CO202 – Software Engineering – Algorithms

Dynamic Programming - Solutions

Exercise 1: Compute LCS

	j	0	1	2	3	4	5	6
i		y_j	**B**	**D**	**C**	**A**	**B**	**A**
0	x_i							
1	**A**							
2	**B**							
3	**C**							
4	**B**							
5	**D**							
6	**A**							
7	**B**							

Exercise 1: Compute LCS

	j	0	1	2	3	4	5	6
i		y_j	**B**	**D**	**C**	**A**	**B**	**A**
0	x_i	0	0	0	0	0	0	0
1	**A**	0						
2	**B**	0						
3	**C**	0						
4	**B**	0						
5	**D**	0						
6	**A**	0						
7	**B**	0						

```
for j = 1 to n
    for i = 1 to m
        if xi == yj
            c[i,j] = c[i-1,j-1] + 1
            b[i,j] = ↖
        elseif c[i-1,j] ≥ c[i,j-1]
            c[i,j] = c[i-1,j]
            b[i,j] = ↑
        else
            c[i,j] = c[i,j-1]
            b[i,j] = ←
```

Exercise 1: Compute LCS

	j	0	1	2	3	4	5	6
i		y_j	**B**	**D**	**C**	**A**	**B**	**A**
0	x_i	0	0	0	0	0	0	0
1	**A**	0	0					
2	**B**	0						
3	**C**	0						
4	**B**	0						
5	**D**	0						
6	**A**	0						
7	**B**	0						

```
for j = 1 to n
    for i = 1 to m
        if xi == yj
            c[i,j] = c[i-1,j-1] + 1
            b[i,j] = ↖
        elseif c[i-1,j] ≥ c[i,j-1]
            c[i,j] = c[i-1,j]
            b[i,j] = ↑
        else
            c[i,j] = c[i,j-1]
            b[i,j] = ←
```

Exercise 1: Compute LCS

	j	0	1	2	3	4	5	6
i		y_j	**B**	**D**	**C**	**A**	**B**	**A**
0	x_i	0	0	0	0	0	0	0
1	**A**	0	↑ 0					
2	**B**	0						
3	**C**	0						
4	**B**	0						
5	**D**	0						
6	**A**	0						
7	**B**	0						

```
for j = 1 to n
    for i = 1 to m
        if x_i == y_j
            c[i,j] = c[i-1,j-1] + 1
            b[i,j] = ↖
        elseif c[i-1,j] ≥ c[i,j-1]
            c[i,j] = c[i-1,j]
            b[i,j] = ↑
        else
            c[i,j] = c[i,j-1]
            b[i,j] = ←
```

Exercise 1: Compute LCS

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0					
2	B	0	1					
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

```
for j = 1 to n
    for i = 1 to m
        if x_i == y_j
            c[i,j] = c[i-1,j-1] + 1
            b[i,j] = ↖
        elseif c[i-1,j] ≥ c[i,j-1]
            c[i,j] = c[i-1,j]
            b[i,j] = ↑
        else
            c[i,j] = c[i,j-1]
            b[i,j] = ←
```

Exercise 1: Compute LCS

	j	0	1	2	3	4	5	6
i		y_j	**B**	**D**	**C**	**A**	**B**	**A**
0	x_i	0	0	0	0	0	0	0
1	**A**	0	↑ 0					
2	**B**	0	↖ 1					
3	**C**	0						
4	**B**	0						
5	**D**	0						
6	**A**	0						
7	**B**	0						

```
for j = 1 to n
    for i = 1 to m
        if x_i == y_j
            c[i,j] = c[i-1,j-1] + 1
            b[i,j] = ↖
        elseif c[i-1,j] ≥ c[i,j-1]
            c[i,j] = c[i-1,j]
            b[i,j] = ↑
        else
            c[i,j] = c[i,j-1]
            b[i,j] = ←
```

