Memoization

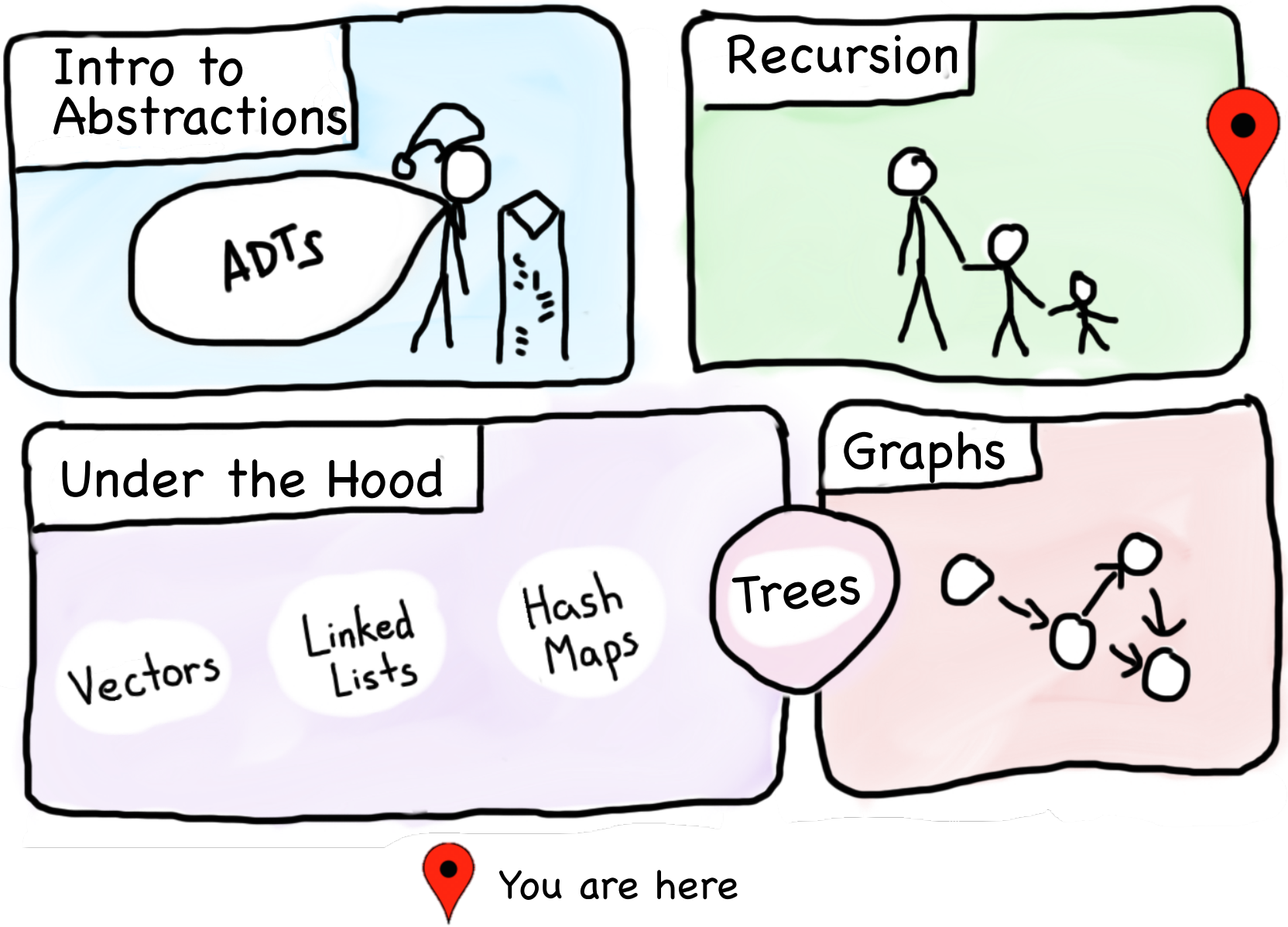
Chris Piech

CS 106B
Lecture 10
Jan 29, 2015

> Tell me and I forget. Teach me and I rememoize.*
>
> - Xun Kuang, 300 BCE

* This is almost the correct quote

Course Syllabus



Socrative.com

Room: **106BWIN16**

Midterm

Date: Tuesday, Feb 9th
Time: 7-9pm
Location: Hewlett 200, Hewlett 201
+ Braun Aud.

More details on Wednesday

Milestone 5

Write a recursive function that completes a grammar.

Sentence is an easy to understand test case.

Mystery trace is a fun test case.

Milestone 5

```
QUESTION: {{lots of text that I excluded for space}}
-----
RECURSION:CALL|CALL OP TERM
-----
CALL:mystery(level - 1, XEXP)
-----
COMBO:REXP OP XEXP
-----
XEXP:x|x OP XEXP|x OP NUM
-----
REXP:r|r OP REXP|r OP NUM
-----
NUM:1|5
-----
OP:+|-|*|%|/
-----
```

Milestone 5

QUESTION

Milestone 5

```
int mystery(int level, int x) {
  if(level == 0) {
    return NUM;
  } else {
    int r = CALL;
    return COMBO;
  }
}
What is the result of mystery(3, 5);
```

Milestone 5

```
“int mystery(int level, int x) {
  if(level == 0) {
    return NUM;
  } else {
    int r = CALL;
    return COMBO;
  }
}
What is the result of mystery(3, 5);”
```

Milestone 5

```
int mystery(int level, int x) {
  if(level == 0) {
    return NUM;
  } else {
    int r = CALL;
    return COMBO;
  }
}
What is the result of mystery(3, 5);
```

Milestone 5

```
int mystery(int level, int x) {
  if(level == 0) {
    return 5;
  } else {
    int r = CALL;
    return COMBO;
  }
}
```

What is the result of mystery(3, 5);

Milestone 5

```
int mystery(int level, int x) {
  if(level == 0) {
    return 5;
  } else {
    int r = mystery(level-1, XEXP);
    return COMBO;
  }
}
```

What is the result of mystery(3, 5);

Milestone 5

```
int mystery(int level, int x) {
  if(level == 0) {
    return 5;
  } else {
    int r = mystery(level-1, x OP XEXP);
    return COMBO;
  }
}
```

What is the result of mystery(3, 5);

Milestone 5

```
int mystery(int level, int x) {
  if(level == 0) {
    return 5;
  } else {
    int r = mystery(level-1, x + XEXP);
    return COMBO;
  }
}
```

What is the result of mystery(3, 5);

Milestone 5

```
int mystery(int level, int x) {
  if(level == 0) {
    return 5;
  } else {
    int r = mystery(level-1, x + x);
    return COMBO;
  }
}
```

What is the result of mystery(3, 5);

Milestone 5

```
int mystery(int level, int x) {
  if(level == 0) {
    return 5;
  } else {
    int r = mystery(level-1, x + x);
    return REXP OP XEXP;
  }
}
What is the result of mystery(3, 5);
```

Milestone 5

```
int mystery(int level, int x) {
  if(level == 0) {
    return 5;
  } else {
    int r = mystery(level-1, x + x);
    return r OP XEXP;
  }
}
What is the result of mystery(3, 5);
```

Milestone 5

```
int mystery(int level, int x) {
  if(level == 0) {
    return 5;
  } else {
    int r = mystery(level-1, x + x);
    return r * XEXP;
  }
}
What is the result of mystery(3, 5);
```

Milestone 5

```
int mystery(int level, int x) {
  if(level == 0) {
    return 5;
  } else {
    int r = mystery(level-1, x + x);
    return r * x OP NUM;
  }
}
```

What is the result of mystery(3, 5);

Milestone 5

```
int mystery(int level, int x) {
  if(level == 0) {
    return 5;
  } else {
    int r = mystery(level-1, x + x);
    return r * x - NUM;
  }
}
```

What is the result of mystery(3, 5);

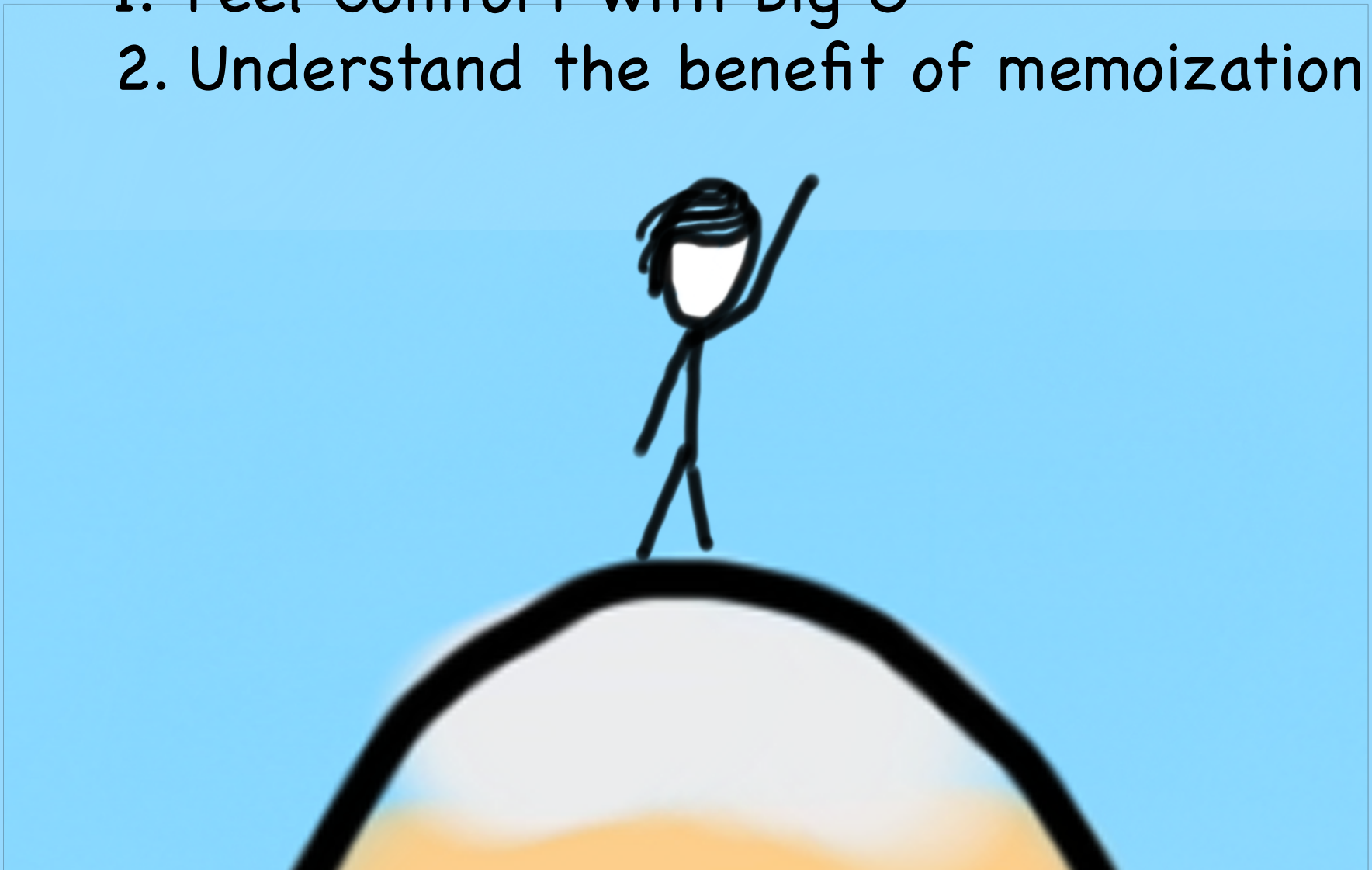
Milestone 5

```
int mystery(int level, int x) {
  if(level == 0) {
    return 5;
  } else {
    int r = mystery(level-1, x + x);
    return r * x - 1;
  }
}
What is the result of mystery(3, 5);
```

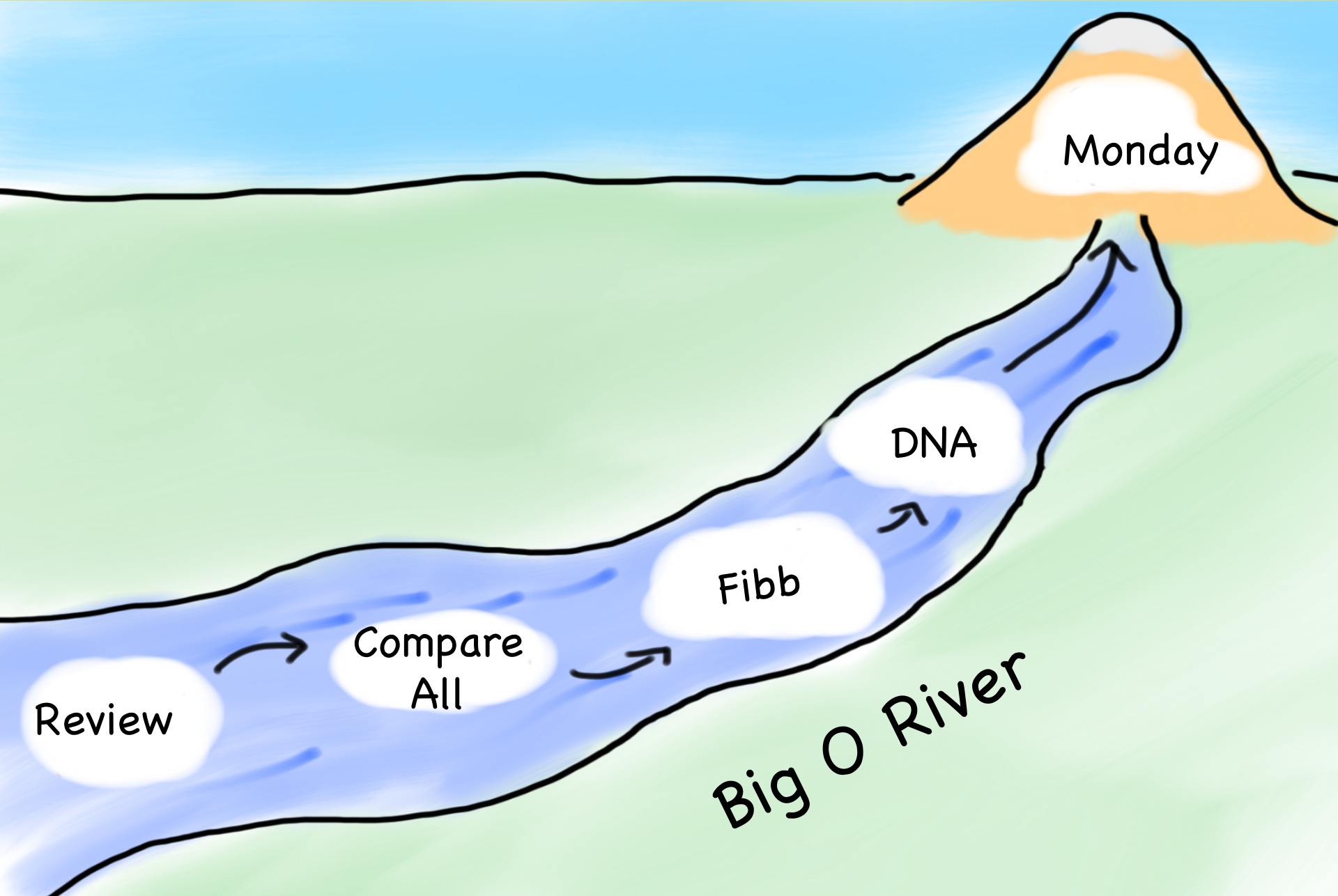
Jazzy

Today's Goals

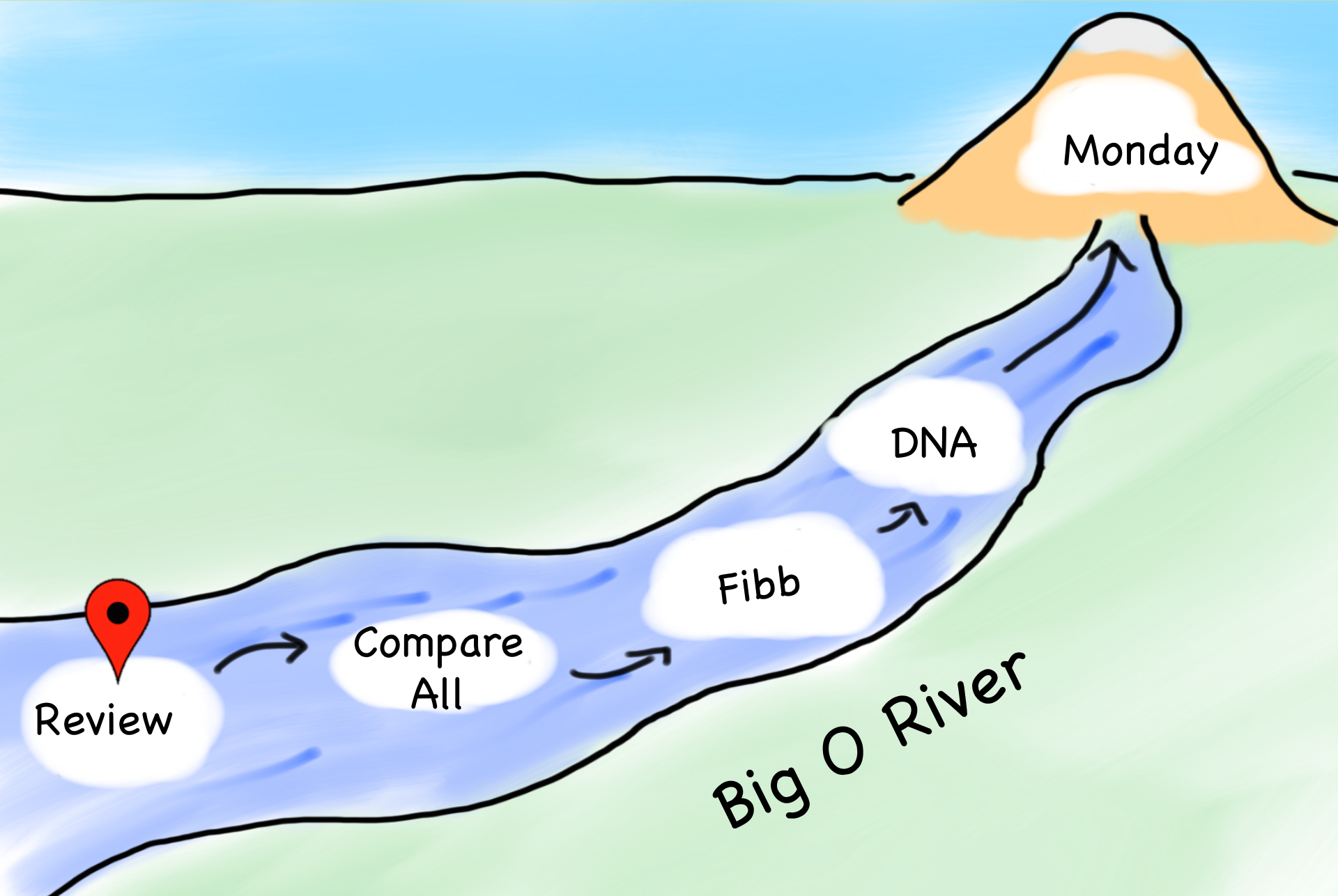
1. Feel Comfort with Big O
2. Understand the benefit of memoization



