

FIT2004

Algorithms and Data Structures

Ian Wern Han Lim
lim.wern.han@monash.edu

Referencing materials by
Nathan Compane, Aamir Cheema, Arun Konagurthu and Lloyd Allison



Ready?

Agenda

- Complexity Analysis
 - Big-O
 - Recurrence relation

Agenda

- Complexity Analysis
 - Big-O
 - Recurrence relation

Agenda

- Complexity Analysis
 - Big-O
 - Recurrence relation
- } Covered in Tutorial 02
using tutorial questions
as case study

Let us begin...

Complexity Analysis

Recap

- You have done time complexity last time

Complexity Analysis

Recap

- You have done time complexity last time
- You know what Big-O is

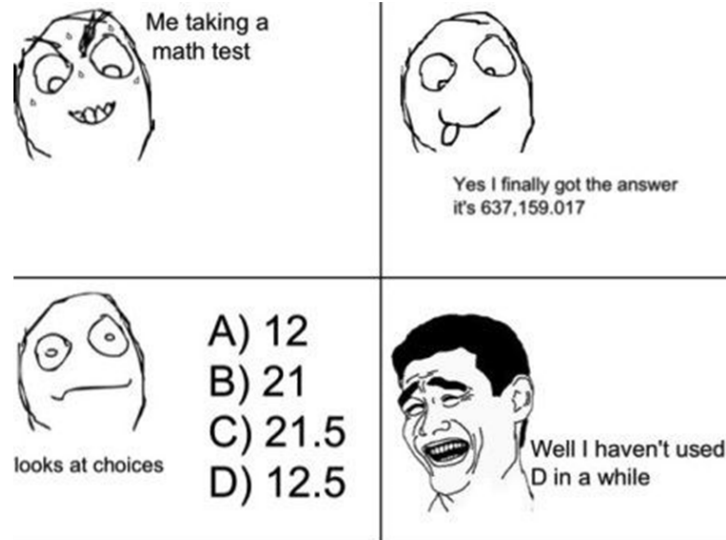
Complexity Analysis

Recap

- You have done time complexity last time
- You know what Big-O is

What is the complexity of an algorithm in Big-O notation that runs in $30N \log(N^2) + 10 \log N + 8N$?

- A. $O(N \log N)$
- B. $O(N \log(N^2))$
- C. $O(N \log(N^2) + N + \log N)$
- D. Option D because



What is your
answer?

Complexity Analysis

Recap

- You have done time complexity last time
- You know what Big-O is

What is the complexity of an algorithm in Big-O notation that runs in $30N \log(N^2) + 10 \log N + 8N$?

- A. $O(N \log N)$
- B. $O(N \log(N^2))$
- C. $O(N \log(N^2) + N + \log N)$
- D. Option D because



Me taking a math test

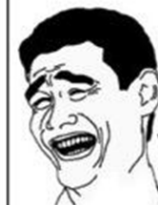


Yes I finally got the answer
it's 637,159.017



looks at choices

A) 12
B) 21
C) 21.5
D) 12.5



Well I haven't used
D in a while

Complexity Analysis

Recap

- You have done time complexity last time
- You know what Big-O is

What is the complexity of an algorithm in Big-O notation that runs in $30N \log(N^2) + 10 \log N + 8N$?

- A. $O(N \log N)$
- B. $O(N \log(N^2))$
- C. $O(N \log(N^2) + N + \log N)$
- D. Option D because



Me taking a math test



Yes I finally got the answer
it's 637,159.017



looks at choices

A) 12
B) 21
C) 21.5
D) 12.5



Well I haven't used
D in a while

Complexity Analysis

Recap

- You have done time complexity last time
- You know what Big-O is

What is the complexity of an algorithm in Big-O notation that runs in $30N \log(N^2) + 10 \log N + 8N$?

- A. $O(N \log N)$
- B. $O(N \log(N^2))$
- C. $O(N \log(N^2) + N + \log N)$
- D. Option D because →



Me taking a math test



Yes I finally got the answer
it's 637,159.017



looks at choices

A) 12
B) 21
C) 21.5
D) 12.5



Well I haven't used
D in a while

Complexity Analysis

Recap

- You have done time complexity last time
- You know what Big-O is

What is the complexity of an algorithm in Big-O notation that runs in $30N \log(N^2) + 10 \log N + 8N$?