Exercise 1: Compute LCS

	j	0	1	2	3	4	5	6
i		y_j	**B**	**D**	**C**	**A**	**B**	**A**
0	x_i	0	0	0	0	0	0	0
1	**A**	0	↑ 0					
2	**B**	0	↖ 1					
3	**C**	0	1					
4	**B**	0						
5	**D**	0						
6	**A**	0						
7	**B**	0						

```
for j = 1 to n
    for i = 1 to m
        if x_i == y_j
            c[i,j] = c[i-1,j-1] + 1
            b[i,j] = ↖
        elseif c[i-1,j] ≥ c[i,j-1]
            c[i,j] = c[i-1,j]
            b[i,j] = ↑
        else
            c[i,j] = c[i,j-1]
            b[i,j] = ←
```

Exercise 1: Compute LCS

	j	0	1	2	3	4	5	6
i		y_j	**B**	**D**	**C**	**A**	**B**	**A**
0	x_i	0	0	0	0	0	0	0
1	**A**	0	↑ 0					
2	**B**	0	↖ 1					
3	**C**	0	↑ 1					
4	**B**	0						
5	**D**	0						
6	**A**	0						
7	**B**	0						

```
for j = 1 to n
    for i = 1 to m
        if x_i == y_j
            c[i,j] = c[i-1,j-1] + 1
            b[i,j] = ↖
        elseif c[i-1,j] ≥ c[i,j-1]
            c[i,j] = c[i-1,j]
            b[i,j] = ↑
        else
            c[i,j] = c[i,j-1]
            b[i,j] = ←
```

Exercise 1: Compute LCS

	j	0	1	2	3	4	5	6
i		y_j	**B**	**D**	**C**	**A**	**B**	**A**
0	x_i	0	0	0	0	0	0	0
1	**A**	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	**B**	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	**C**	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	**B**	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	**D**	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	**A**	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	**B**	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Exercise 1: Compute LCS

	j	0	1	2	3	4	5	6
i		y_j	**B**	**D**	**C**	**A**	**B**	**A**
0	x_i	0	0	0	0	0	0	0
1	**A**	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	**B**	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	**C**	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	**B**	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	**D**	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	**A**	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	**B**	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Exercise 1: Compute LCS

	j	0	1	2	3	4	5	6
i		y_j	**B**	**D**	**C**	**A**	**B**	**A**
0	x_i	0	0	0	0	0	0	0
1	**A**	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	**B**	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	**C**	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	**B**	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	**D**	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	**A**	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	**B**	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Exercise 1: Compute LCS

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Exercise 1: Compute LCS

	j	0	1	2				
i		y_j	**B**	**D**				
0	x_i	0	0	0				
1	**A**	0	↑ 0	↑ 0				
2	**B**	0	↖ 1	← 1				
3	**C**	0	↑ 1	↑ 1				
4	**B**	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	**D**	0	↑ 1	↖ 2	← 2	↑ 2	↑ 3	↑ 3
6	**A**	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	**B**	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ ← 4

```
for j = 1 to n
    for i = 1 to m
        if x_i == y_j
            c[i,j] = c[i-1,j-1] + 1
            b[i,j] = ↖
        elseif c[i-1,j] ≥ c[i,j-1]
            c[i,j] = c[i-1,j]
            b[i,j] = ↑
        else
            c[i,j] = c[i,j-1]
            b[i,j] = ←
```

Exercise 1: Compute LCS

	j	0	1	2	3	4	5	6
i		y_j	**B**	**D**	**C**	**A**	**B**	**A**
0	x_i	0	0	0	0	0	0	0
1	**A**	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	**B**	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	**C**	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	**B**	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	**D**	0	↑ 1	↖ 2	← 2	↑ 2	↑ 3	↑ 3
6	**A**	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	**B**	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	← 4