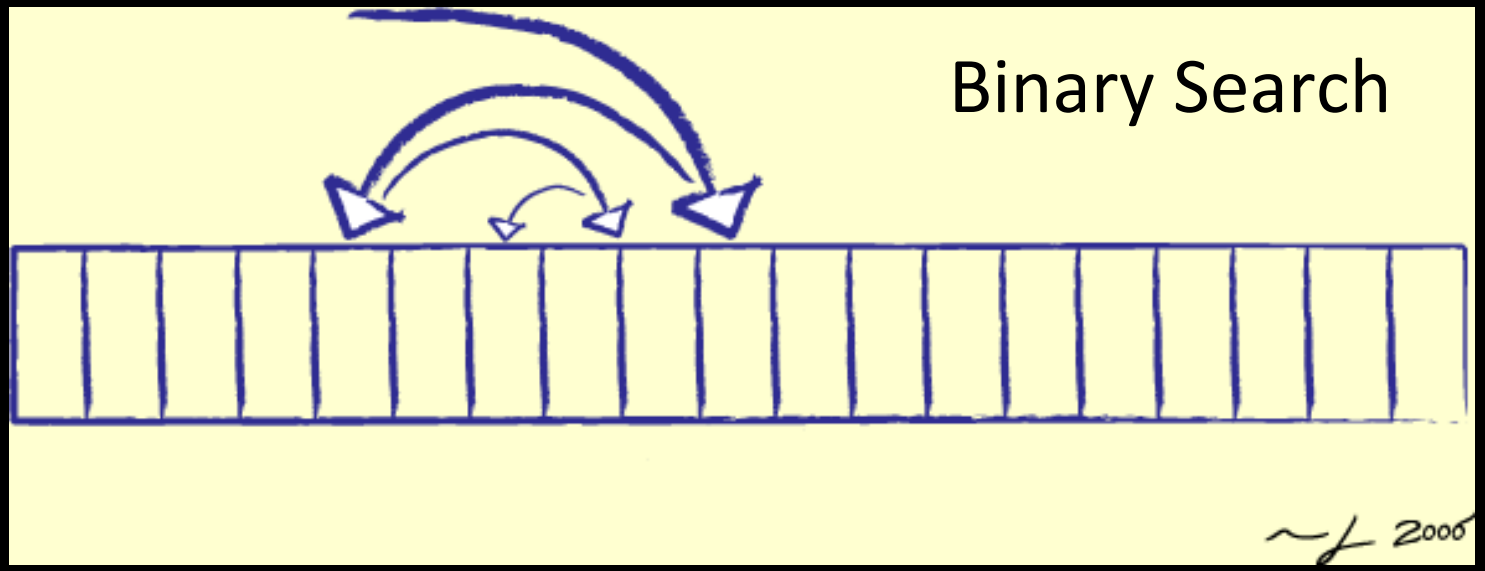
Today's Goals



Today's Goals



How Can We Compare Algorithms?



Count Number of Executions

```
int find(Vector<int> & vec, int goal) {
  for(int i = 0; i < vec.size(); i++) {
    if(vec[i] == goal) return i;
  }
  return -1;
}
```

$2n + 1$

$2n$

1

$$T(n) = 4n + 2$$

Do we really care about the 4?

Do we really care about the +2?

Big O Notation

- Ignore *everything* except the dominant growth term, including constant factors.
- Examples:
  - $4n + 2 = \mathbf{O(n)}$
  - $137n + 271 = \mathbf{O(n)}$
  - $n^2 + 3n + 4 = \mathbf{O(n^2)}$
  - $2^n + n^3 = \mathbf{O(2^n)}$
  - $\log_2 n = \mathbf{O(\log n)}$

Standard Complexity

The complexity of a particular algorithm tends to fall into one of a small number of standard complexity classes:

constant	$O(1)$	Finding first element in a vector
logarithmic	$O(\log N)$	Binary search in a sorted vector
linear	$O(N)$	Summing a vector; linear search
$N \log N$	$O(N \log N)$	Merge sort
quadratic	$O(N^2)$	Selection sort
cubic	$O(N^3)$	Obvious algorithms for matrix multiplication
exponential	$O(2^N)$	Tower of Hanoi solution

$\log n$	n	$n \log n$	n^2	2^n
2	**4**	8	16	16
3	**8**	24	64	256
4	**16**	64	256	65,536
5	**32**	160	1,024	4,294,967,296
6	**64**	384	4,096	1.84×10^{19}
7	**128**	896	16,384	3.40×10^{38}
8	**256**	2,048	65,536	1.16×10^{77}
9	**512**	4,608	262,144	1.34×10^{154}
10	**1,024**	10,240 (.000003s)	1,048,576 (.0003s)	1.80×10^{308}
30	**1,300,000,000**	39000000000 (13s)	1690000000000000000 (18 years)	LOL

# of Facebook accounts


THIS IS ME NOT CARING
ABOUT PERFORMANCE TUNING UNLESS IT CHANGES BIG-O
memegenerator.net

Types of Analysis

Worst-Case Analysis

- What's the *worst* possible runtime for the algorithm?
- Useful for "sleeping well at night."
- What we use in CS106B

Best-Case Analysis

- Said nobody ever.

Average-Case Analysis

- What's the *average* runtime for the algorithm?
- Far beyond the scope of this class; take CS109, CS161!

Types of Analysis

Worst-Case Analysis

- What's the *worst* possible runtime for the algorithm?
- Useful for "sleeping well at night."
- What we use in CS106B

Best-Case Analysis

- Said nobody ever.

Average-Case Analysis

- What's the *average* runtime for the algorithm?
- Far beyond the scope of this class; take CS109, CS161!

Types of Analysis

Worst-Case Analysis

- What's the *worst* possible runtime for the algorithm?
- Useful for "sleeping well at night."
- What we use in CS106B

Best-Case Analysis

- Said nobody ever.

Average-Case Analysis

- What's the *average* runtime for the algorithm?
- Far beyond the scope of this class; take CS109, CS161!

Types of Analysis

Worst-Case Analysis

- What's the *worst* possible runtime for the algorithm?
- Useful for "sleeping well at night."
- What we use in CS106B

Best-Case Analysis

- Said nobody ever.

Average-Case Analysis

- What's the *average* runtime for the algorithm?
- Far beyond the scope of this class; take CS109, CS161!

Example

What is the big O of this function?

```
void friendly(Set<string>& facebookUsers) {

  int n = facebookUsers.size();
  cout << "n: " << n << endl;

  for(string user : facebookUsers){
    notifyIfBirthday(user);
    changeName(user, "John Cena");
    addFriend(user, "Mehran Sahami");
  }
}
```

Example

What is the big O of this function?

```
void friendly(Set<string>& facebookUsers) {
  int n = facebookUsers.size();
  cout << "n: " << n << endl;

  for(string user : facebookUsers){
    notifyIfBirthday(user);
    changeName(user, "John Cena");
    addFriend(user, "Mehran Sahami");
  }
}
```

Example

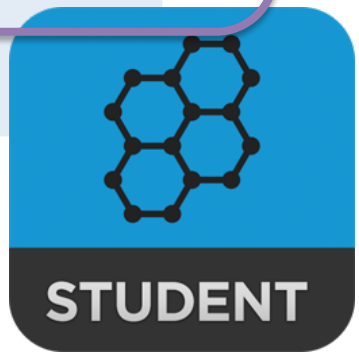
What is the big O of this function?

```
void friendly(Set<string>& facebookUsers) {
  int n = facebookUsers.size();
  cout << "n: " << n << endl;

  for(string user : facebookUsers){
    notifyIfBirthday(user);
    changeName(user, "John Cena");
    addFriend(user, "Mehran Sahami");
  }
}
```

Example

What is the big O of this function?

```
void friendly(Set<string>& facebookUsers) {

  int n = facebookUsers.size();
  cout << "n: " << n << endl;

  for(string user : facebookUsers){
    notifyIfBirthday(user);
    changeName(user, "John Cena");
    addFriend(user, "Mehran Sahami");
  }
}
```



Example

What is the big O of this function?

```
void friendly(Set<string>& facebookUsers) {

  int n = facebookUsers.size();
  cout << "n: " << n << endl;

  for(string user : facebookUsers){
    notifyIfBirthday(user);
    changeName(user, "John Cena");
    addFriend(user, "Mehran Sahami");
  }
}
```

Example

What is the big O of this function?

```
void friendly(Set<string>& facebookUsers) {

  int n = facebookUsers.size();
  cout << "n: " << n << endl;

  for(string user : facebookUsers){
    notifyIfBirthday(user);
    changeName(user, "John Cena");
    addFriend(user, "Mehran Sahami");
  }
}
```

Example

What is the big O of this function?

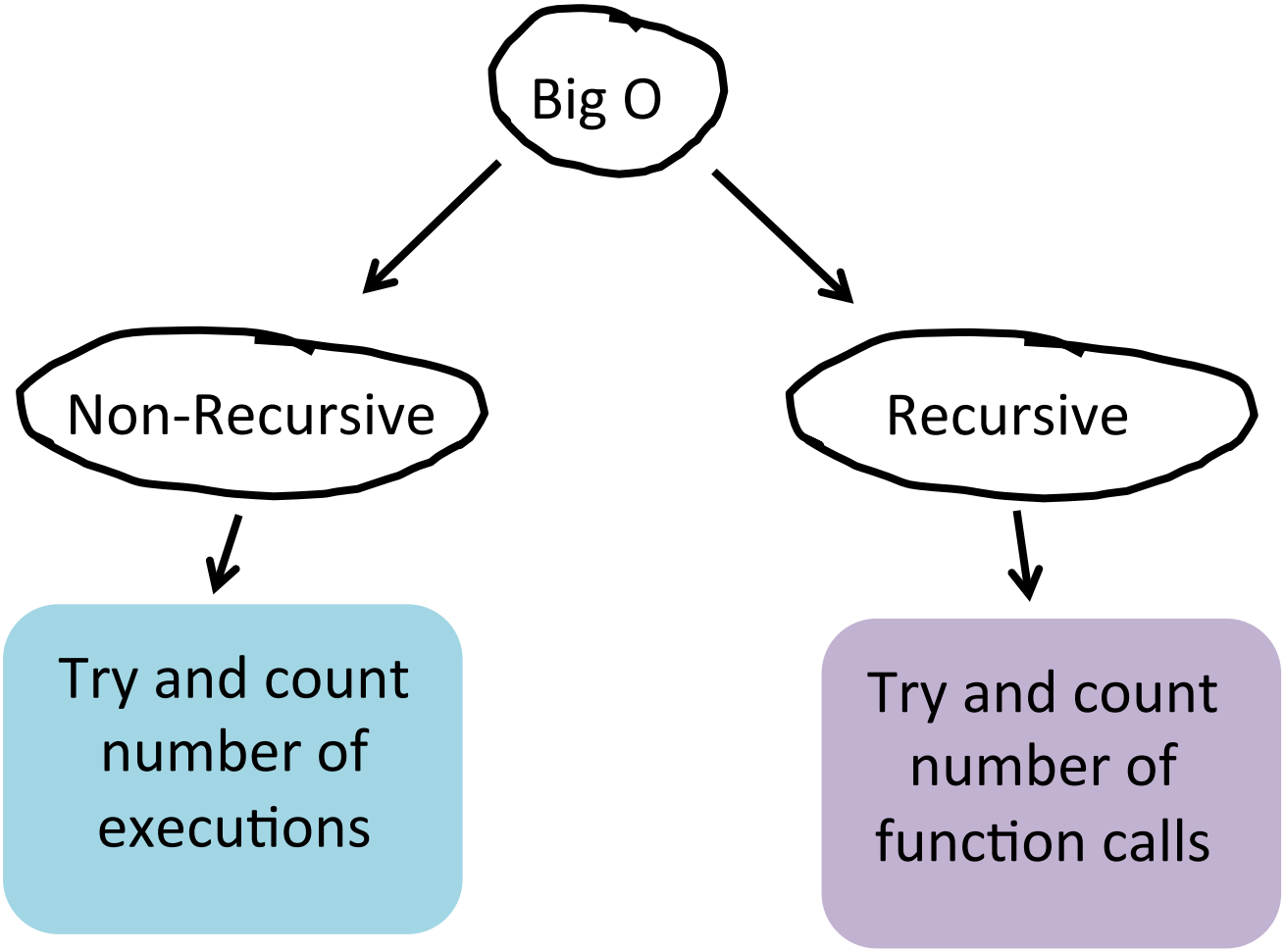
```
void friendly(Set<string>& facebookUsers) {

  int n = facebookUsers.size();
  cout << "n: " << n << endl;

  for(string user : facebookUsers){
    notifyIfBirthday(user);
    changeName(user, "John Cena");
    addFriend(user, "Mehran Sahami");
  }
}
```

Example

What is the big O of this function?

```
void friendly(Set<string>& facebookUsers) {

  int n = facebookUsers.size();
  cout << "n: " << n << endl;

  for(string user : facebookUsers){
    notifyIfBirthday(user);
    changeName(user, "John Cena");
    addFriend(user, "Mehran Sahami");
  }
}
```

Strategy

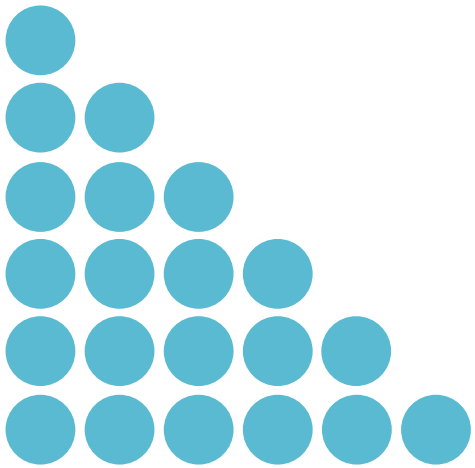

Big O
Non-Recursive
Recursive
Try and count number of executions
Try and count number of function calls

Gauss Summation

Arithmetic Sum

- You can convince yourself that

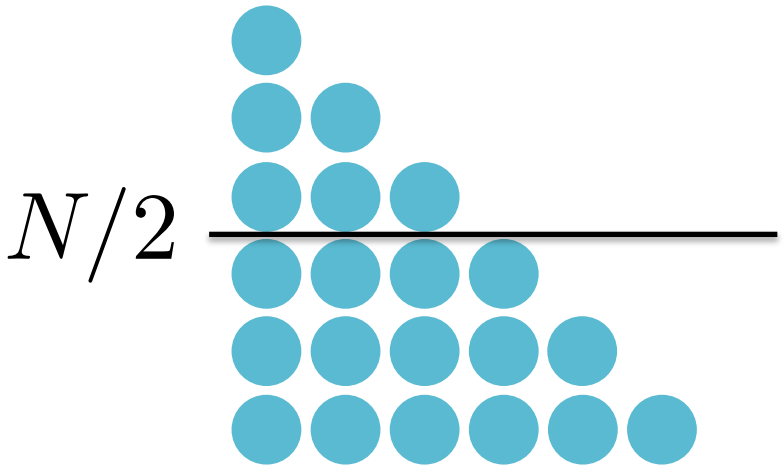
$$1 + 2 + 3 + \cdots + (N-2) + (N-1) + N = \frac{N \times (N+1)}{2}$$



Arithmetic Sum

- You can convince yourself that

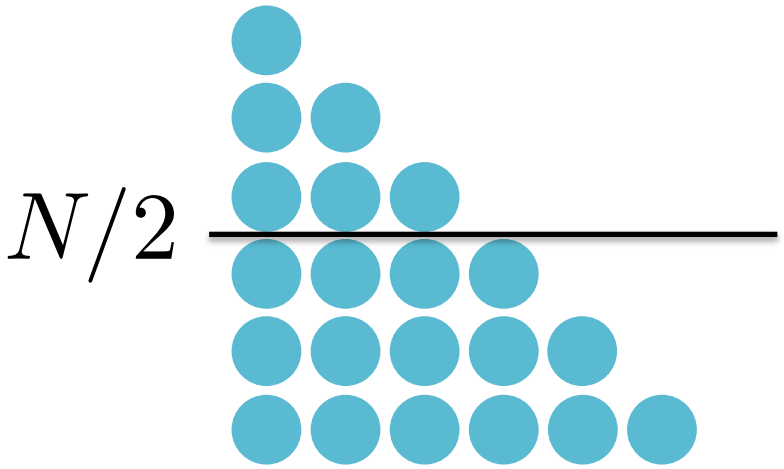
$$1 + 2 + 3 + \cdots + (N-2) + (N-1) + N = \frac{N \times (N+1)}{2}$$



Arithmetic Sum

- You can convince yourself that

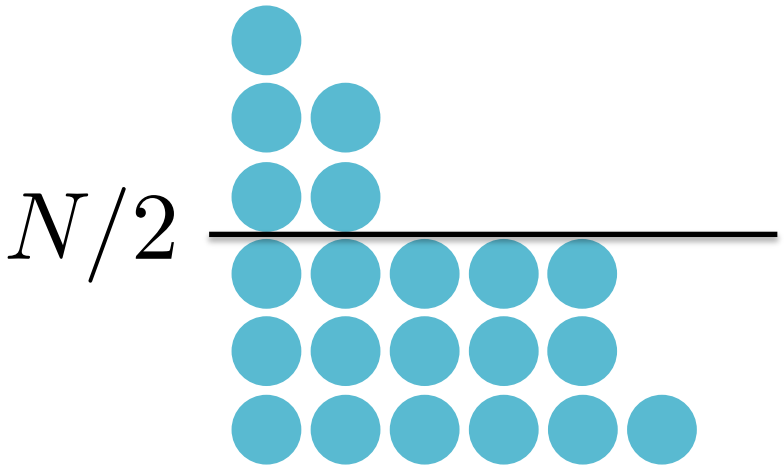
$$1 + 2 + 3 + \cdots + (N-2) + (N-1) + N = \frac{N \times (N+1)}{2}$$