- A. $O(N \log N)$
- B. $O(N \log(N^2))$
- C. $O(N \log(N^2) + N + \log N)$
- D. Option D because



Me taking a math test



Yes I finally got the answer
it's 637,159.017



looks at choices

A) 12
B) 21
C) 21.5
D) 12.5



Well I haven't used
D in a while

$$\log N^2 = 2 \log N$$

Complexity Analysis

Recap

- You have done time complexity last time
- You know what Big-O is
- If I have a list, and I want to sort it with bubble sort...

Complexity Analysis

Recap

- You have done time complexity last time
- You know what Big-O is
- If I have a list, and I want to sort it with bubble sort...
 - Best case?
 - Worst case?

Complexity Analysis

Recap

- You have done time complexity last time
- You know what Big-O is
- If I have a list, and I want to sort it with bubble sort...
 - Best case? $O(1)$ when list is empty
 - Worst case? $O(N^2)$ when list is sorted in reverse order

Complexity Analysis

Recap

- You have done time complexity last time
- You know what Big-O is
- If I have a list, and I want to sort it with bubble sort...
 - Best case? $O(1)$ when list is empty
 - Worst case? $O(N^2)$ when list is sorted in reverse order



Complexity Analysis

Recap

- You have done time complexity last time
- You know what Big-O is

$n > k$, n is larger and significantly larger than k

big-O notation not care small output

- If I have a list, and I want to sort it with bubble sort...
 - Best case? $O(N)$ when list is sorted and we can terminate earlier
 - Worst case? $O(N^2)$ when list is sorted in reverse order

$N > k$
Big-O complexity
only cares about the large N
scalable part



Complexity Analysis

Recap

- You have done time complexity last time
- You know what Big-O is

you vs. the guy she tells
you not to worry about

$O(n^2)$

$O(1)$

Complexity Analysis

Recap

- You have done time complexity last time
- You know what Big-O is
- So what's new?

Complexity Analysis

Recap

- You have done time complexity last time
- You know what Big-O is
- So what's new?
 - Space complexity
 - Time complexity for recursion

Complexity Analysis

Recap

- You have done time complexity last time
- You know what Big-O is
- So what's new?
 - Space complexity
 - Time complexity for recursion
 - By solving recurrence relation

Complexity Analysis

Recap

- You have done time complexity last time
- You know what Big-O is
- So what's new?
 - Space complexity
 - We'll leave this for Lecture 02 when we go through some sorting algorithms
 - Time complexity for recursion
 - By solving recurrence relation

Questions?

Complexity Analysis

Recurrence relation

- This is asked a lot
 - Final exam

- This is asked a lot
 - Final exam
- Helps you figure out the complexity of recursive functions

- Look at the algorithm written in Python

```
1 def power(x,n):  
2     """  
3     Returns x^n calculated recursively  
4     """  
5     if n == 0:  
6         return 1  
7     elif n == 1:  
8         return x  
9     else:  
10        return x * power (x,n-1)
```

- Look at the algorithm written in Python
- Note: Can you explain what this function do? ... and reason out the complexity?
 - Using FIT1008 knowledge

```
1 def power(x,n):  
2     """  
3     Returns x^n calculated recursively  
4     """  
5     if n == 0:  
6         return 1  
7     elif n == 1:  
8         return x  
9     else:  
10        return x * power (x,n-1)
```

- Look at the algorithm written in Python
- What is the recurrence relation?

```
1 def power(x,n):  
2     """  
3     Returns x^n calculated recursively  
4     """  
5     if n == 0:  
6         return 1  
7     elif n == 1:  
8         return x  
9     else:  
10        return x * power (x,n-1)
```

- Look at the algorithm written in Python
- What is the recurrence relation? $T(N)$
 - Base case
 - Recursive case

```
1 def power(x,n):  
2     """  
3     Returns x^n calculated recursively  
4     """  
5     if n == 0:  
6         return 1  
7     elif n == 1:  
8         return x  
9     else:  
10        return x * power (x,n-1)
```