Exercise 1: Compute LCS

	j	0	1	2	3	4	5	6
i		y_j	**B**	**D**	**C**	**A**	**B**	**A**
0	x_i	0	0	0	0	0	0	0
1	**A**	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	**B**	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	**C**	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	**B**	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	**D**	0	↑ 1	↖ 2	↑ ← 2	↑ 2	↑ 3	↑ 3
6	**A**	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	**B**	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ ← 4

Exercise 2: Compute Levenshtein Distance

	j	0	1	2	3	4	5	6	7	8	9
i		y_j	**E**	**M**	**P**	**I**	**R**	**I**	**C**	**A**	**L**
0	x_i										
1	**I**										
2	**M**										
3	**P**										
4	**E**										
5	**R**										
6	**I**										
7	**A**										
8	**L**										

Exercise 2: Compute Levenshtein Distance

	j	0	1	2	3	4	5	6	7	8	9
i		y_j	**E**	**M**	**P**	**I**	**R**	**I**	**C**	**A**	**L**
0	x_i	0	1	2	3	4	5	6	7	8	9
1	**I**	1									
2	**M**	2									
3	**P**	3									
4	**E**	4									
5	**R**	5									
6	**I**	6									

```
for j = 1 to n
    for i = 1 to m
        c = x_i == y_j ? 0 : 1
        d[i,j] = min(d[i-1,j] + 1, d[i,j-1] + 1, d[i-1,j-1] + c)
```

Exercise 2: Compute Levenshtein Distance

	j	0	1	2	3	4	5	6	7	8	9
i		y_j	E	M	P	I	R	I	C	A	L
0	x_i	0	1	2	3	4	5	6	7	8	9
1	I	1	R 1								
2	M	2									
3	P	3									
4	E	4									
5	R	5									
6	I	6									

```
for j = 1 to n
    for i = 1 to m
        c = x_i == y_j ? 0 : 1
        d[i,j] = min(d[i-1,j] + 1, d[i,j-1] + 1, d[i-1,j-1] + c)
```

Exercise 2: Compute Levenshtein Distance

	j	0	1	2	3	4	5	6	7	8	9
i		y_j	**E**	**M**	**P**	**I**	**R**	**I**	**C**	**A**	**L**
0	x_i	0	1	2	3	4	5	6	7	8	9
1	**I**	1	R 1								
2	**M**	2	R D 2								
3	**P**	3									
4	**E**	4									
5	**R**	5									
6	**I**	6									

```
for j = 1 to n
    for i = 1 to m
        c = x_i == y_j ? 0 : 1
        d[i,j] = min(d[i-1,j] + 1, d[i,j-1] + 1, d[i-1,j-1] + c)
```

Exercise 2: Compute Levenshtein Distance

	j	0	1	2	3	4	5	6	7	8	9
i		y_j	E	M	P	I	R	I	C	A	L
0	x_i	0	1	2	3	4	5	6	7	8	9
1	I	1	R 1								
2	M	2	R D 2								
3	P	3	R D 3								
4	E	4									
5	R	5									
6	I	6									

```
for j = 1 to n
    for i = 1 to m
        c = x_i == y_j ? 0 : 1
        d[i,j] = min(d[i-1,j] + 1, d[i,j-1] + 1, d[i-1,j-1] + c)
```

Exercise 2: Compute Levenshtein Distance

	j	0	1	2	3	4	5	6	7	8	9
i		y_j	**E**	**M**	**P**	**I**	**R**	**I**	**C**	**A**	**L**
0	x_i	0	1	2	3	4	5	6	7	8	9
1	**I**	1	R 1								
2	**M**	2	R D 2								
3	**P**	3	R D 3								
4	**E**	4	K 3								
5	**R**	5									
6	**I**	6									

```
for j = 1 to n
    for i = 1 to m
        c = x_i == y_j ? 0 : 1
        d[i,j] = min(d[i-1,j] + 1, d[i,j-1] + 1, d[i-1,j-1] + c)
```