Arithmetic Sum

- You can convince yourself that

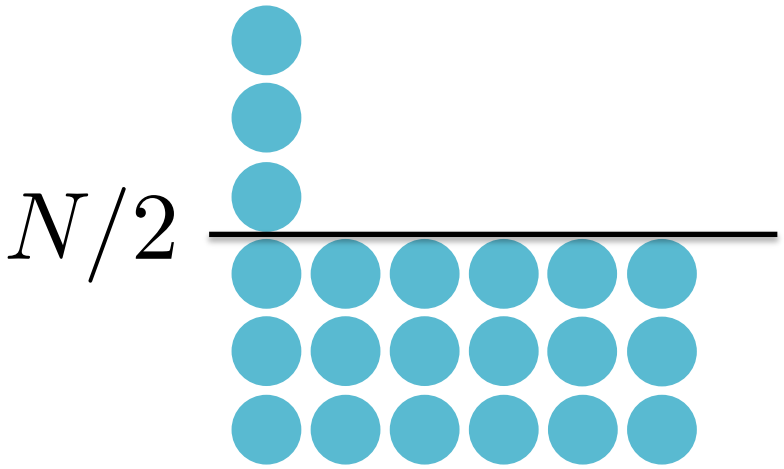
$$1 + 2 + 3 + \cdots + (N-2) + (N-1) + N = \frac{N \times (N+1)}{2}$$



Arithmetic Sum

- You can convince yourself that

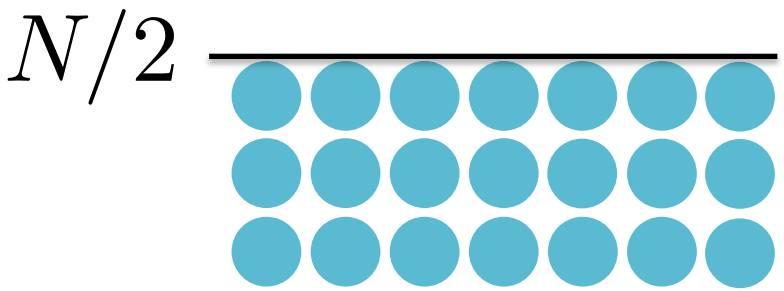
$$1 + 2 + 3 + \cdots + (N-2) + (N-1) + N = \frac{N \times (N+1)}{2}$$



Arithmetic Sum

- You can convince yourself that

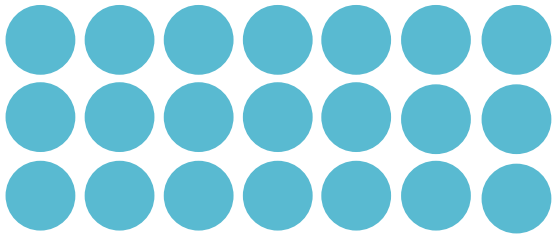
$$1 + 2 + 3 + \cdots + (N-2) + (N-1) + N = \frac{N \times (N+1)}{2}$$



Arithmetic Sum

- You can convince yourself that

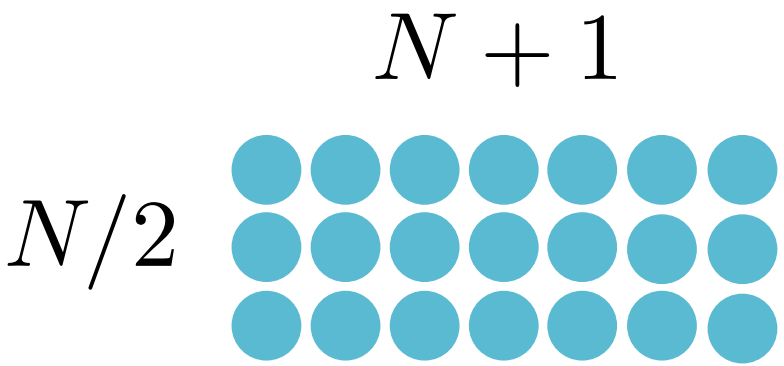
$$1 + 2 + 3 + \cdots + (N-2) + (N-1) + N = \frac{N \times (N+1)}{2}$$



Arithmetic Sum

- You can convince yourself that

$$1 + 2 + 3 + \cdots + (N-2) + (N-1) + N = \frac{N \times (N+1)}{2}$$

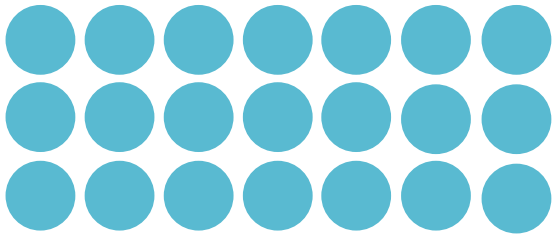


Arithmetic Sum

- You can convince yourself that

$$1 + 2 + 3 + \cdots + (N-2) + (N-1) + N = \frac{N \times (N+1)}{2}$$

$N+1$

$N/2$ 

$$= \frac{N \cdot (N+1)}{2}$$

Arithmetic Sum

- You can convince yourself that

$$1 + 2 + 3 + \cdots + (N-2) + (N-1) + N = \frac{N \times (N+1)}{2}$$

$$= \frac{N \cdot (N+1)}{2}$$

Arithmetic Sum

- You can convince yourself that

$$1 + 2 + 3 + \cdots + (N-2) + (N-1) + N = \frac{N \times (N+1)}{2}$$

$$= \frac{N \cdot (N+1)}{2}$$

$$= \frac{1}{2}N^2 + \frac{1}{2}N$$

Arithmetic Sum

- You can convince yourself that

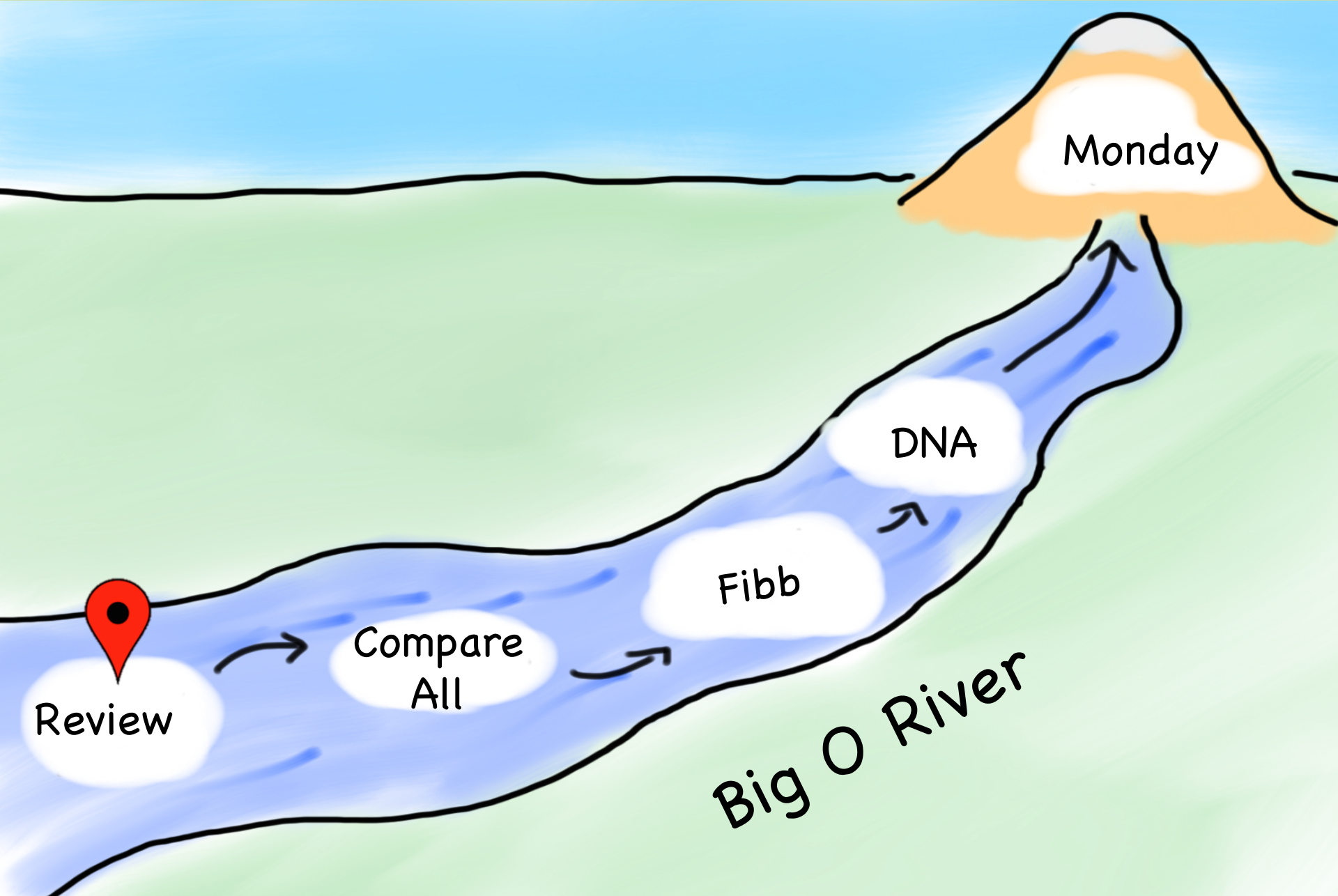
$$1 + 2 + 3 + \cdots + (N-2) + (N-1) + N = \frac{N \times (N+1)}{2}$$

$$= \frac{N \cdot (N+1)}{2}$$

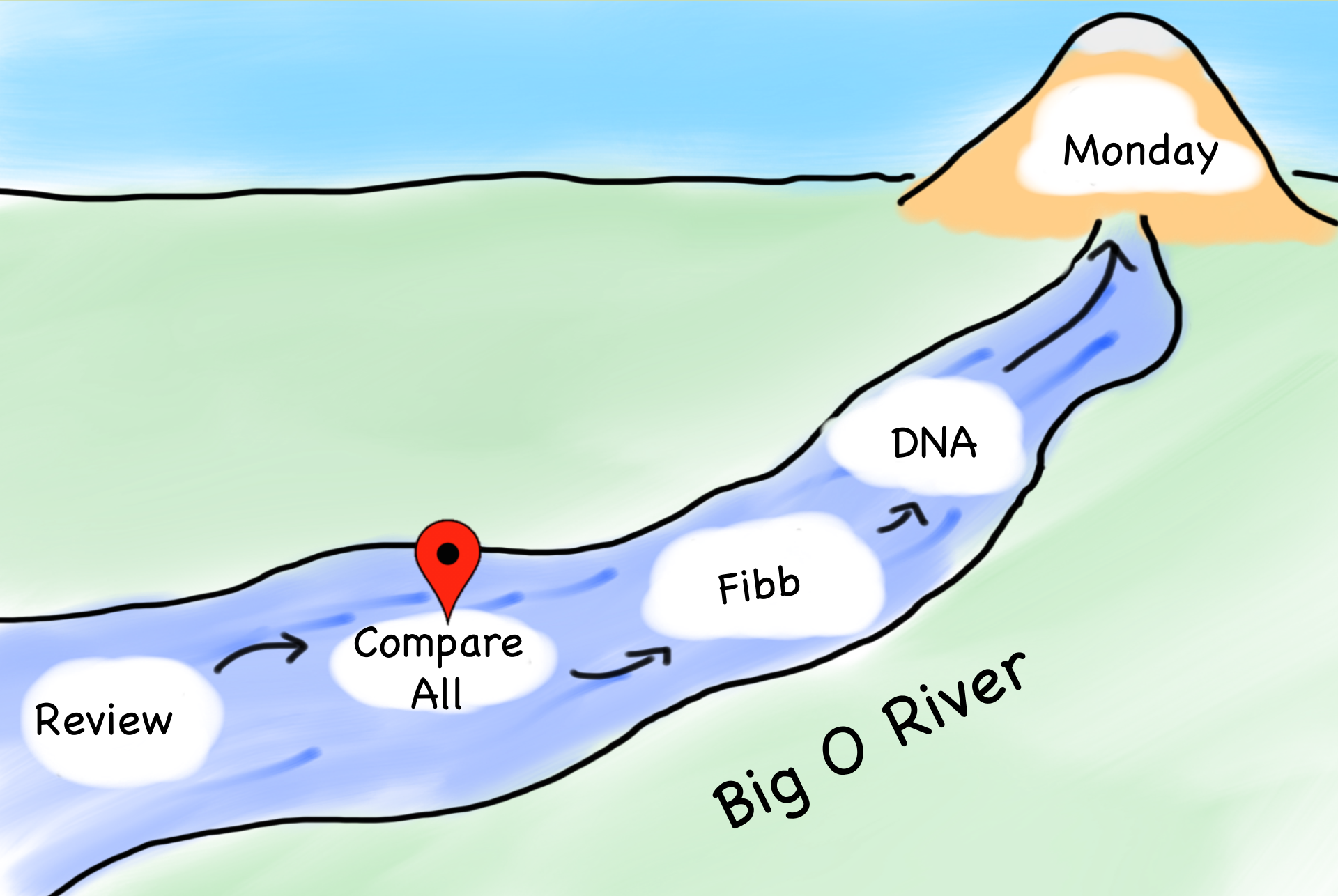
$$= \frac{1}{2}N^2 + \frac{1}{2}N$$

$$\mathcal{O}(N^2)$$

Today's Goals



Today's Goals



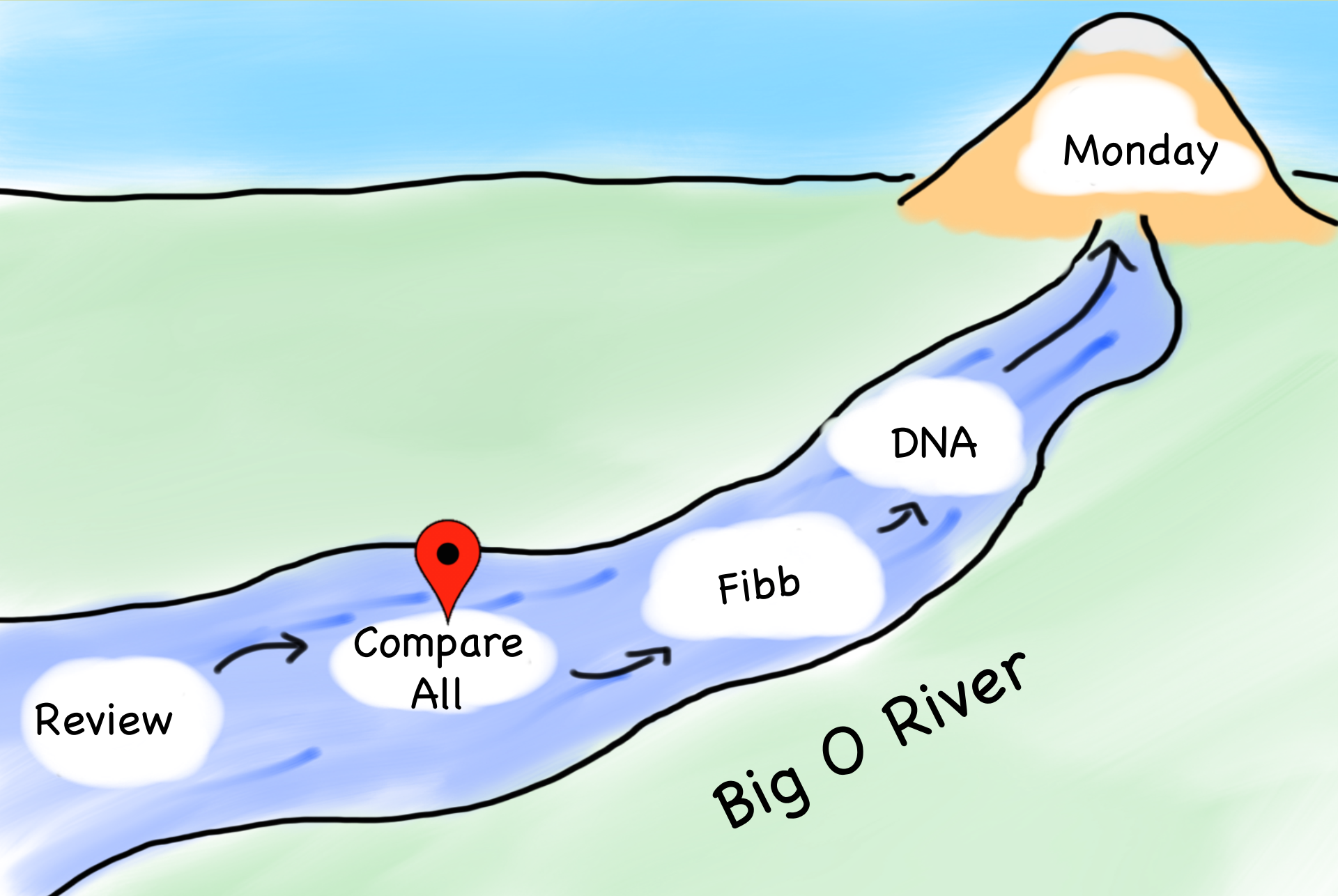
Complexity Compare All

What is the complexity of this function?
Let n be data.size()

