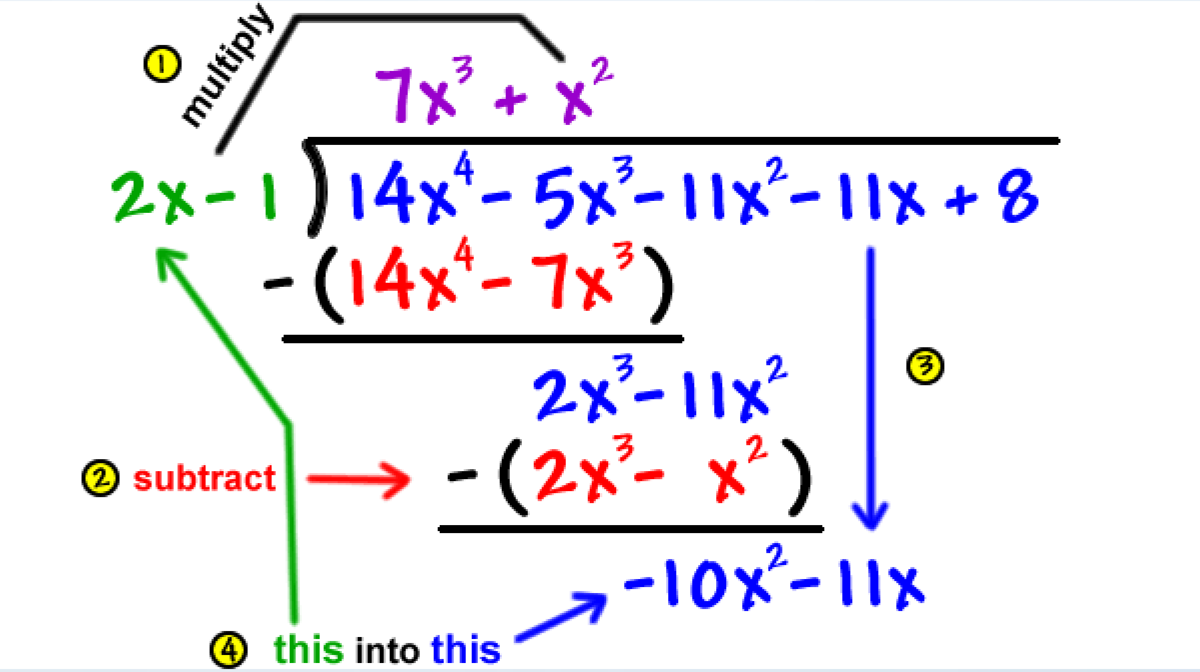
Methodology of CS

Introduction to algorithms

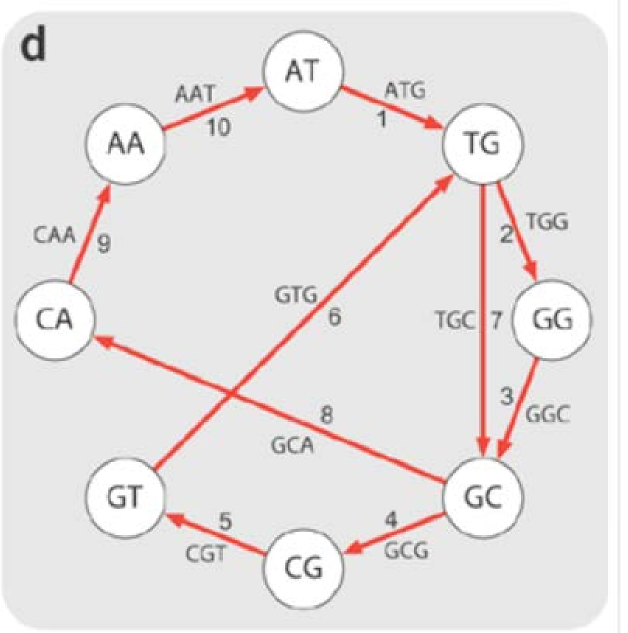
Dustin Xu

What is an algorithm?