- Look at the algorithm written in Python
- What is the recurrence relation? $T(N)$
 - Base case
 - Recursive case

```
1 def power(x,n):  
2     """  
3     Returns x^n calculated recursively  
4     """  
5     if n == 0:  
6         return 1  
7     elif n == 1:  
8         return x  
9     else:  
10        return x * power (x,n-1)
```

- Look at the algorithm written in Python
- What is the recurrence relation? $T(N)$
 - Base case
 $T(0) = a$
 - Recursive case

```
1 def power(x,n):  
2     """  
3     Returns x^n calculated recursively  
4     """  
5     if n == 0:  
6         return 1  
7     elif n == 1:  
8         return x  
9     else:  
10        return x * power (x,n-1)
```


- Look at the algorithm written in Python
- What is the recurrence relation? $T(N)$
 - Base case
 - $T(0) = a$
 - $T(1) = b$
 - Recursive case

```
1 def power(x,n):  
2     """  
3     Returns x^n calculated recursively  
4     """  
5     if n == 0:  
6         return 1  
7     elif n == 1:  
8         return x  
9     else:  
10        return x * power (x,n-1)
```

- Look at the algorithm written in Python
- What is the recurrence relation? $T(N)$
 - Base case
 - $T(0) = a$
 - $T(1) = b$
 - Constant operating cost...
 - Recursive case

```
1 def power(x,n):  
2     """  
3     Returns x^n calculated recursively  
4     """  
5     if n == 0:  
6         return 1  
7     elif n == 1:  
8         return x  
9     else:  
10        return x * power (x,n-1)
```

- Look at the algorithm written in Python
- What is the recurrence relation? $T(N)$
 - Base case
 - $T(0) = a$
 - $T(1) = b$
 - Constant operating cost...
 - Recursive case
 - $T(N) = T(N-1)$

```
1 def power(x,n):  
2     """  
3     Returns x^n calculated recursively  
4     """  
5     if n == 0:  
6         return 1  
7     elif n == 1:  
8         return x  
9     else:  
10        return x * power (x,n-1)
```

- Look at the algorithm written in Python
- What is the recurrence relation? $T(N)$
 - Base case
 - $T(0) = a$
 - $T(1) = b$
 - Constant operating cost...
 - Recursive case
 - $T(N) = T(N-1) * X?$

```
1 def power(x,n):  
2     """  
3     Returns x^n calculated recursively  
4     """  
5     if n == 0:  
6         return 1  
7     elif n == 1:  
8         return x  
9     else:  
10        return x * power (x,n-1)
```

- Look at the algorithm written in Python
- What is the recurrence relation? $T(N)$

- Base case

$$T(0) = a$$

$$T(1) = b$$

Constant operating cost...

- Recursive case

$$T(N) = T(N-1) + c \quad \text{constant constant operating cost in multiplying from } x * \text{power}(x, n-1)$$

Cause of constant operating cost in multiplying...

```
1 def power(x,n):  
2     """  
3     Returns x^n calculated recursively  
4     """  
5     if n == 0:  
6         return 1  
7     elif n == 1:  
8         return x  
9     else:  
10        return x * power(x,n-1)
```

just a simple constant
constant operating cost in multiplying

- Look at the algorithm written in Python
- What is the recurrence relation? $T(N)$
 - Base case
 - $T(0) = a$
 - $T(1) = b$
 - Constant operating cost...
 - Recursive case (general case)
 - $T(N) = T(N-1) + c$
 - Cause of constant operating cost in multiplying...

```
1 def power(x,n):  
2     """  
3     Returns x^n calculated recursively  
4     """  
5     if n == 0:  
6         return 1  
7     elif n == 1:  
8         return x  
9     else:  
10        return x * power (x,n-1)
```

- $T(0) = a$
- $T(1) = b$
- $T(N) = T(N-1) + c$

```
1  def power(x,n):  
2      """  
3      Returns x^n calculated recursively  
4      """  
5      if n == 0:  
6          return 1  
7      elif n == 1:  
8          return x  
9      else:  
10         return x * power (x,n-1)
```