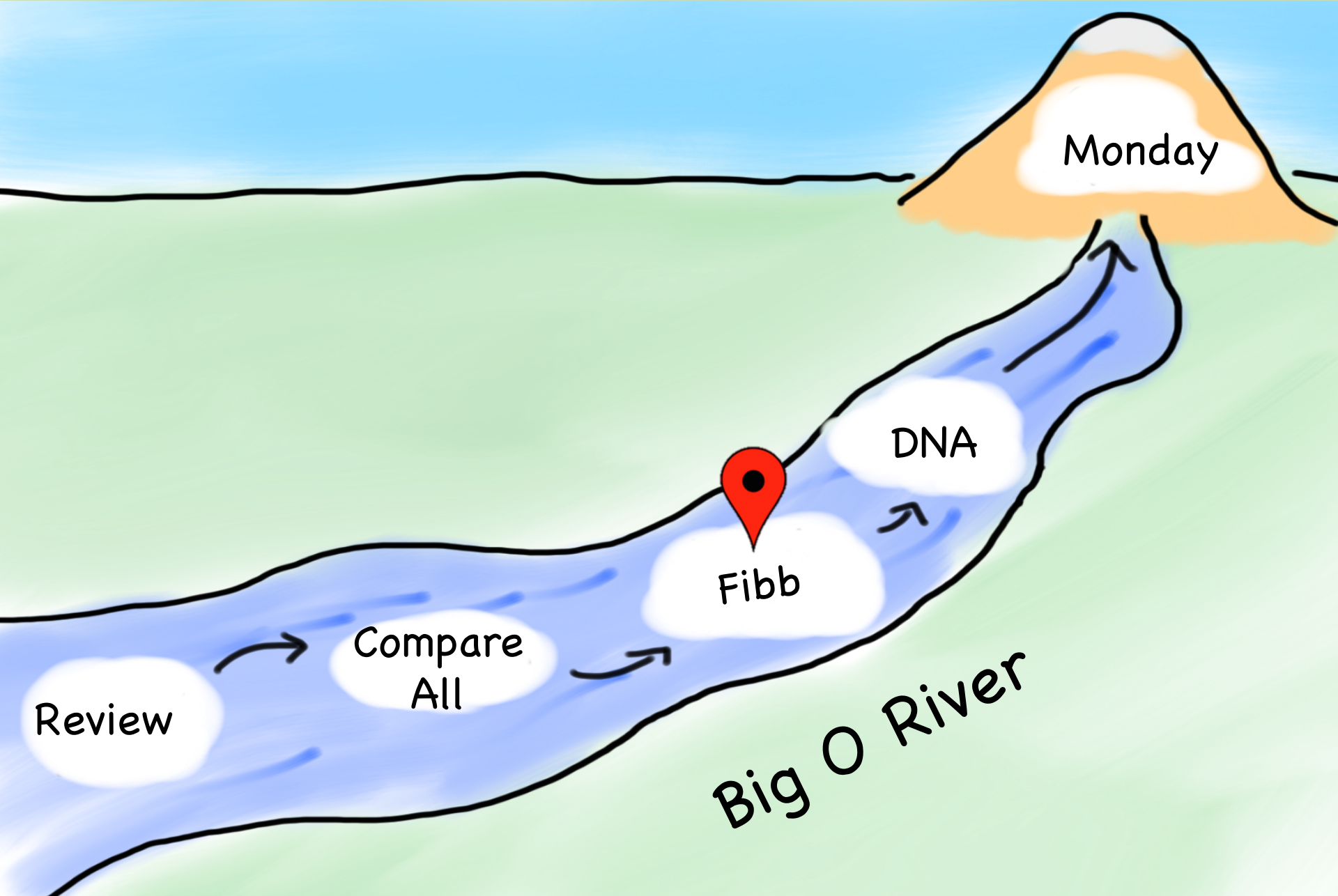
```
Grid<double> compareAll(Vector<double>& data) {
  int n = data.size();
  Grid<double> similarity(n, n);
  for(int i = 0; i < n; i++) {
    for(int j = 0; j < n; j++) {
      double diff = calculateDiff(data[i], data[j]);
      similarity[i][j] = diff;
    }
  }
  return similarity;
}
```