How to use addition(+), subtraction(-), multiplication(*), and division(/)?



How to draw one line and visit all edges in a graph?



Eulerian cycle
Visit each edge once
(easier to solve)

What is an algorithm?

- a set of rules that precisely defines a sequence of operations

addition(+), subtraction(-), multiplication(*), and division(/)

Euclid's Algorithm to obtain greatest common divisor (gcd)

Sort Algorithms

Eulerian Path Algorithm

Shortest Path Algorithms (Dijkstra, SPFA, Floyd)

How to design a delivery route for a courier in S.F. Express?

How to analyze big data?

How to play chess with AlphaGo?

How to evaluate an algorithm?

Computational Complexity

Computational Complexity

Rough Estimation

Time Complexity

- Time needed for a sequence of operations

Space Complexity

- Space needed for data structures used in a program

Specific Evaluation

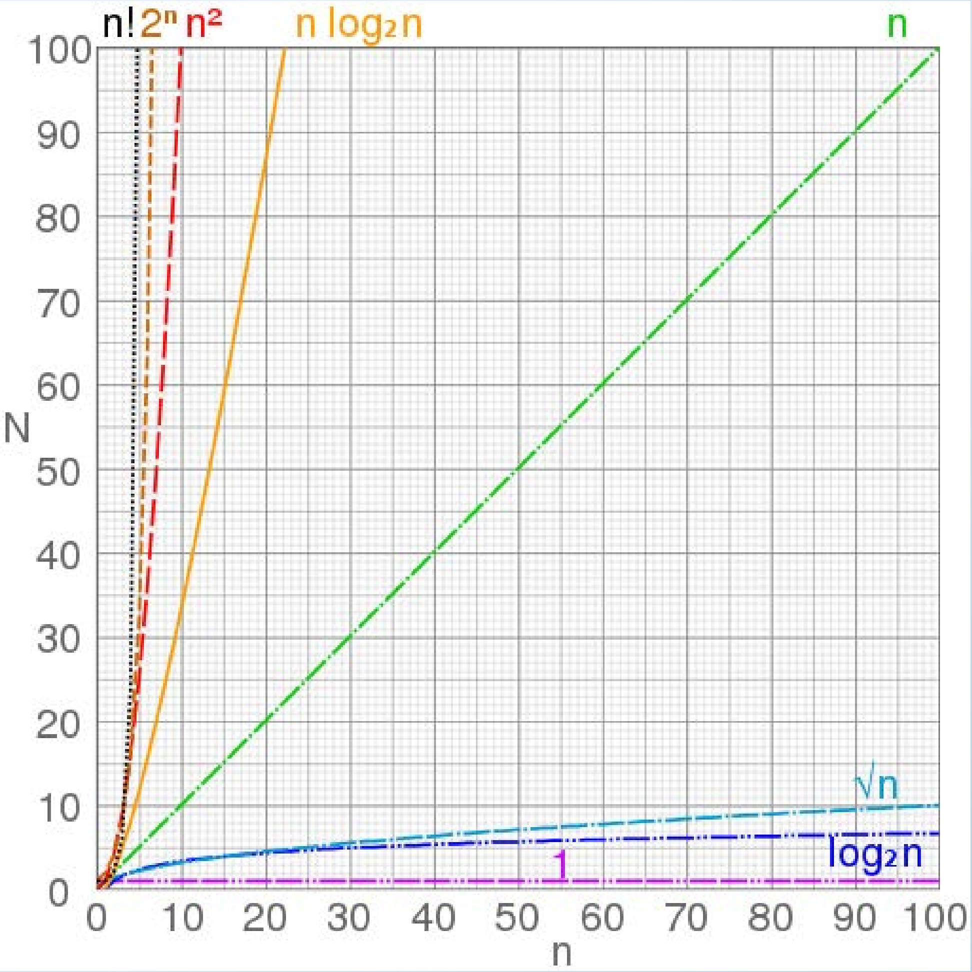
- Time Efficiency
- Running time is always measured to evaluate an algorithm.
- It also depends on CPU and other factors.