Exercise 2: Compute Levenshtein Distance

	j	0	1	2	3	4	5	6	7	8	9
i		y_j	E	M	P	I	R	I	C	A	L
0	x_i	0	1	2	3	4	5	6	7	8	9
1	I	1	R 1								
2	M	2	R D 2								
3	P	3	R D 3								
4	E	4	K 3								
5	R	5	D 4								
6	I	6									

```
for j = 1 to n
    for i = 1 to m
        c = x_i == y_j ? 0 : 1
        d[i,j] = min(d[i-1,j] + 1, d[i,j-1] + 1, d[i-1,j-1] + c)
```

Exercise 2: Compute Levenshtein Distance

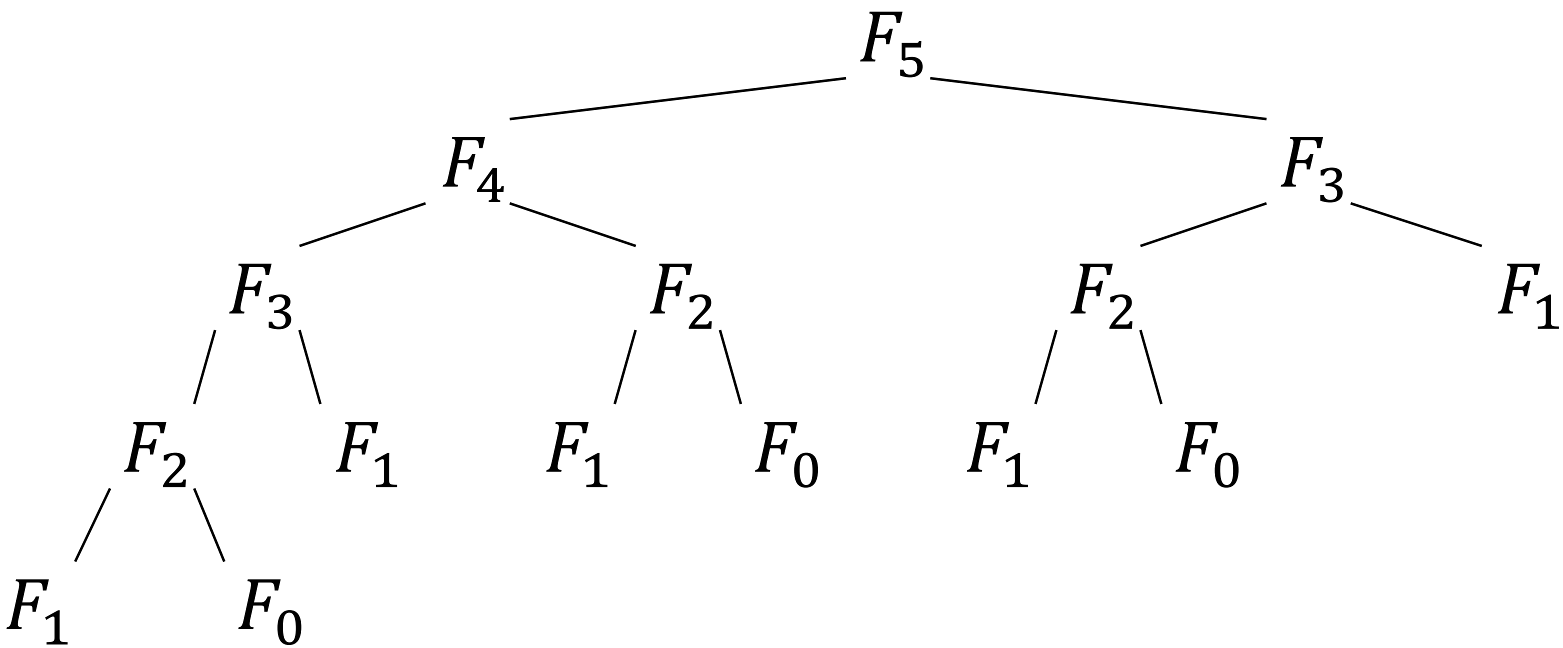
I	M	P	E	R	I	-	A	L
E	M	P	I	R	I	C	A	L
REPLACE	KEEP	KEEP	REPLACE	KEEP	KEEP	INSERT	KEEP	KEEP