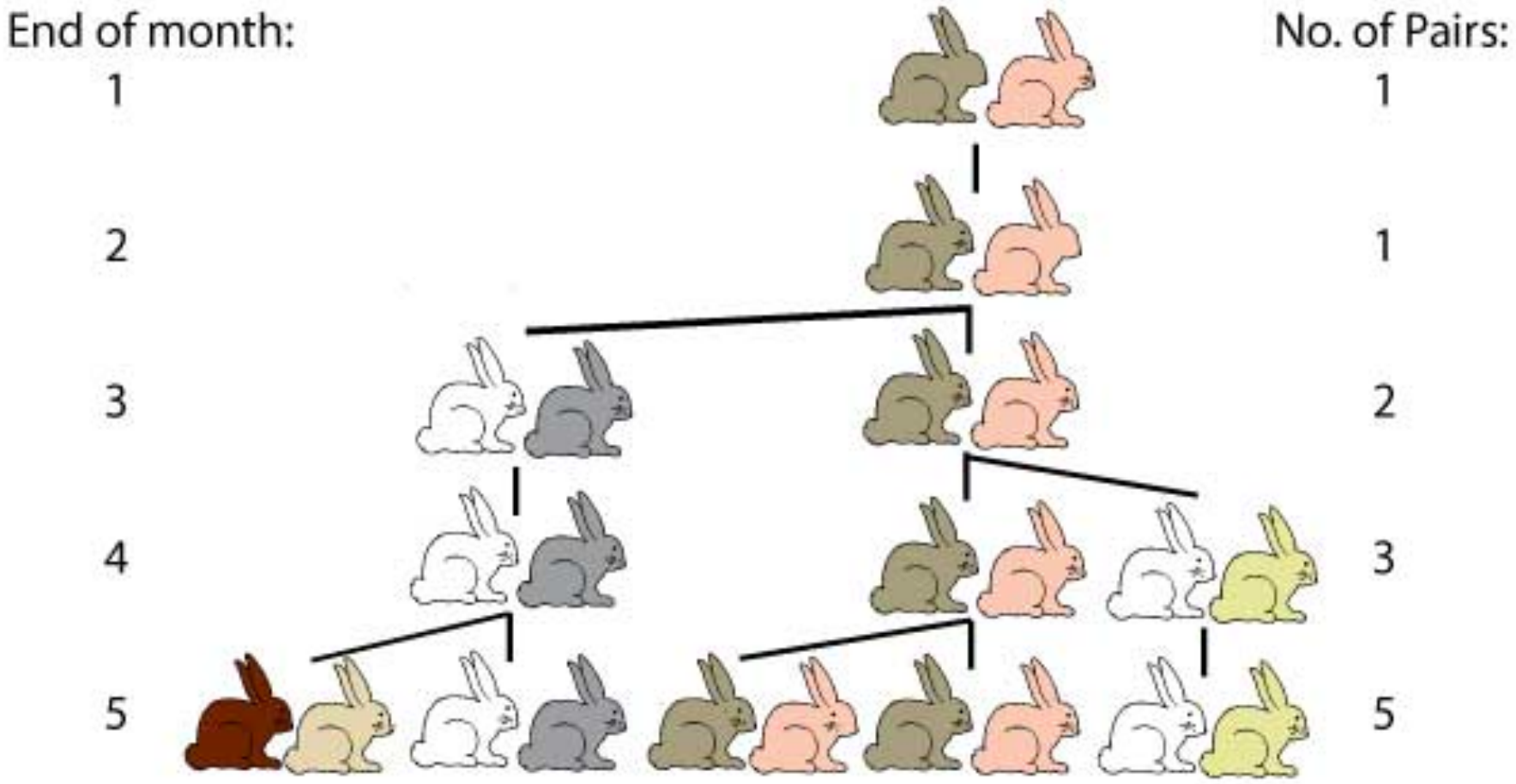

Qt

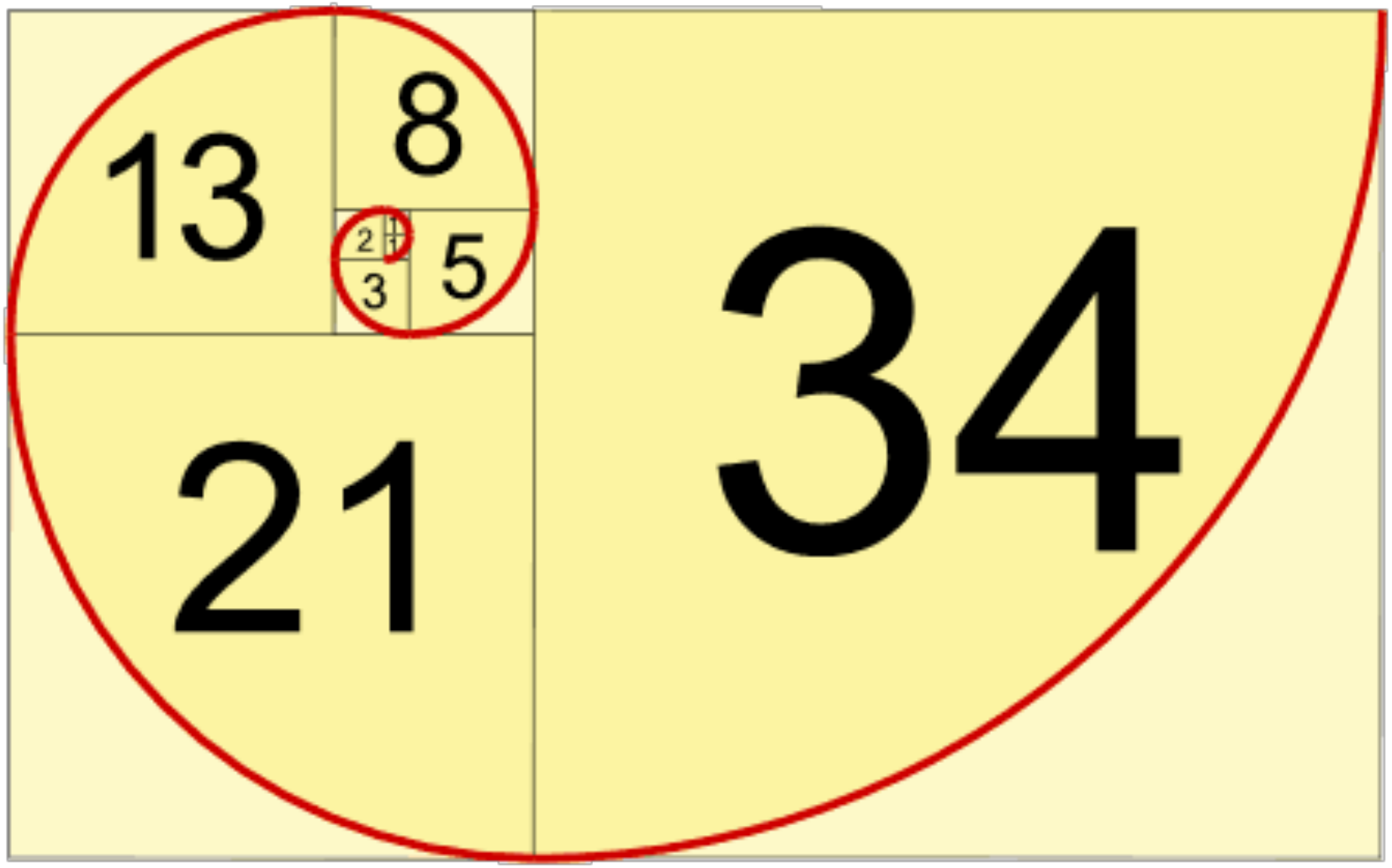
Today's Goals

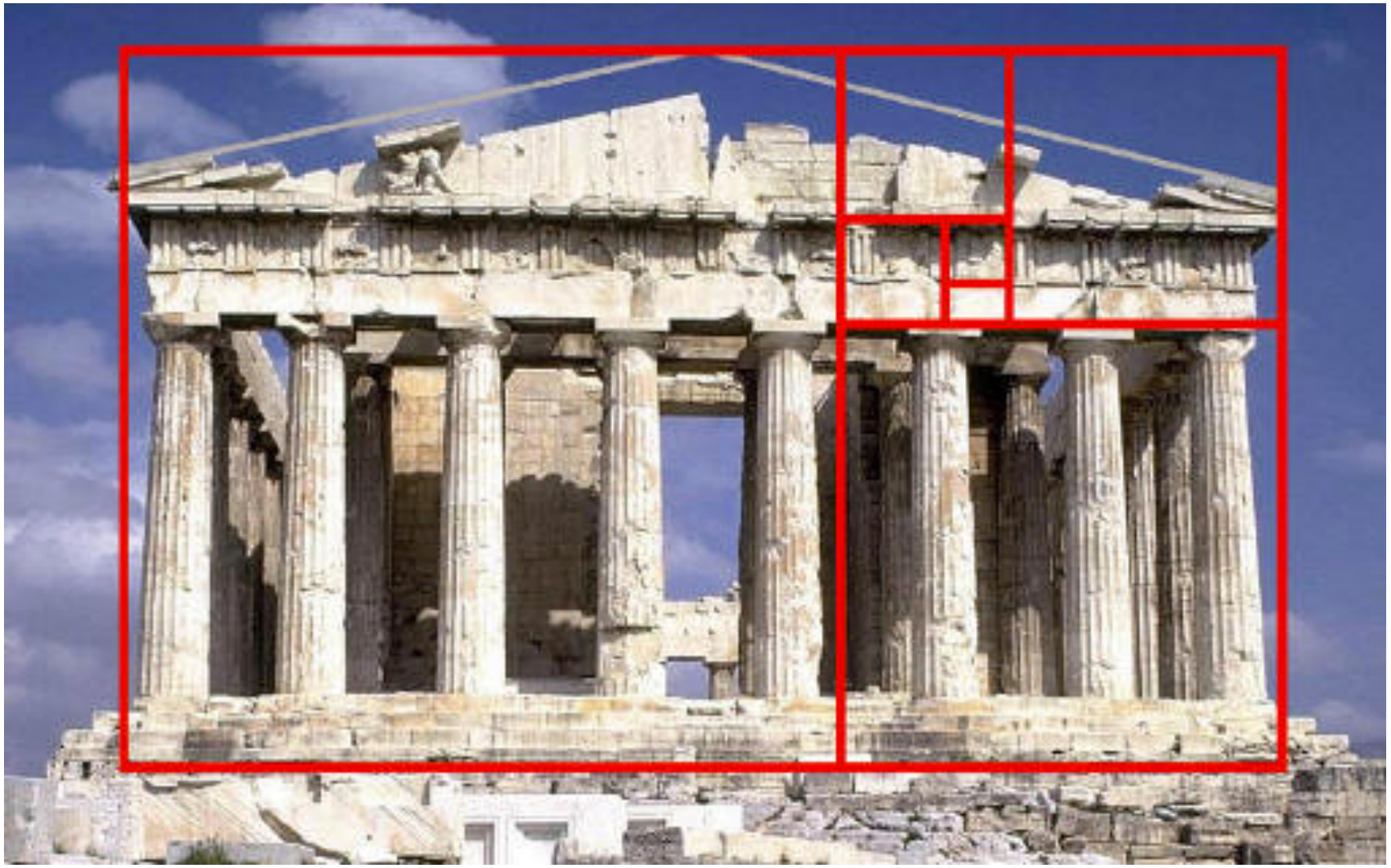


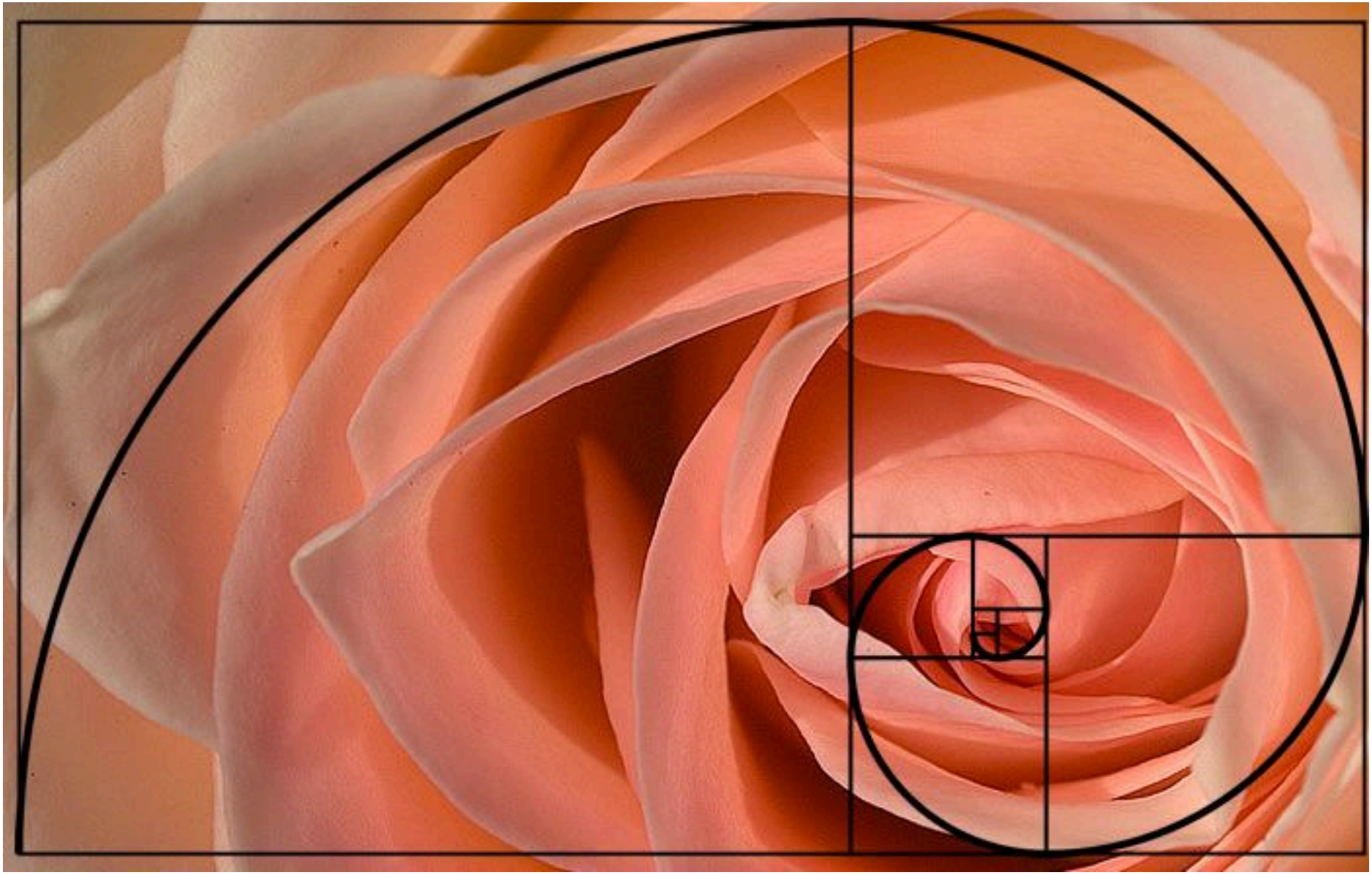
Today's Goals

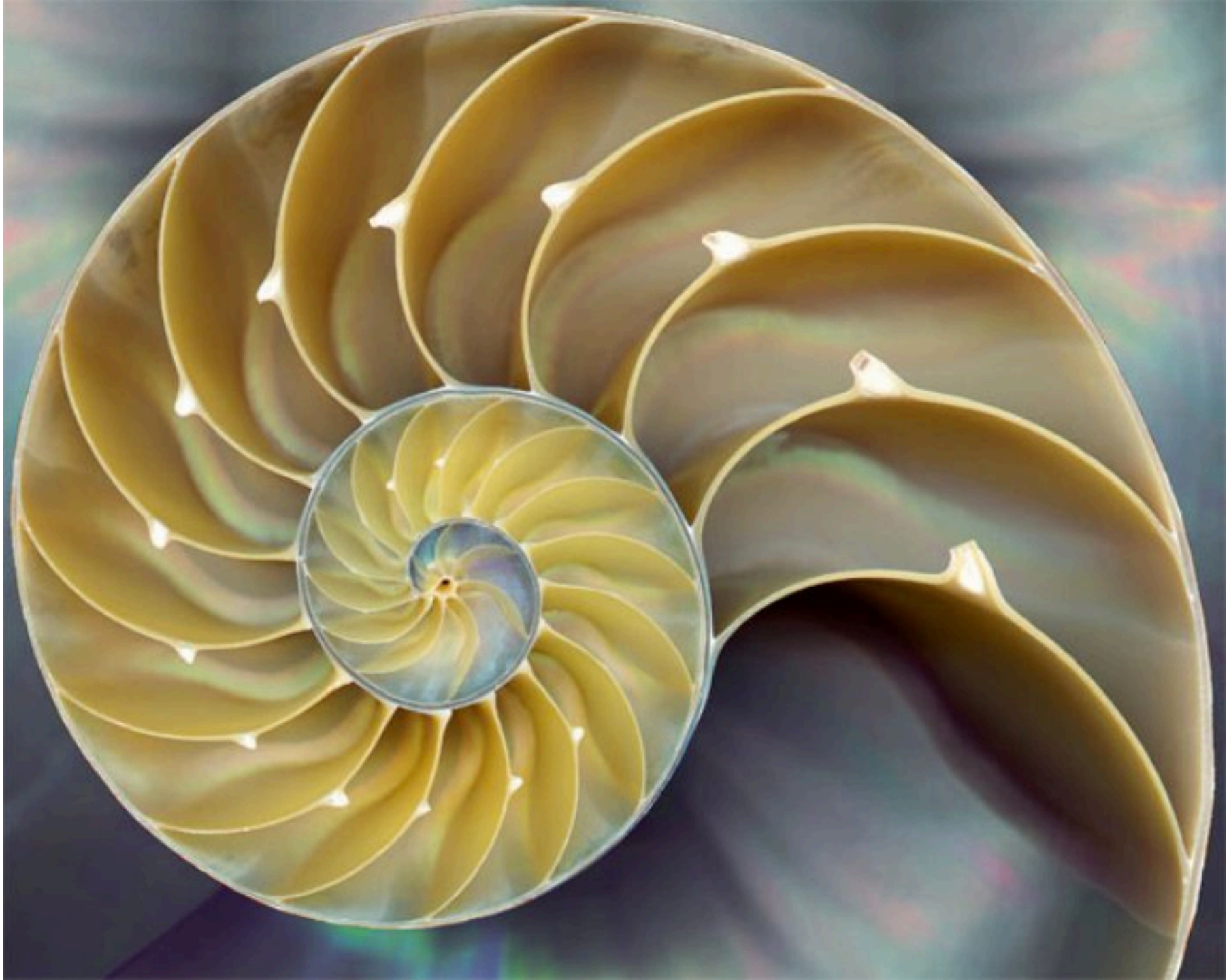


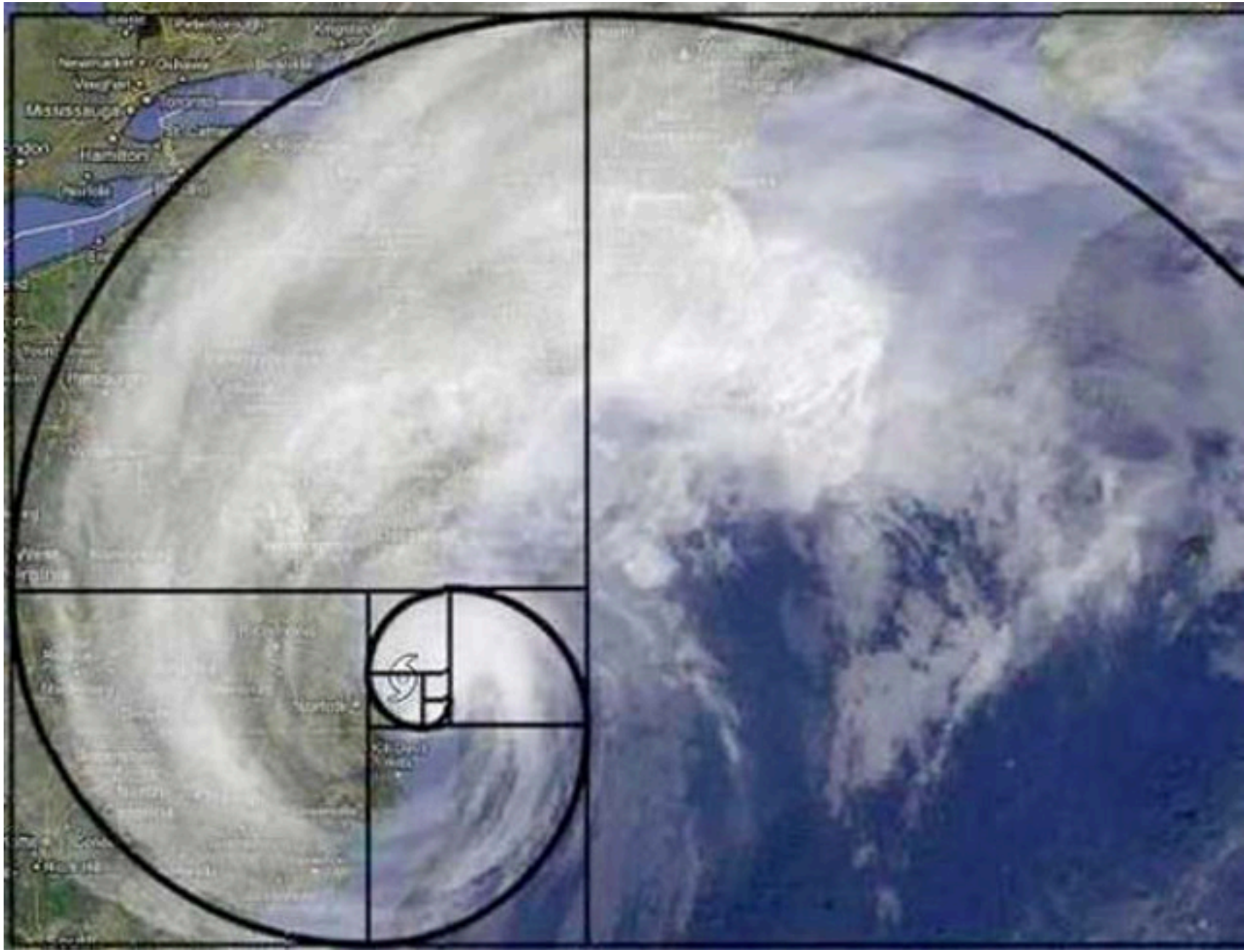

End of month:
No. of Pairs:
1
1
2
1
3
2
4
3
5
5


13
8
2
3
5
21
34








Mississauga
Hamilton

Fibonacci

Fibonacci

$$f(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ f(n-1) + f(n-2) & \text{else} \end{cases}.$$

n = 0, 1, 2, 3, 4, 5, 6…

$f(n)$ = 1, 1, 2, 3, 5, 8, 13 …

Fibonacci

```
int fib(int n) {
  if(n <= 1) {
    // base case
    return 1;
  } else {
    // recursive case
    return fib(n – 1) + fib(n – 2);
  }
}
```

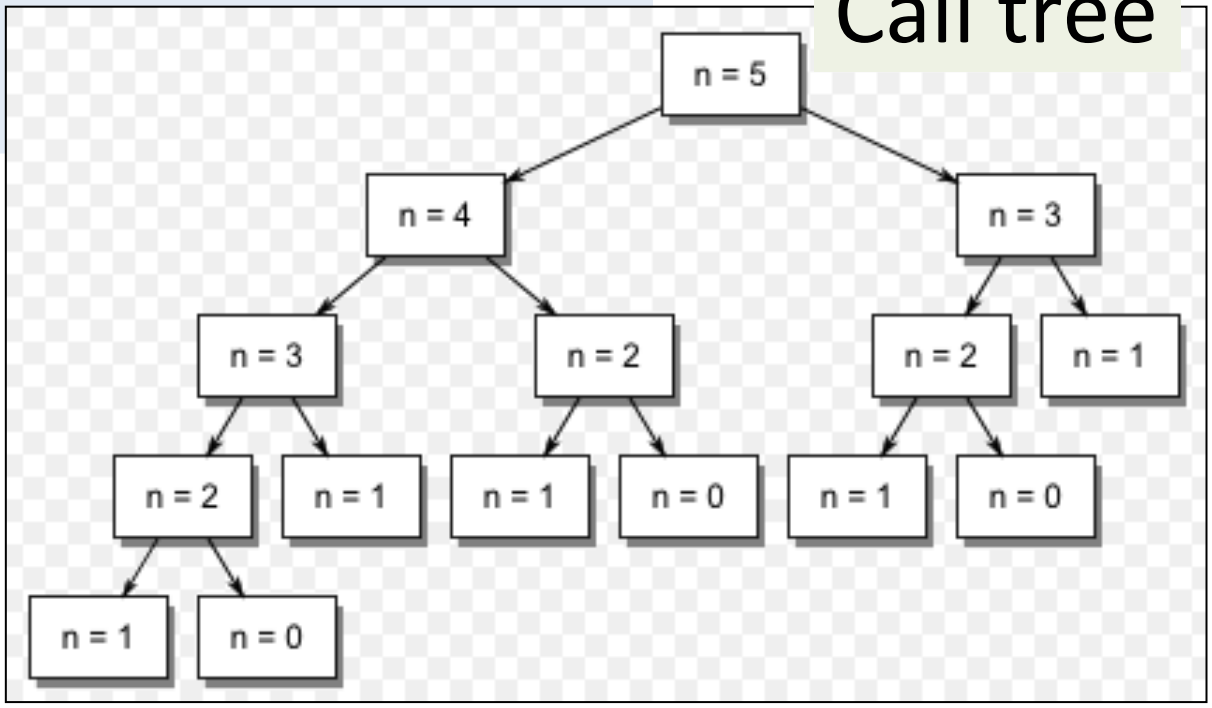
Fibonacci

```
int fib(int n) {
  if(n <= 1) {
    // base case
    return 1;
  } else {
    // recursive case
    return fib(n – 1) + fib(n – 2);
  }
}
```

Fibonacci

```
int fib(int n) {
  if(n <= 1) {
    // base case
    return 1;
  } else {
    // recursive case
    return fib(n – 1) + fib(n – 2);
  }
}
```

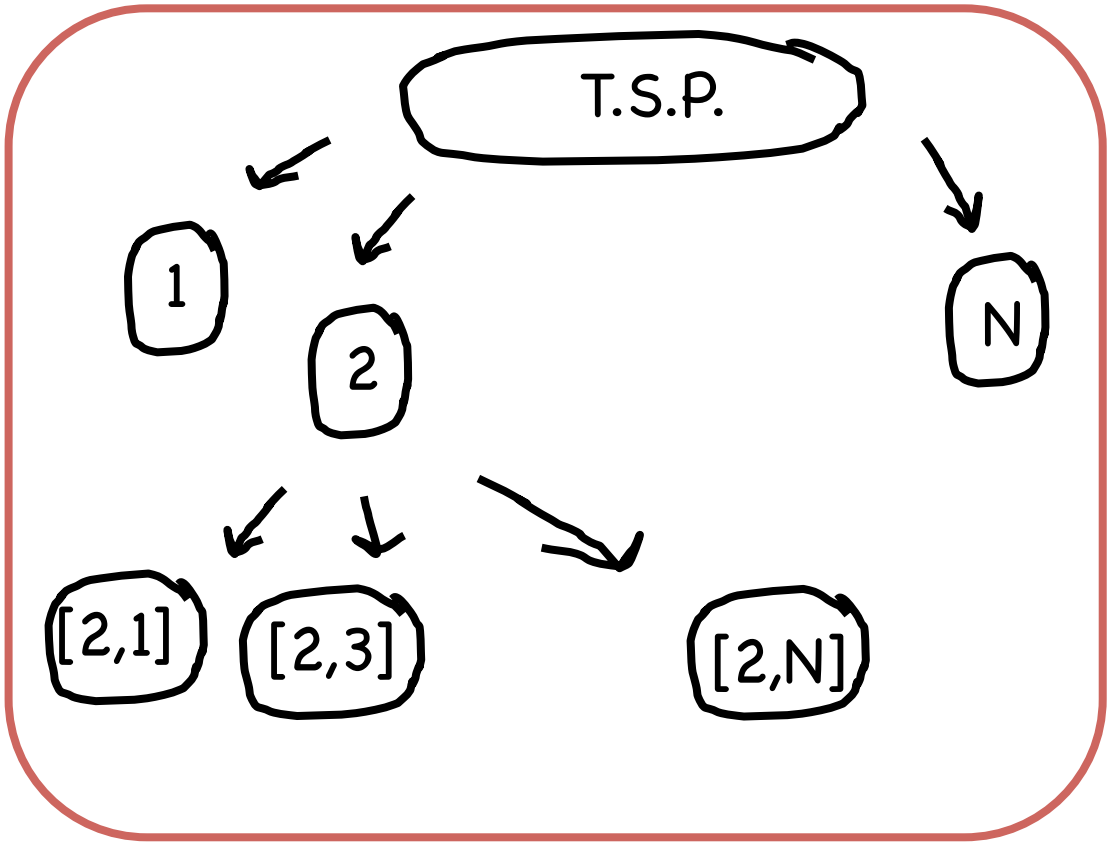
Call tree



Big O?

Big O and Recursion

Count the number of recursive calls

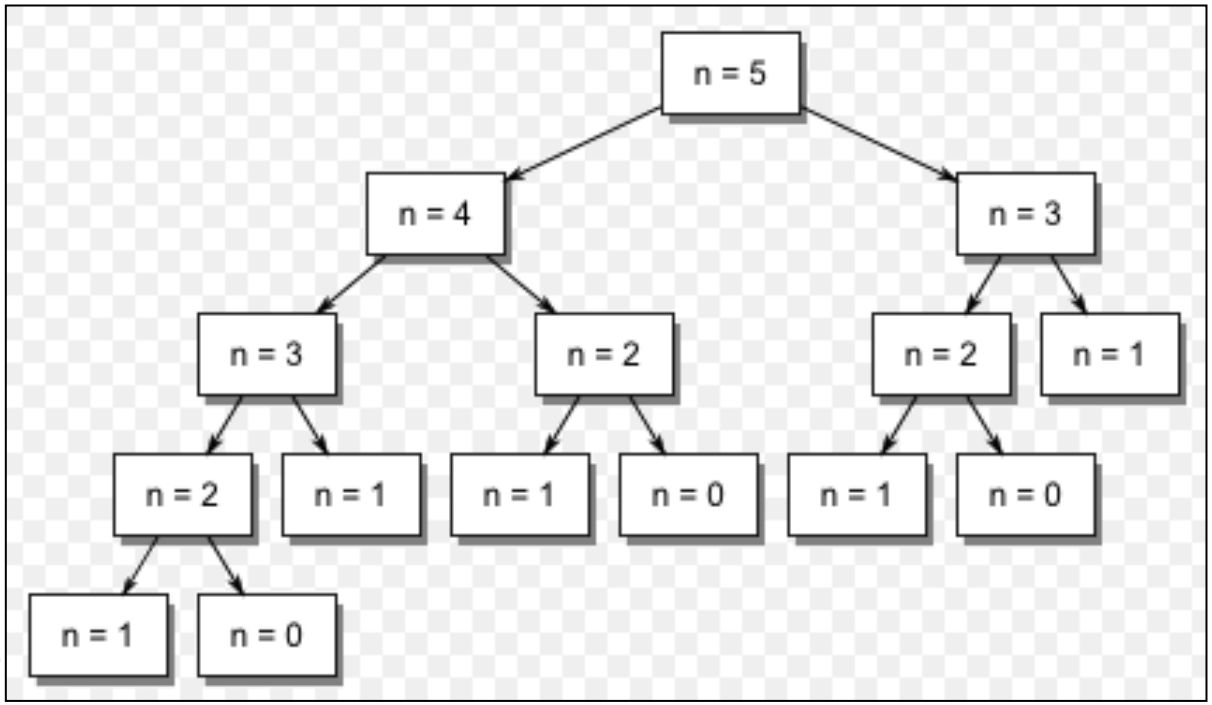


$$\# \, calls = (N)\,(N-1) \times \ldots \times 1$$

$$\mathcal{O}(n!)$$

Big O and Fibonacci

Count the number of recursive calls



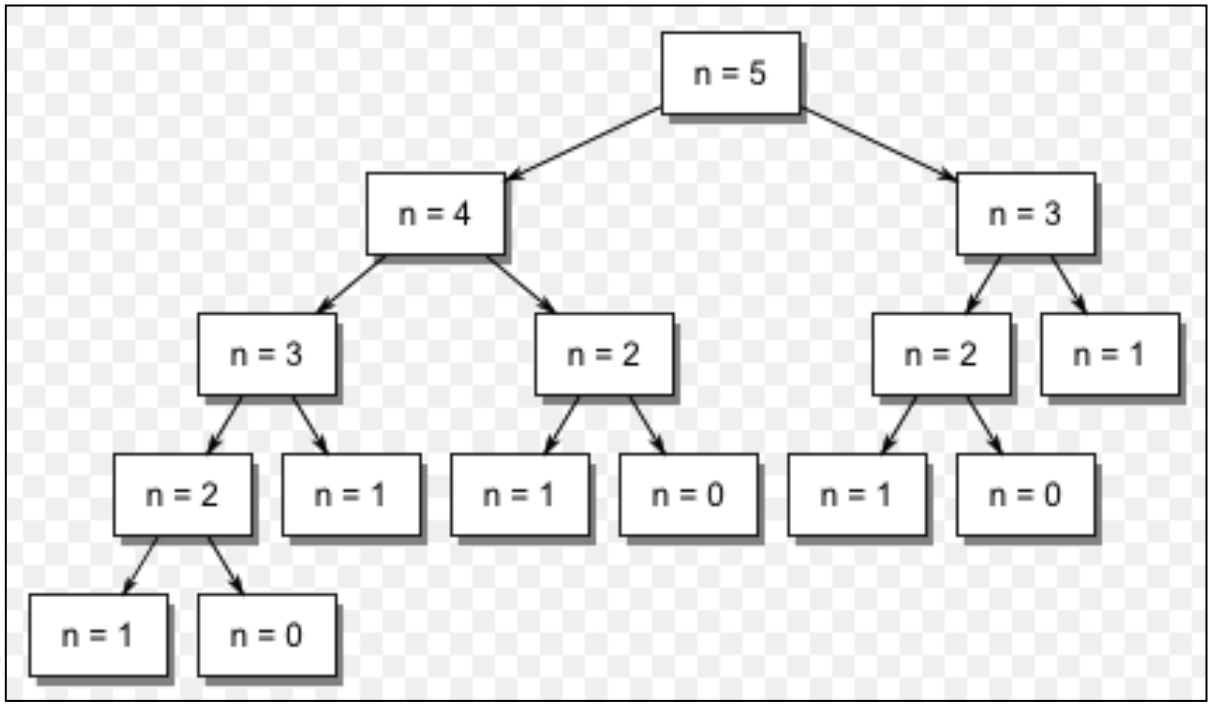
Branching (*b*) = decisions in worse recursive case.
Depth (*d*) = longest chain of recursive calls.