- $T(0) = a$
- $T(1) = b$
- $T(N) = T(N-1) + c$
- Now solve it for the complexity

```
1 def power(x,n):  
2     """  
3     Returns x^n calculated recursively  
4     """  
5     if n == 0:  
6         return 1  
7     elif n == 1:  
8         return x  
9     else:  
10        return x * power (x,n-1)
```


- $T(0) = a$
- $T(1) = b$
- $T(N) = T(N-1) + c$

- Now solve it for the complexity
 - $T(N) = T(N-1) + c$
 - so $T(N-1) = T(N-2) + c$

```
1 def power(x,n):  
2     """  
3     Returns x^n calculated recursively  
4     """  
5     if n == 0:  
6         return 1  
7     elif n == 1:  
8         return x  
9     else:  
10        return x * power (x,n-1)
```

- $T(0) = a$
- $T(1) = b$
- $T(N) = T(N-1) + c$

- Now solve it for the complexity
 - $T(N) = T(N-1) + c$
 - so $T(N-1) = T(N-2) + c$
 - $T(N) = T(N-2) + c + c$
 - $T(N) = T(N-3) + c + c + c$

```
1 def power(x,n):  
2     """  
3     Returns x^n calculated recursively  
4     """  
5     if n == 0:  
6         return 1  
7     elif n == 1:  
8         return x  
9     else:  
10        return x * power (x,n-1)
```

- $T(0) = a$
- $T(1) = b$
- $T(N) = T(N-1) + c$
- Now solve it for the complexity

- $T(N) = T(N-1) + c$
- so $T(N-1) = T(N-2) + c$
- $T(N) = T(N-2) + c + c$
- $T(N) = T(N-3) + c + c + c$

} Known as telescoping

```
1 def power(x,n):  
2     """  
3     Returns x^n calculated recursively  
4     """  
5     if n == 0:  
6         return 1  
7     elif n == 1:  
8         return x  
9     else:  
10        return x * power (x,n-1)
```

- $T(0) = a$
- $T(1) = b$
- $T(N) = T(N-1) + c$

- Now solve it for the complexity
 - $T(N) = T(N-1) + c$
 - so $T(N-1) = T(N-2) + c$
 - $T(N) = T(N-2) + c + c$
 - $T(N) = T(N-3) + c + c + c$
 - generalized into $T(N) = T(N-k) + kc$
pattern

```
1 def power(x,n):  
2     """  
3     Returns x^n calculated recursively  
4     """  
5     if n == 0:  
6         return 1  
7     elif n == 1:  
8         return x  
9     else:  
10        return x * power (x,n-1)
```

- $T(0) = a$
- $T(1) = b$
- $T(N) = T(N-1) + c$

- Now solve it for the complexity
 - $T(N) = T(N-1) + c$
 - so $T(N-1) = T(N-2) + c$
 - $T(N) = T(N-2) + c + c$
 - $T(N) = T(N-3) + c + c + c$
 - generalized into $T(N) = T(N-k) + kc$
 - Base case when $N=0$ or $N=1$

```
1 def power(x,n):  
2     """  
3     Returns x^n calculated recursively  
4     """  
5     if n == 0:  
6         return 1  
7     elif n == 1:  
8         return x  
9     else:  
10        return x * power (x,n-1)
```

- $T(0) = a$
- $T(1) = b$
- $T(N) = T(N-1) + c$

- Now solve it for the complexity
 - $T(N) = T(N-1) + c$
 - so $T(N-1) = T(N-2) + c$
 - $T(N) = T(N-2) + c + c$
 - $T(N) = T(N-3) + c + c + c$
 - generalized into $T(N) = T(N-k) + kc$
 - Base case when $N=0$ or $N=1$
 - $N-k = 0$, therefore base case when $k=N$. we replace into above...

```
1 def power(x,n):  
2     """  
3     Returns x^n calculated recursively  
4     """  
5     if n == 0:  
6         return 1  
7     elif n == 1:  
8         return x  
9     else:  
10        return x * power (x,n-1)
```