	j	0	1	2	3	4	5	6	7	8	9
i		y_j	E	M	P	I	R	I	C	A	L
0	x_i	0	I 1	I 2	I 3	I 4	I 5	I 6	I 7	I 8	I 9
1	I	D 1	R 1	R I 2	R I 3	K 3	I 4	K I 5	I 6	I 7	I 8
2	M	D 2	R D 2	K 1	I 2	I 3	R I 4	R I 5	R I 6	R I 7	R I 8
3	P	D 3	R D 3	D 2	K 1	I 2	I 3	I 4	I 5	I 6	I 7
4	E	D 4	K 3	D 3	D 2	R 2	R I 3	R I 4	R I 5	R I 6	R I 7
5	R	D 5	D 4	R D 4	D 3	R D 3	K 2	I 3	I 4	I 5	I 6
6	I	D 6	D 5	R D 5	D 4	K 3	D 3	K 2	I 3	I 4	I 5
7	A	D 7	D 6	R D 6	D 5	D 4	R D 4	D 3	R 3	K 3	I 4
8	L	D 8	D 7	R D 7	D 6	D 5	R D 5	D 4	R D 4	R D 4	K 3

Exercise 3: Fibonacci Sequence

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, …

$$F(n) = \begin{cases} 0, & \text{if } n = 0 \\ 1, & \text{if } n = 1 \\ F(n-1) + F(n-2), & \text{otherwise} \end{cases}$$



Exercise 3: Fibonacci Sequence

```
NAÏVE-FIBONACCI(n)
if n == 0
    return 0
if n == 1
    return 1
return NAÏVE-FIBONACCI(n-1) + NAÏVE-FIBONACCI(n-2)
```

Running time of NAÏVE-FIBONACCI:

$$T(n) = O(2^{0.694n})$$

Exercise 3: Fibonacci Sequence

```
BOTTOM-UP-FIBONACCI(n)
  if n == 0
      return 0
  let f[0..n] be a new array
  f[0] = 0
  f[1] = 1
  ?
  ?
  ?
```

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, …

$$F(n) = \begin{cases} 0, & \text{if } n = 0 \\ 1, & \text{if } n = 1 \\ F(n-1) + F(n-2), & \text{otherwise} \end{cases}$$

What is the running time of BOTTOM-UP-FIBONACCI?

Exercise 3: Fibonacci Sequence

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, …

$$F(n) = \begin{cases} 0, & \text{if } n = 0 \\ 1, & \text{if } n = 1 \\ F(n-1) + F(n-2), & \text{otherwise} \end{cases}$$

```
BOTTOM-UP-FIBONACCI(n)
if n == 0
    return 0
let f[0..n] be a new array
f[0] = 0
f[1] = 1
for i = 2 to n
    f[i] = f[i-1] + f[i-2]
return f[n]
```

What is the running time of BOTTOM-UP-FIBONACCI?

$T(n) = O(n)$ …or?

Exercise 4: Coin Change Problem

Coin change is the problem of finding the least number of coins for a given amount of money.

For example, the UK coin set contains the following coins:

- 1p, 2p, 5p, 10p, 20p, 50p, £1, £2, and £5 (very uncommon).
- For £2.82, the optimal change is £2, 50p, 20p, 10p, 2p, so 5 coins.