- Space Efficiency
- Memory is always measured to evaluate an algorithm.
- It also depends on programming language and other factors.

Computational Complexity

Big O Notation

- $O(1)$ Constant Complexity
- $O(log_2 n)$ Logarithmic Complexity
- $O(n)$ Linear Complexity
- $O(n\ log_2 n)$ Linearithmic Complexity
- $O(n\hat{}2)$ $O(n\hat{}3)$ ... Polynomial Complexity
- $O(a\hat{}n)$ Exponential Complexity
- $O(n!)$ Factorial Complexity



Big O Notation evaluates the magnitude of complexity of an algorithm based on input size n

Core Ideas

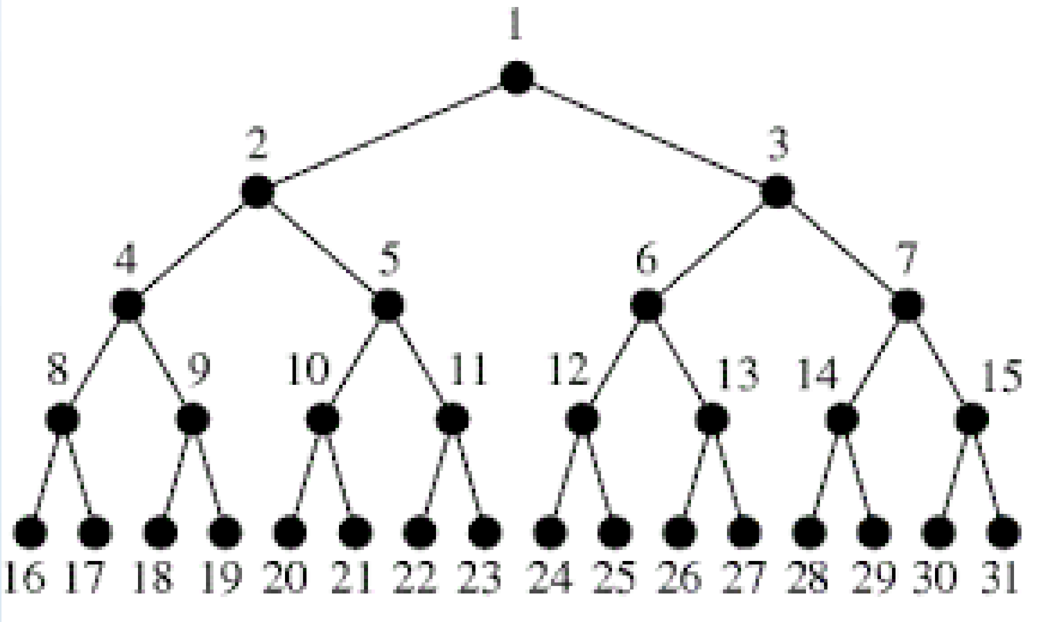
- Coefficients $O(kn)$ ✘
- Constant $O(n + C)$ ✘

Some Examples

Look for a number in this perfect binary tree with n nodes from the root.