- $T(0) = a$
- $T(1) = b$
- $T(N) = T(N-1) + c$
- Now solve it for the complexity
 - $T(N) = T(N-1) + c$
 - so $T(N-1) = T(N-2) + c$
 - $T(N) = T(N-2) + c + c$
 - $T(N) = T(N-3) + c + c + c$
 - generalized into $T(N) = T(N-k) + kc$
 - Base case when $N=0$ or $N=1$
 - $N-k = 0$, therefore base case when $k=N$. we replace into above...
 - $T(N) = T(N-N) + Nc = T(0) + Nc = a + Nc$

```
1 def power(x,n):  
2     """  
3     Returns x^n calculated recursively  
4     """  
5     if n == 0:  
6         return 1  
7     elif n == 1:  
8         return x  
9     else:  
10        return x * power(x,n-1)
```

- $T(0) = a$
- $T(1) = b$
- $T(N) = T(N-1) + c$
- Now solve it for the complexity
 - $T(N) = T(N-1) + c$
 - so $T(N-1) = T(N-2) + c$
 - $T(N) = T(N-2) + c + c$
 - $T(N) = T(N-3) + c + c + c$
 - generalized into $T(N) = T(N-k) + kc$
 - Base case when $N=0$ or $N=1$
 - $N-k = 0$, therefore base case when $k=N$. we replace into above...
 - $T(N) = T(N-N) + Nc = T(0) + Nc = a + Nc = O(N)$, eliminating the constant

```
1 def power(x,n):
2     """
3     Returns x^n calculated recursively
4     """
5     if n == 0:
6         return 1
7     elif n == 1:
8         return x
9     else:
10        return x * power(x,n-1)
```


Questions?

- Let us try another one
- Come up with the recurrence relation

```
12 def power_squaring(x,n):
13     """
14     Returns x^n
15     via exponential by squaring
16     """
17     if n == 0:
18         return 1
19     elif n == 1:
20         return x
21     elif n%2 == 0:
22         return power (x*x, n//2)
23     elif n%2 == 1:
24         return power (x*x, n//2) * n
```

- Let us try another one
- Come up with the recurrence relation
- Fun fact, this was a programming question for the exam...

```
12 def power_squaring(x,n):
13     """
14     Returns x^n
15     via exponential by squaring
16     """
17     if n == 0:
18         return 1
19     elif n == 1:
20         return x
21     elif n%2 == 0:
22         return power(x*x, n//2)
23     elif n%2 == 1:
24         return power(x*x, n//2) * x
```

- Let us try another one
- Come up with the recurrence relation
- Fun fact, this was a programming question for the exam...
- Do you understand the code?

```
12 def power_squaring(x,n):
13     """
14     Returns x^n
15     via exponential by squaring
16     """
17     if n == 0:
18         return 1
19     elif n == 1:
20         return x
21     # when power is even
22     # N^4 = N^2 * N^2
23     elif n%2 == 0:
24         return power(x*x, n//2)
25     # when the power is odd
26     # N^9 = N^4 & N^4 * N
27     elif n%2 == 1:
28         return power(x*x, n//2) * x
```

Complexity Analysis

Recurrence relation

- Let us try another one
- Come up with the recurrence relation
- Fun fact, this was a programming question for the exam...
- Do you understand the code?

```
12 def power_squaring(x,n):
13     """
14     Returns x^n
15     via exponential by squaring
16     """
17     if n == 0:
18         return 1
19     elif n == 1:
20         return x
21     # when power is even
22     # N^4 = N^2 * N^2
23     elif n%2 == 0:
24         return power(x*x, n//2)
25     # when the power is odd
26     # N^9 = N^4 & N^4 * N
27     elif n%2 == 1:
28         return power(x*x, n//2) * n
```

Should be *x

Complexity Analysis

Recurrence relation

- $T(0) = a$

```
12 def power_squaring(x,n):
13     """
14     Returns x^n
15     via exponential by squaring
16     """
17     if n == 0:
18         return 1
19     elif n == 1:
20         return x
21     # when power is even
22     # N^4 = N^2 * N^2
23     elif n%2 == 0:
24         return power(x*x, n//2)
25     # when the power is odd
26     # N^9 = N^4 & N^4 * N
27     elif n%2 == 1:
28         return power(x*x, n//2) * n
```