1. Write a mathematical recurrence equation that determines the least number of coins.

2. Devise a pseudo-code, bottom-up dynamic programming algorithm `coin_change(n,coins)`.

Exercise 4: Coin Change Problem

1. Write a mathematical recurrence equation that determines the least number of coins.

$$counts[n] = \begin{cases} 0, & \text{if } n = 0 \\ \min\limits_{coin}(counts[n - coin] + 1)\,, & \text{otherwise} \end{cases}$$

Exercise 4: Coin Change Problem

2. Devise a pseudo-code, bottom-up dynamic programming algorithm `coin_change(n,coins)`.

```
def coin_change(n,coins):
    if n == 0:
        return 0
    counts = [0]*(n+1)
    change = [0]*(n+1)
    for i in range(1,n+1):
        counts[i] = n+1
        for coin in coins:
            if coin <= i:
                count = 1 + counts[i-coin]
                if count < counts[i]:
                    counts[i] = count
                    change[i] = coin
    return counts, change
```

Exercise 5: Fibonacci Challenge

D&C Fibonacci Revisited

Naïve:

$$F(n) = \begin{cases} 0, & \text{if } n = 0 \\ 1, & \text{if } n = 1 \\ F(n-1) + F(n-2), & \text{otherwise} \end{cases}$$

Let's rewrite Fibonacci

$$F(n) = F(2k) = F(k)^2 + 2F(k)F(k-1) \quad \text{for even } n$$

$$F(n) = F(2k-1) = F(k)^2 + F(k-1)^2 \quad \text{for odd } n$$

$$F(n) = \begin{cases} 0, & \text{if } n = 0 \\ 1, & \text{if } n = 1 \\ F(\lceil n/2 \rceil)^2 + 2F(\lceil n/2 \rceil)F(\lceil n/2 \rceil - 1), & \text{if } n \geq 2 \text{ and } n \text{ is even} \\ F(\lceil n/2 \rceil)^2 + F(\lceil n/2 \rceil - 1)^2, & \text{if } n \geq 2 \text{ and } n \text{ is odd} \end{cases}$$

Exercise 5: Fibonacci Challenge

D&C Fibonacci Revisited

$$F(n) = \begin{cases} 0, & \text{if } n = 0 \\ 1, & \text{if } n = 1 \\ \boldsymbol{F}(\lceil \boldsymbol{n}/\mathbf{2} \rceil)^2 + 2\boldsymbol{F}(\lceil \boldsymbol{n}/\mathbf{2} \rceil)\boldsymbol{F}(\lceil \boldsymbol{n}/\mathbf{2} \rceil - \mathbf{1}), & \text{if } n \geq 2 \text{ and } n \text{ is even} \\ \boldsymbol{F}(\lceil \boldsymbol{n}/\mathbf{2} \rceil)^2 + \boldsymbol{F}(\lceil \boldsymbol{n}/\mathbf{2} \rceil - \mathbf{1})^2, & \text{if } n \geq 2 \text{ and } n \text{ is odd} \end{cases}$$

```
DC-FIBONACCI(n)
if n == 0 || n == 1
    return n
else
    a = DC-FIBONACCI((n+1)/2)
    b = DC-FIBONACCI((n+1)/2-1)
    if n is even
        return a * ( a + 2 * b)
    else
        return a * a + b * b
```

$$T(n) = 2T(n/2) + \Theta(1)$$

Case 1: $d < \log_b a$, then $T(n) = O\left(n^{\log_b a}\right)$

Exercise 5: Fibonacci Challenge

Dynamic Programming vs. Divide-and-Conquer

```
BOTTOM-UP-FIBONACCI(n)
if n == 0
    return 0
let f[0..n] be a new array
f[0] = 0
f[1] = 1
for i = 2 to n
    f[i] = f[i-1] + f[i-2]
return f[n]
```

```
DC-FIBONACCI(n)
if n == 0 || n == 1
    return n
else
    a = DC-FIBONACCI((n+1)/2)
    b = DC-FIBONACCI((n+1)/2-1)
    if n is even
        return a * ( a + 2 * b)
    else
        return a * a + b * b
```

Asymptotic running time: $O(n)$

But, who wins in practice?