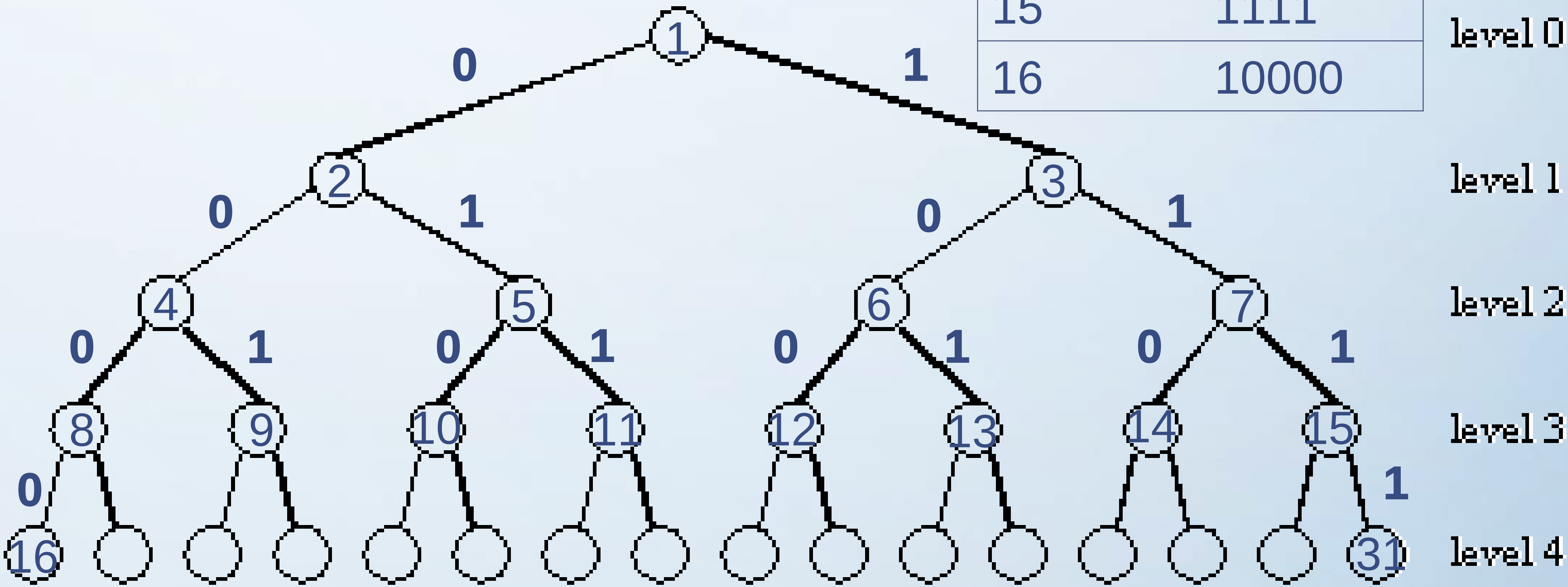
```
count = 0
for i in range(n):
    for j in range(i,n):
        count += 1
```

Time: $O(n^2)$
Space: $O(1)$



A Perfect Binary Tree

Decimal	Binary
5	101
14	1110
15	1111
16	10000



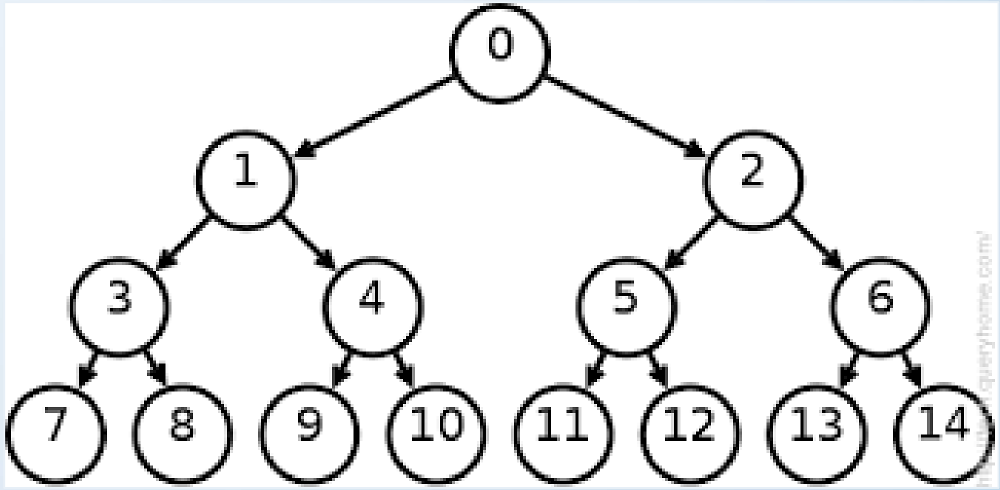
The depth of a perfect binary tree with n nodes is $int(\log_2 n)$

Some Examples

```
count = 0
for i in range(n):
    for j in range(i,n):
        count += 1
```

Time: $O(n^2)$
Space: $O(1)$

Look for a number in this perfect binary tree with n nodes from the root.