Complexity Analysis

Recurrence relation

- $T(0) = a$
- $T(1) = b$

```
12 def power_squaring(x,n):
13     """
14     Returns x^n
15     via exponential by squaring
16     """
17     if n == 0:
18         return 1
19     elif n == 1:
20         return x
21     # when power is even
22     # N^4 = N^2 * N^2
23     elif n%2 == 0:
24         return power(x*x, n//2)
25     # when the power is odd
26     # N^9 = N^4 & N^4 * N
27     elif n%2 == 1:
28         return power(x*x, n//2) * n
```

Complexity Analysis

Recurrence relation

- $T(0) = a$
- $T(1) = b$
- $T(N) = T(N//2) + c$ when N even

```
12 def power_squaring(x,n):
13     """
14     Returns x^n
15     via exponential by squaring
16     """
17     if n == 0:
18         return 1
19     elif n == 1:
20         return x
21     # when power is even
22     # N^4 = N^2 * N^2
23     elif n%2 == 0:
24         return power(x*x, n//2)
25     # when the power is odd
26     # N^9 = N^4 & N^4 * N
27     elif n%2 == 1:
28         return power(x*x, n//2) * n
```


- $T(0) = a$
- $T(1) = b$
- $T(N) = T(N//2) + c$ when N even
- $T(N) = T(N//2) + d$ when N odd

```
12 def power_squaring(x,n):
13     """
14     Returns x^n
15     via exponential by squaring
16     """
17     if n == 0:
18         return 1
19     elif n == 1:
20         return x
21     # when power is even
22     # N^4 = N^2 * N^2
23     elif n%2 == 0:
24         return power(x*x, n//2)
25     # when the power is odd
26     # N^9 = N^4 & N^4 * N
27     elif n%2 == 1:
28         return power(x*x, n//2) * n
```

- $T(0) = a$
- $T(1) = b$
- $T(N) = T(N//2) + c$ when N even
- $T(N) = T(N//2) + d$ when N odd

this is output we just care about complexity

– Not that it isn't $T(N) = T(N//2) * N$

WHY?

```
12 def power_squaring(x,n):
13     """
14     Returns x^n
15     via exponential by squaring
16     """
17     if n == 0:
18         return 1
19     elif n == 1:
20         return x
21     # when power is even
22     # N^4 = N^2 * N^2
23     elif n%2 == 0:
24         return power(x*x, n//2)
25     # when the power is odd
26     # N^9 = N^4 & N^4 * N
27     elif n%2 == 1:
28         return power(x*x, n//2) * n
```

- $T(0) = a$
- $T(1) = b$
- $T(N) = T(N//2) + c$ when N even
- $T(N) = T(N//2) + d$ when N odd
- Now you solve it...

```
12 def power_squaring(x,n):
13     """
14     Returns x^n
15     via exponential by squaring
16     """
17     if n == 0:
18         return 1
19     elif n == 1:
20         return x
21     # when power is even
22     # N^4 = N^2 * N^2
23     elif n%2 == 0:
24         return power(x*x, n//2)
25     # when the power is odd
26     # N^9 = N^4 & N^4 * N
27     elif n%2 == 1:
28         return power(x*x, n//2) * n
```

- $T(0) = a$
- $T(1) = b$
- $T(N) = T(N//2) + c$ when N even
- $T(N) = T(N//2) + d$ when N odd
- Now you solve it...
Let me take the even one to start

```
12 def power_squaring(x,n):
13     """
14     Returns x^n
15     via exponential by squaring
16     """
17     if n == 0:
18         return 1
19     elif n == 1:
20         return x
21     # when power is even
22     # N^4 = N^2 * N^2
23     elif n%2 == 0:
24         return power(x*x, n//2)
25     # when the power is odd
26     # N^9 = N^4 & N^4 * N
27     elif n%2 == 1:
28         return power(x*x, n//2) * n
```