$$\mathcal{O}(b^d) = \mathcal{O}(2^n)$$

Aside

Big O and Fibonacci

Count the number of recursive calls



This is beyond CS106B:

$$\mathcal{O}(\phi^n) = \mathcal{O}(1.62^n)$$

Big O

$\mathcal{O}(1.62^n)$ technically is $\mathcal{O}(2^n)$

since

$$\mathcal{O}(1.62^n) < \mathcal{O}(2^n)$$

We call it a “tighter” bound

End Aside

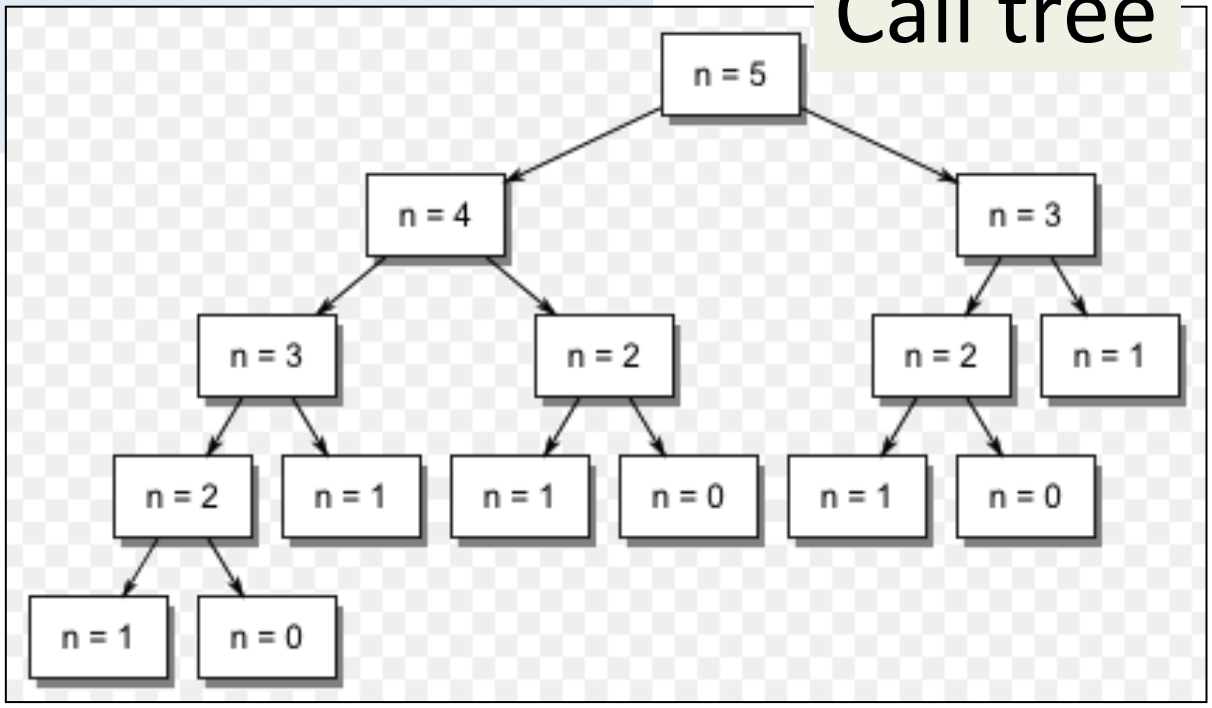
Doesn’t look good.

Call Ghost Busters!

Fibonacci

```
int fib(int n) {
  if(n <= 1) {
    // base case
    return 1;
  } else {
    // recursive case
    return fib(n – 1) + fib(n – 2);
  }
}
```

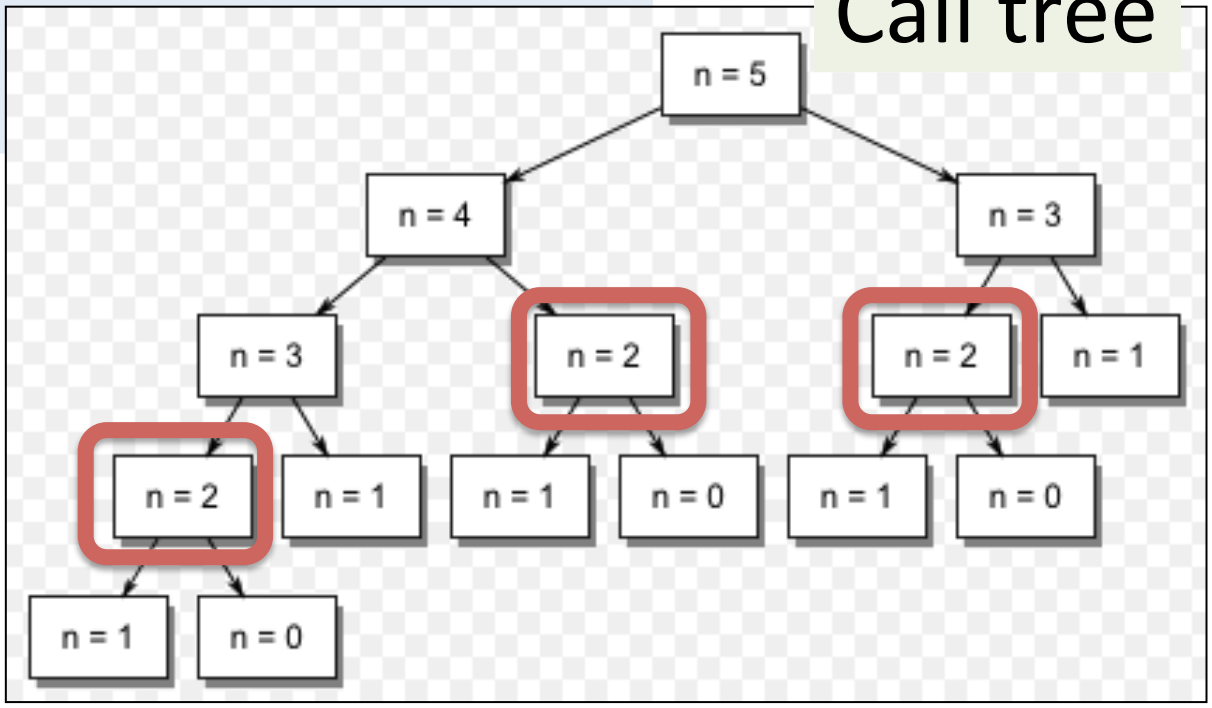
Call tree



Fibonacci

```
int fib(int n) {
  if(n <= 1) {
    // base case
    return 1;
  } else {
    // recursive case
    return fib(n – 1) + fib(n – 2);
  }
}
```

Call tree

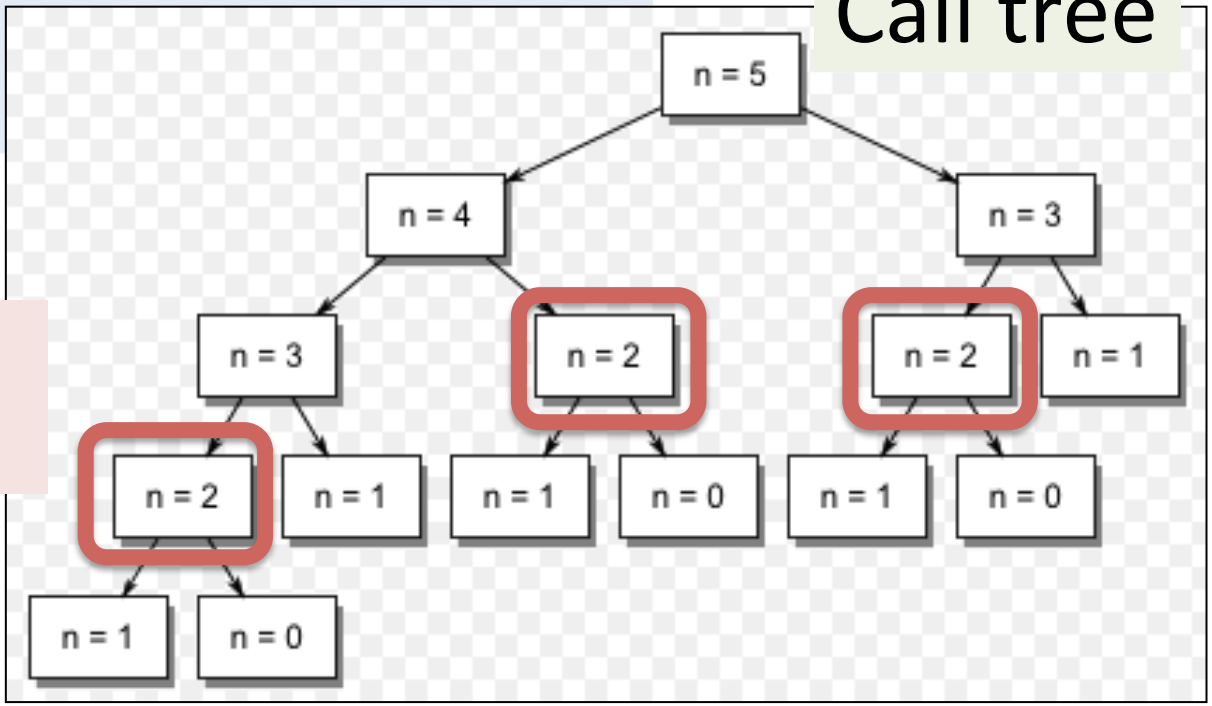


Fibonacci

```
int fib(int n) {
  if(n <= 1) {
    // base case
    return 1;
  } else {
    // recursive case
    return fib(n – 1) + fib(n – 2);
  }
}
```

fib(2) is calculated 3 separate times when calculating fib(5)!

Call tree



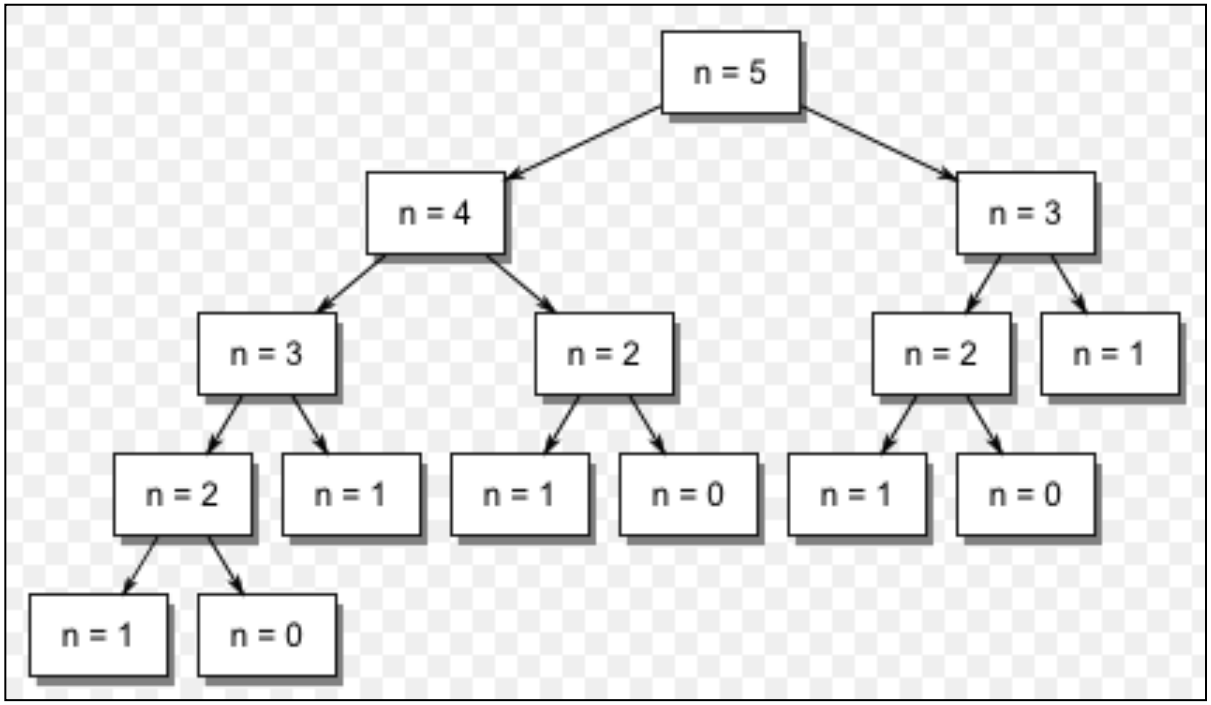
Memoization

Memoization: Store previous results so that in future executions, you don't have to recalculate them.

aka

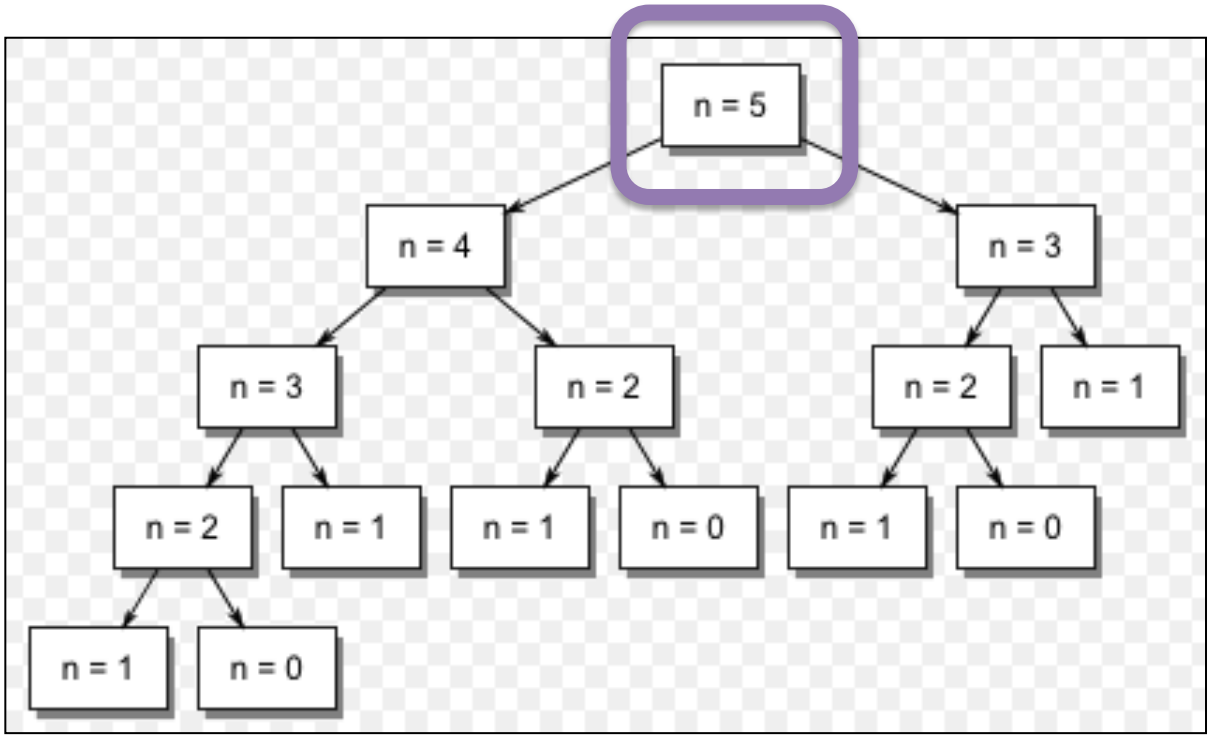
Remember what you have already done!

