to F.

b)

S		
		F

When the grid is 3x3 there are 6 ways to get from S to F.

c)

S			
			F

When the grid is 4x4 there are 20 ways to get from S to F.

Observation

The pattern can be recognized from table 2 where the number in the cells represents how many different ways one can traverse the grid from S to F with the specified rules.

1	1	1	1
1	2	3	4
1	3	6	10
1	4	10	20

Table 2: Pattern