- $T(0) = a$
- $T(1) = b$
- $T(N) = T(N//2) + c$ when N even
- $T(N) = T(N//2) + d$ when N odd
- Now you solve it...
Let me take the even one to start
 - $T(N) = T(N//2) + c$
 - $T(N) = T(N//4) + 2c$
 - $T(N) = T(N//8) + 3c$

```
12 def power_squaring(x,n):
13     """
14     Returns x^n
15     via exponential by squaring
16     """
17     if n == 0:
18         return 1
19     elif n == 1:
20         return x
21     # when power is even
22     # N^4 = N^2 * N^2
23     elif n%2 == 0:
24         return power(x*x, n//2)
25     # when the power is odd
26     # N^9 = N^4 & N^4 * N
27     elif n%2 == 1:
28         return power(x*x, n//2) * n
```

- $T(0) = a$
- $T(1) = b$
- $T(N) = T(N//2) + c$ when N even
- $T(N) = T(N//2) + d$ when N odd

- Now you solve it...
Let me take the even one to start
 - $T(N) = T(N//2) + c$
 - $T(N) = T(N//4) + 2c$
 - $T(N) = T(N//8) + 3c$ can you see the pattern?

```
12 def power_squaring(x,n):
13     """
14     Returns x^n
15     via exponential by squaring
16     """
17     if n == 0:
18         return 1
19     elif n == 1:
20         return x
21     # when power is even
22     # N^4 = N^2 * N^2
23     elif n%2 == 0:
24         return power(x*x, n//2)
25     # when the power is odd
26     # N^9 = N^4 & N^4 * N
27     elif n%2 == 1:
28         return power(x*x, n//2) * n
```

- $T(0) = a$
- $T(1) = b$
- $T(N) = T(N//2) + c$ when N even
- $T(N) = T(N//2) + d$ when N odd

- Now you solve it...
Let me take the even one to start
 - $T(N) = T(N//2) + c$
 - $T(N) = T(N//4) + 2c$
 - $T(N) = T(N//8) + 3c$ can you see the pattern?
 - $T(N) = T(N//2^k) + kc$

```
12 def power_squaring(x,n):
13     """
14     Returns x^n
15     via exponential by squaring
16     """
17     if n == 0:
18         return 1
19     elif n == 1:
20         return x
21     # when power is even
22     # N^4 = N^2 * N^2
23     elif n%2 == 0:
24         return power(x*x, n//2)
25     # when the power is odd
26     # N^9 = N^4 & N^4 * N
27     elif n%2 == 1:
28         return power(x*x, n//2) * n
```

- $T(0) = a$
- $T(1) = b$
- $T(N) = T(N//2) + c$ when N even
- $T(N) = T(N//2) + d$ when N odd
- Now you solve it...
Let me take the even one to start
 - $T(N) = T(N//2) + c$
 - $T(N) = T(N//4) + 2c$
 - $T(N) = T(N//8) + 3c$ can you see the pattern?
 - $T(N) = T(N//2^k) + kc$
 - Base when $N//2^k = 1$... thus $N = 2^k$

```
12 def power_squaring(x,n):
13     """
14     Returns x^n
15     via exponential by squaring
16     """
17     if n == 0:
18         return 1
19     elif n == 1:
20         return x
21     # when power is even
22     # N^4 = N^2 * N^2
23     elif n%2 == 0:
24         return power(x*x, n//2)
25     # when the power is odd
26     # N^9 = N^4 & N^4 * N
27     elif n%2 == 1:
28         return power(x*x, n//2) * n
```


- $T(0) = a$
- $T(1) = b$
- $T(N) = T(N//2) + c$ when N even
- $T(N) = T(N//2) + d$ when N odd

- Now you solve it...

Let me take the even one to start

- $T(N) = T(N//2) + c$
- $T(N) = T(N//4) + 2c$
- $T(N) = T(N//8) + 3c$ can you see the pattern?
- $T(N) = T(N//2^k) + kc$
- Base when $N//2^k = 1$... thus $N = 2^k$ which is $k = \log N$

```
12 def power_squaring(x,n):
13     """
14     Returns x^n
15     via exponential by squaring
16     """
17     if n == 0:
18         return 1
19     elif n == 1:
20         return x
21     # when power is even
22     # N^4 = N^2 * N^2
23     elif n%2 == 0:
24         return power(x*x, n//2)
25     # when the power is odd
26     # N^9 = N^4 & N^4 * N
27     elif n%2 == 1:
28         return power(x*x, n//2) * n
```

- $T(0) = a$
- $T(1) = b$
- $T(N) = T(N//2) + c$ when N even
- $T(N) = T(N//2) + d$ when N odd

- Now you solve it...