Time: $O(\log n)$
Space: $O(n \log n)$

A Task For You

Fibonacci Numbers

Algorithm A

```
f = [1,1]
for i in range(2,n):
    f.append(f[i-1]+f[i-2])
print(f[n-1])
```

Time: $O(n)$
Space: $O(n)$

```
f = [1,1]
for i in range(2,n):
    f[i%2] += f[(i+1)%2]
print(f[(n-1)%2])
```

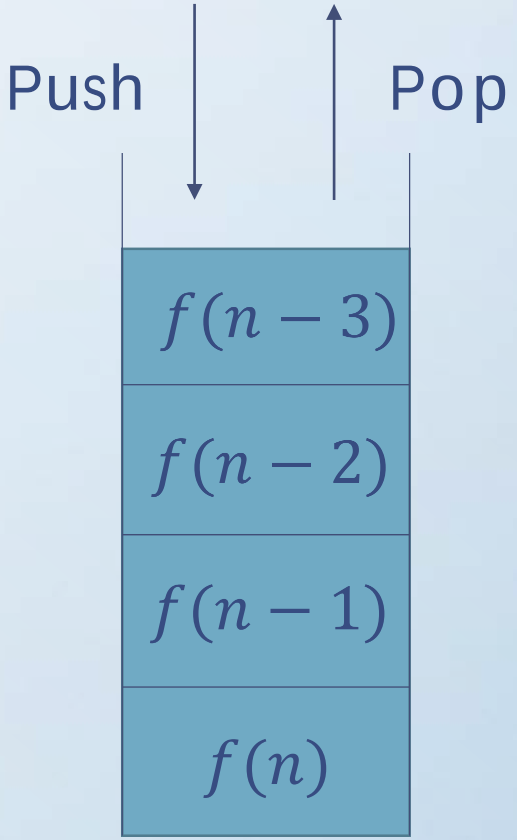
Time: $O(n)$
Space: $O(1)$

Algorithm B

An exponential time algorithm

Time: $O(f(n))$
Space: $O(f(n))$

```python
def f(n):
    if n <= 2:
        return 1
    return f(n-1) + f(n-2)
print(f(n))
```



Stack Overflow!

Algorithm C

```
sqrt5 = 5**0.5
print(1/sqrt5*(((1+sqrt5)/2)**n-((1-sqrt5)/2)**n))
```

Time: $O(1)$
Space: $O(1)$

$$f_{\mathrm{n}} = \frac{1}{\sqrt{5}} \cdot [\left(\frac{1+\sqrt{5}}{2}\right)^{n} - \left(\frac{1-\sqrt{5}}{2}\right)^{n}]$$

```
In [3]: n = 100
```

```
In [4]: f = [1,1]
        for i in range(2,n):
            f.append(f[i-1]+f[i-2])
        print(f[n-1])

        354224848179261915075
```

```
In [5]: f = [1,1]
        for i in range(2,n):
            f[i%2] += f[(i+1)%2]
        print(f[(n-1)%2])

        354224848179261915075
```

```
In [6]: sqrt5 = 5**0.5
        print(int(1/sqrt5*(((1+sqrt5)/2)**n-((1-sqrt5)/2)**n)))

        354224848179263111168
```

The Best Algorithm?

```
f = [1,1]
for i in range(2,n):
    f[i%2] += f[(i+1)%2]
print(f[(n-1)%2])
```

Time: $O(n)$
Space: $O(1)$

A Better Algorithm

Fast Matrix Exponentiation

Matrix Multiplication &
Divide and Conquer

Time: $O(\log n)$
Space: $O(1)$

Thank you!