Let me take the even one to start

- $T(N) = T(N//2) + c$
- $T(N) = T(N//4) + 2c$
- $T(N) = T(N//8) + 3c$ can you see the pattern?
- $T(N) = T(N//2^k) + kc = T(1) + \log N c = b + \log N c$
- Base when $N//2^k = 1$... thus $N = 2^k$ which is $k = \log N$

```
12 def power_squaring(x,n):
13     """
14     Returns x^n
15     via exponential by squaring
16     """
17     if n == 0:
18         return 1
19     elif n == 1:
20         return x
21     # when power is even
22     # N^4 = N^2 * N^2
23     elif n%2 == 0:
24         return power(x*x, n//2)
25     # when the power is odd
26     # N^9 = N^4 & N^4 * N
27     elif n%2 == 1:
28         return power(x*x, n//2) * n
```

- $T(0) = a$
- $T(1) = b$
- $T(N) = T(N//2) + c$ when N even
- $T(N) = T(N//2) + d$ when N odd

- Now you solve it...

Let me take the even one to start

- $T(N) = T(N//2) + c$
- $T(N) = T(N//4) + 2c$
- $T(N) = T(N//8) + 3c$ can you see the pattern?
- $T(N) = T(N//2^k) + kc = T(1) + \log N c = b + \log N c = O(\log N)$
- Base when $N//2^k = 1$... thus $N = 2^k$ which is $k = \log N$

make into 1

$\log_2 N = k$

```
12 def power_squaring(x,n):
13     """
14     Returns x^n
15     via exponential by squaring
16     """
17     if n == 0:
18         return 1
19     elif n == 1:
20         return x
21     # when power is even
22     # N^4 = N^2 * N^2
23     elif n%2 == 0:
24         return power(x*x, n//2)
25     # when the power is odd
26     # N^9 = N^4 & N^4 * N
27     elif n%2 == 1:
28         return power(x*x, n//2) * n
```

Questions?

- Both functions we saw just now are similar
- They are power functions x^n

- Both functions we saw just now are similar
- They are power functions x^n
- But we also saw how their complexity differs
 - $O(N)$
 - $O(\log N)$

- Both functions we saw just now are similar
- They are power functions x^n
- But we also saw how their complexity differs
 - $O(N)$ = normal power
 - $O(\log N)$ = exponential by squaring

Questions?

- So now you know why functions have such complexity?



- So now you know why functions have such complexity?
- Some of the other common ones...

Recurrence and complexity

Recurrence relation:

$$T(N) = T(N/2) + c$$

$$T(1) = b$$

Example algorithm?

Binary search

Solution:

$$O(\log N)$$

Recurrence and complexity

Recurrence relation:

$$T(N) = T(N-1) + c$$

$$T(1) = b$$

Example algorithm?

Linear search

Solution:

$$O(N)$$

Recurrence and complexity

Recurrence relation:

$$T(N) = 2 * T(N/2) + c * N$$

$$T(1) = b$$

Example algorithm?

Merge sort

Solution:

$$O(N \log N)$$

Recurrence and complexity

Recurrence relation:

$$T(N) = T(N-1) + c * N$$

$$T(1) = b$$

Example algorithm?

Selection sort

Solution:

$$O(N^2)$$

Recurrence and complexity

Recurrence relation:

$$T(N) = 2 * T(N-1) + c$$

$$T(0) = b$$

Example algorithm?

Naïve recursive Fibonacci

Solution:

$$O(2^N)$$

- So now you know why functions have such complexity?
- Some of the other common ones...
- Exam would also ask you to proof by induction for complexity...

- So now you know why functions have such complexity?
- Some of the other common ones...

- Exam would also ask you to proof by induction for complexity...
 - We will discuss more in the tutorial
 - Now, let us try to **make my life difficult in the zoom session** later this week...

Questions?

Thank You