Memoized Fibonacci



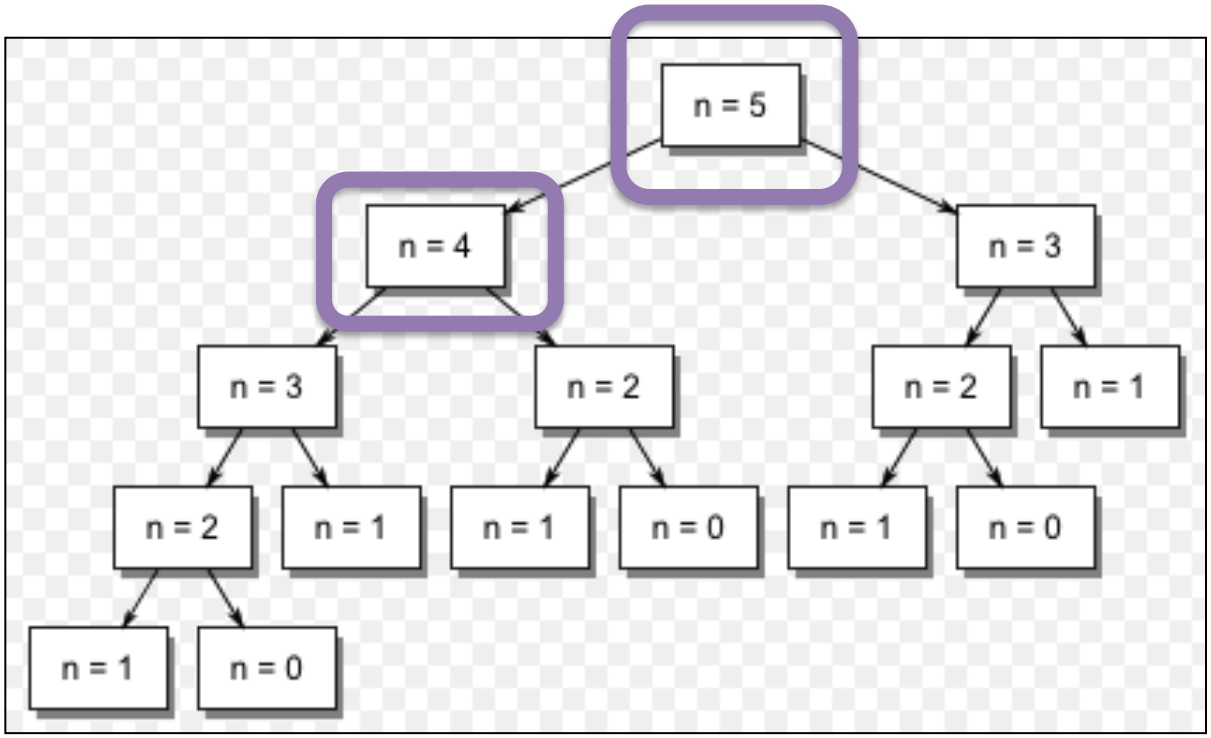
`Cache:`

Memoized Fibonacci



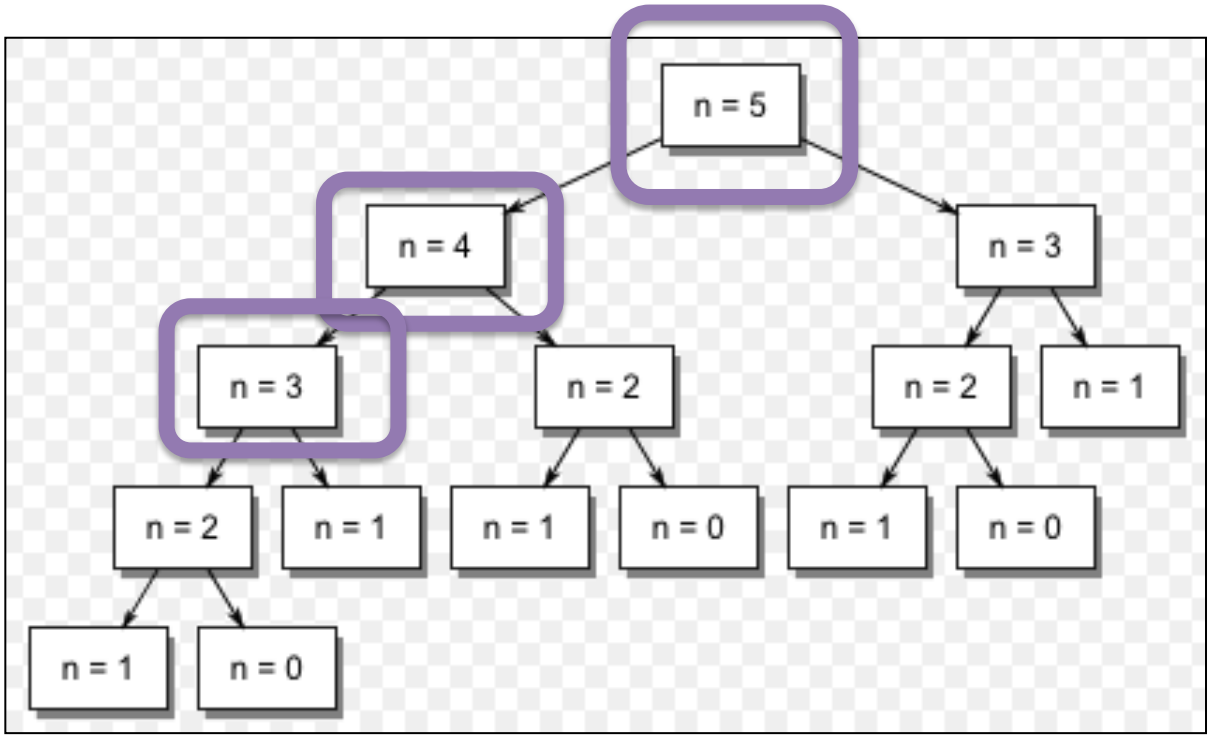
`Cache:`

Memoized Fibonacci



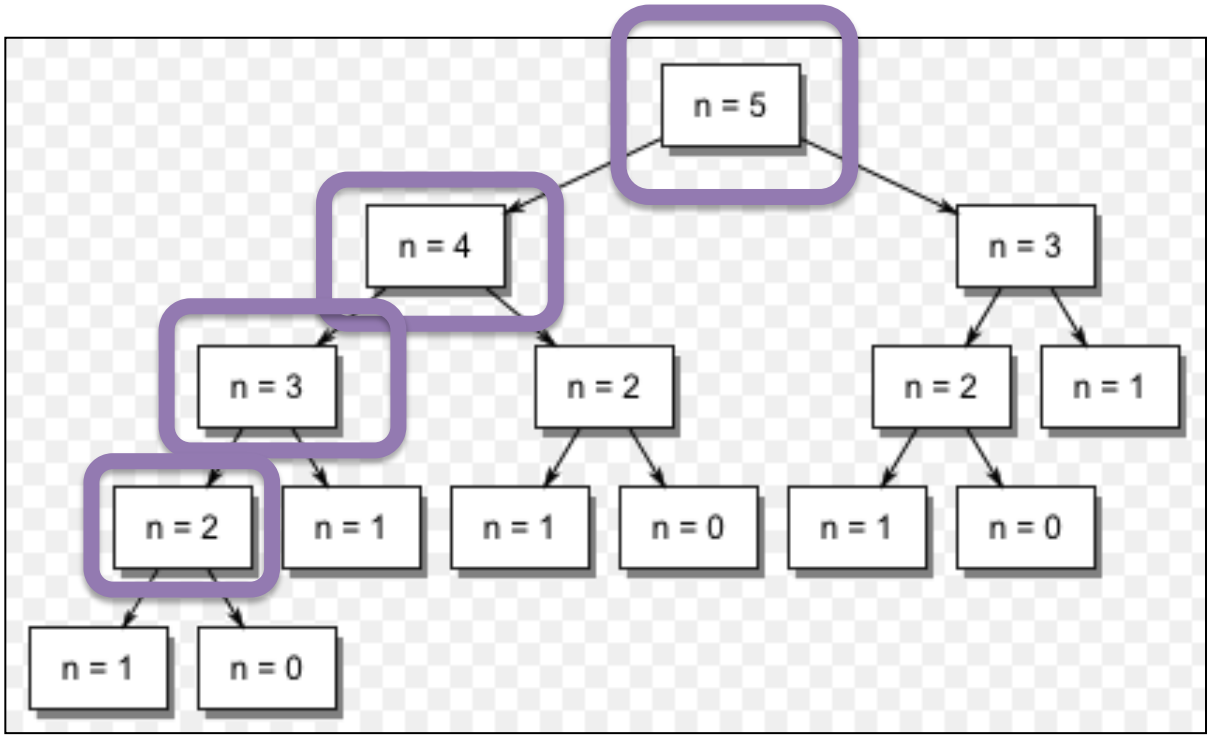
`Cache:`

Memoized Fibonacci



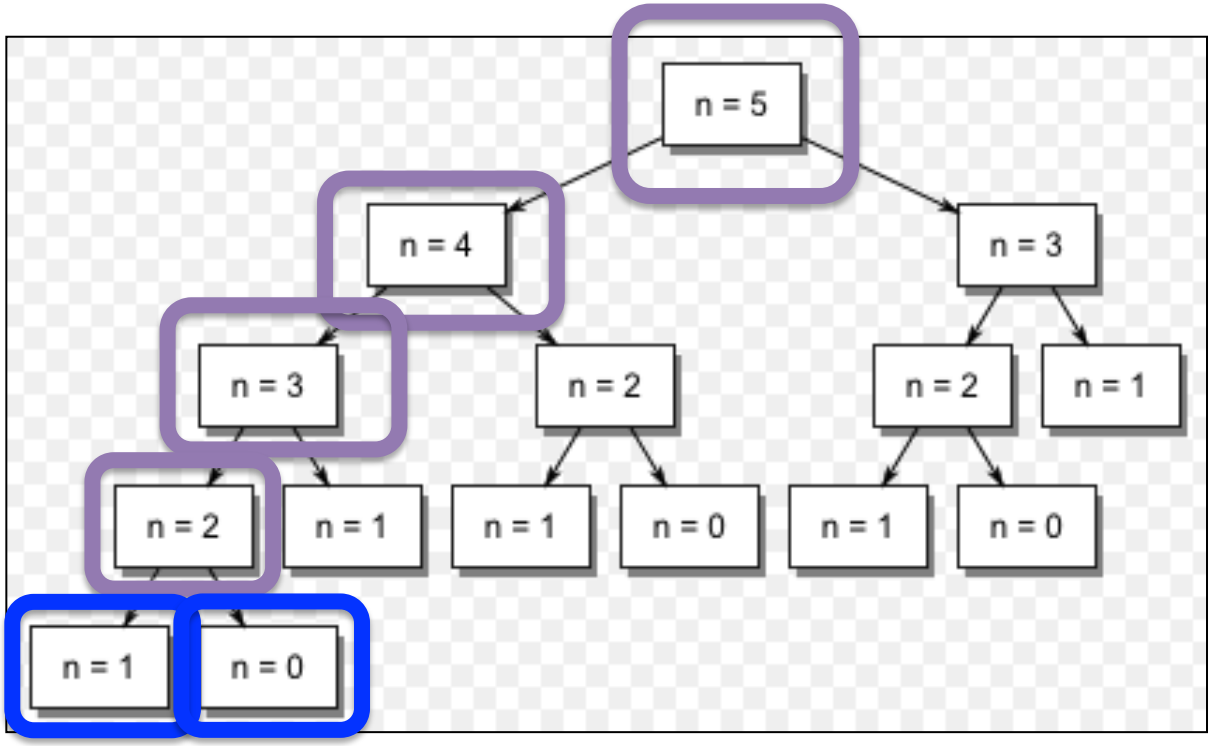
`Cache:`

Memoized Fibonacci



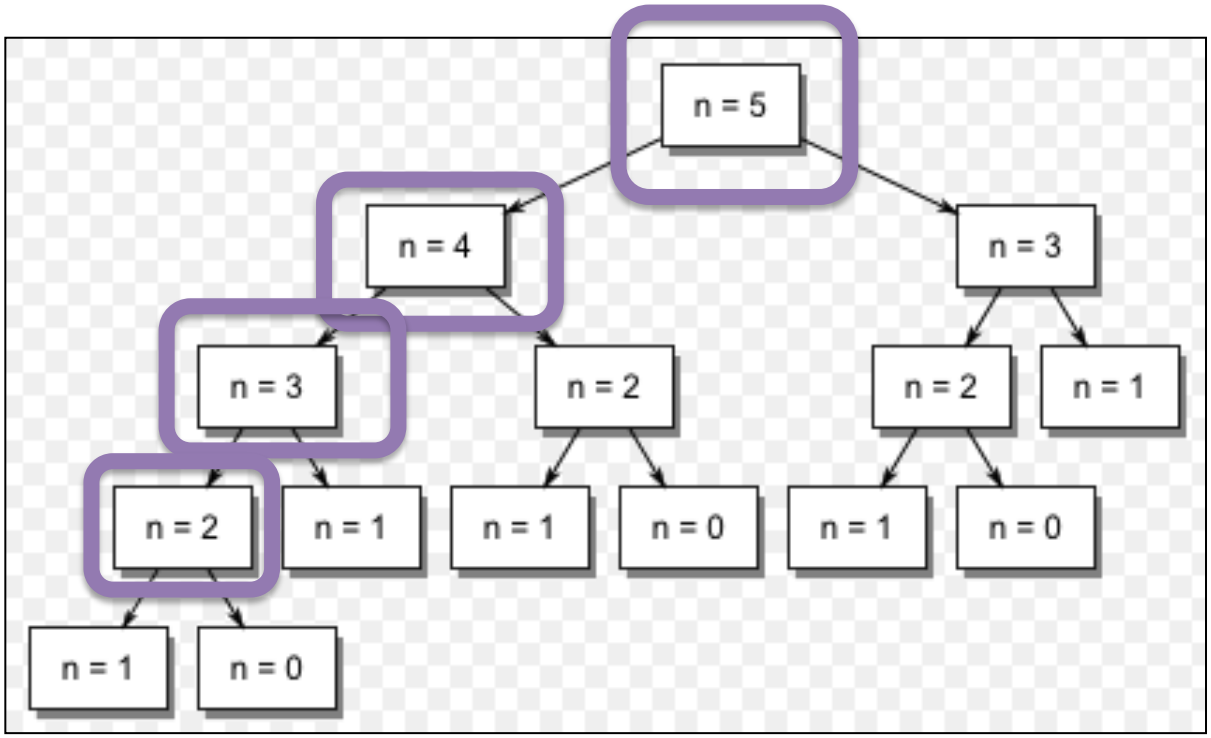
`Cache:`

Memoized Fibonacci



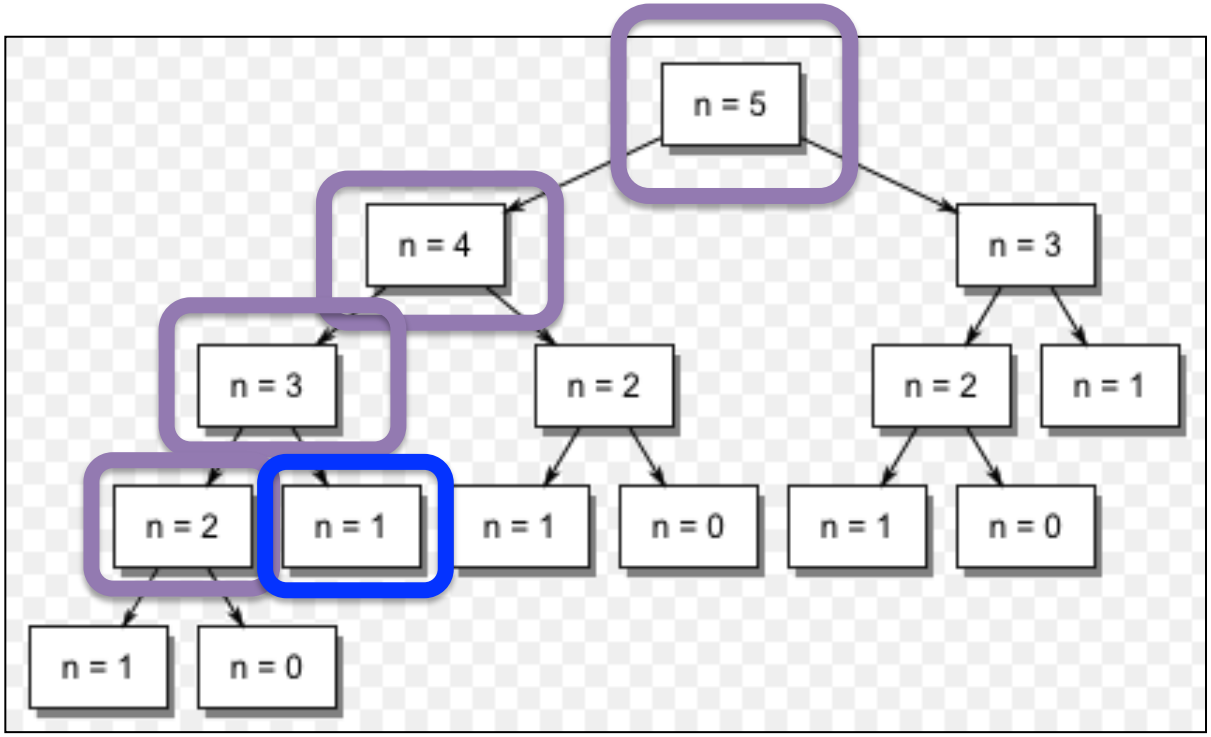

```
Cache:

f(2) = 2
```

Memoized Fibonacci




```
Cache:

f(2) = 2
```

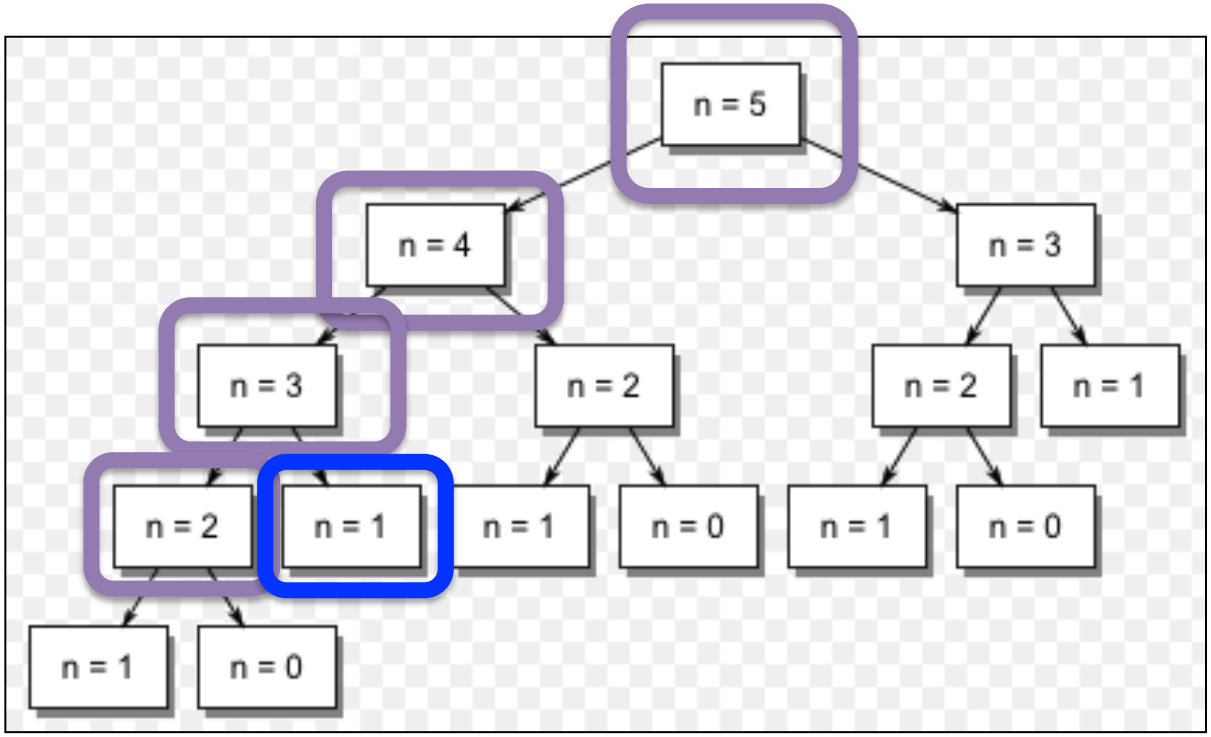
Memoized Fibonacci



Cache:

f(2) = 2

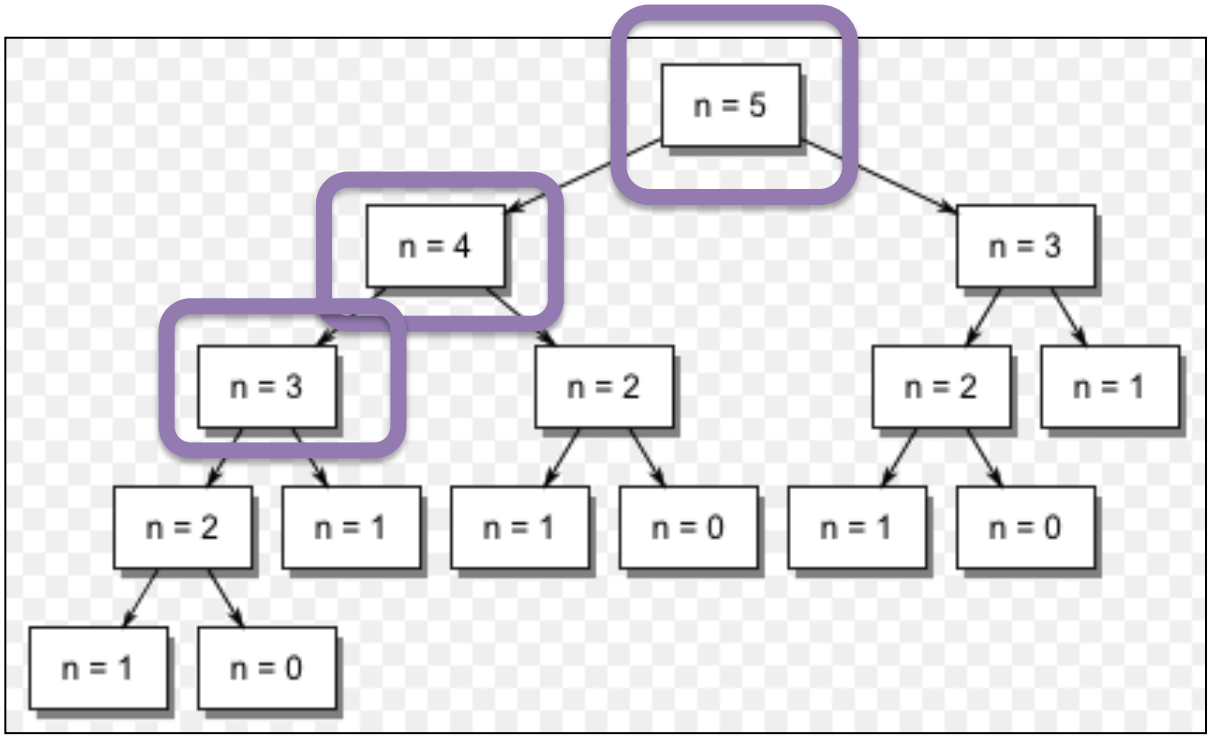
Memoized Fibonacci



Cache:

f(2) = 2, f(3) = 3

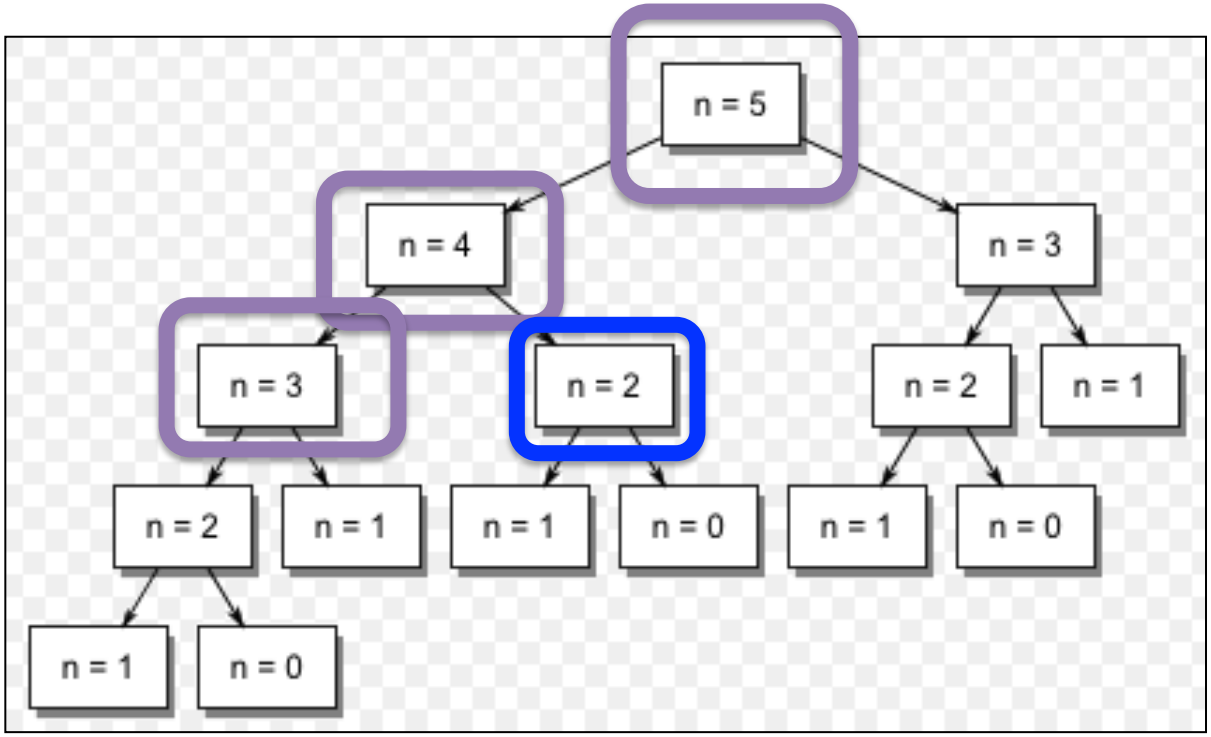
Memoized Fibonacci



Cache:

f(2) = 2, f(3) = 3

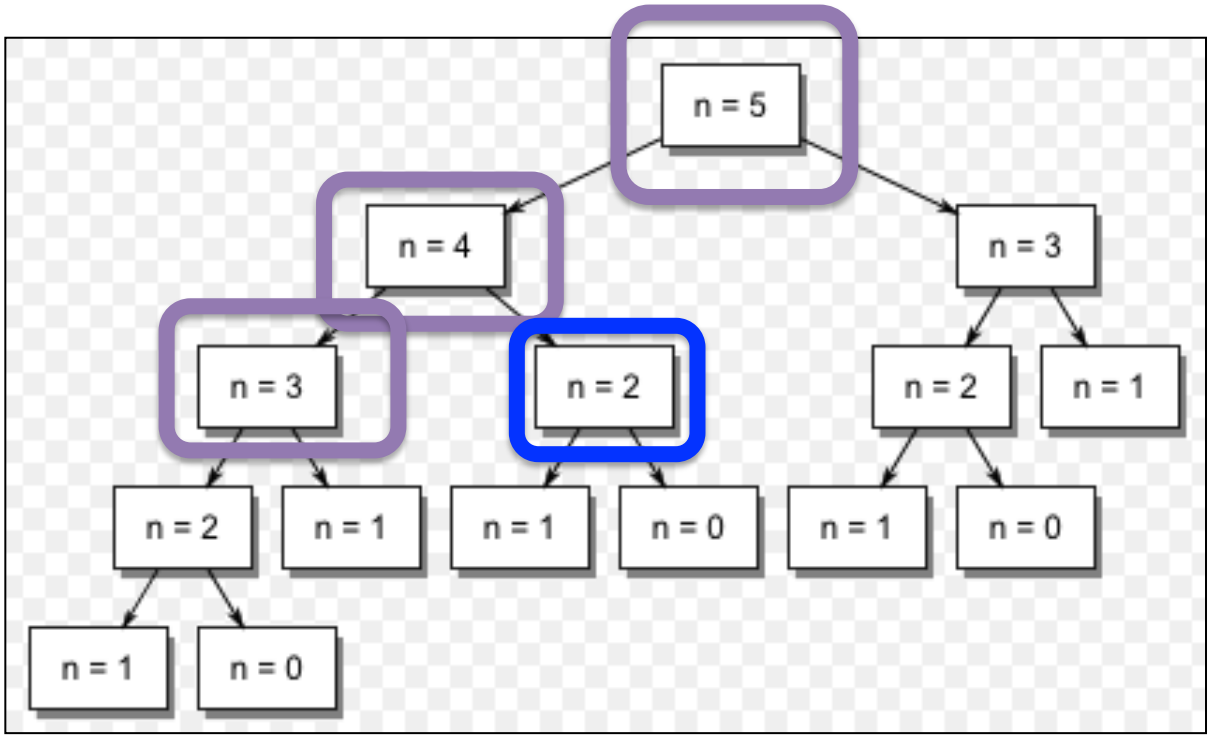
Memoized Fibonacci



Cache:

f(2) = 2, f(3) = 3

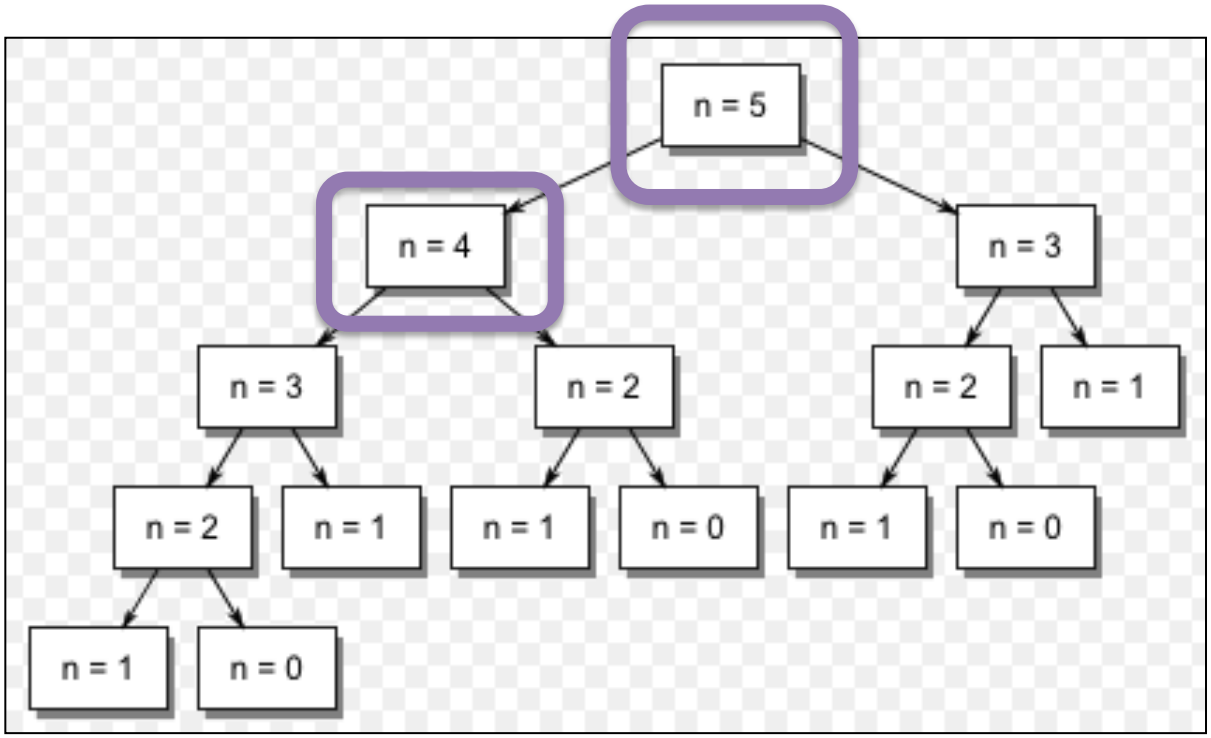
Memoized Fibonacci



Cache:

f(2) = 2, f(3) = 3

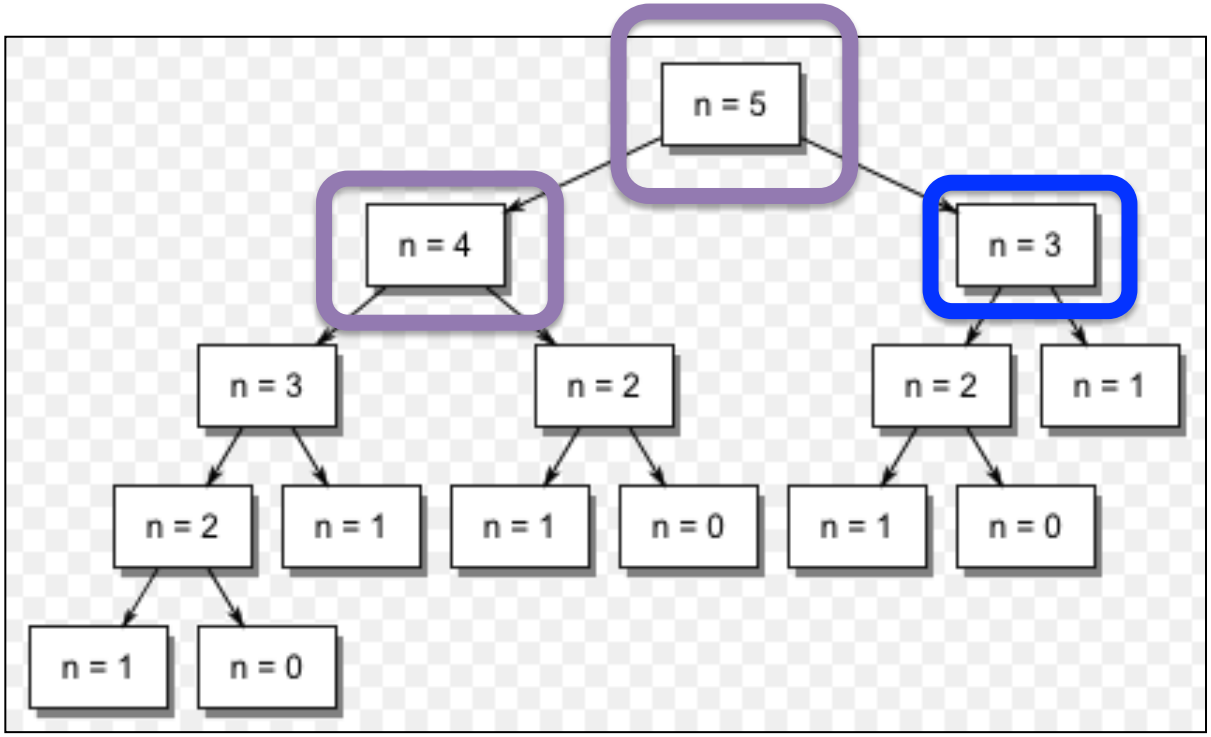
Memoized Fibonacci



Cache:

f(2) = 2, f(3) = 3, f(4) = 5

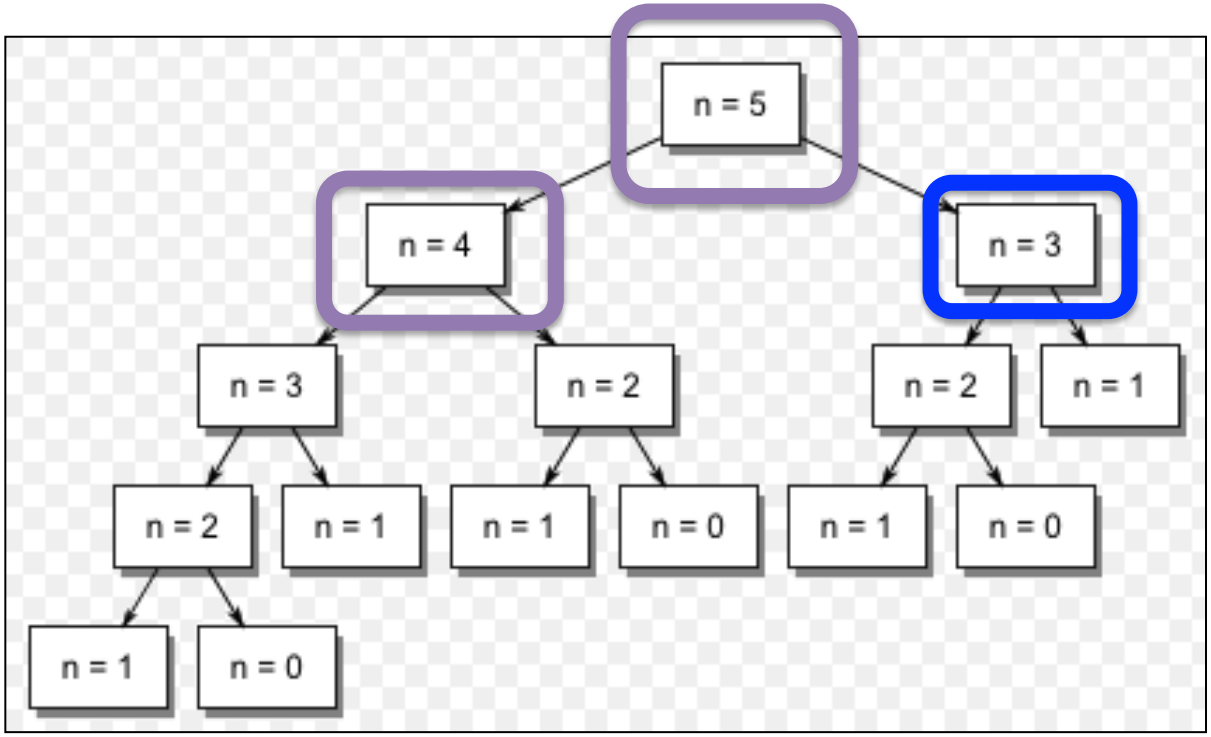
Memoized Fibonacci



Cache:

f(2) = 2, f(3) = 3, f(4) = 5

Memoized Fibonacci



Cache:

f(2) = 2, f(3) = 3 f(4) = 5

Too Fast, Too Furious

Fast Fibb

```
int fastFibb(Map<int, int>&cache, int n) {
  // base case
  if(cache.containsKey(n)) return cache[n];
  if(n <= 1) return 1;

  // recursive case
  int result = fastFibb(cache, n-1) + fastFibb(cache, n-2);
  cache[n] = result;
  return result;
}
```

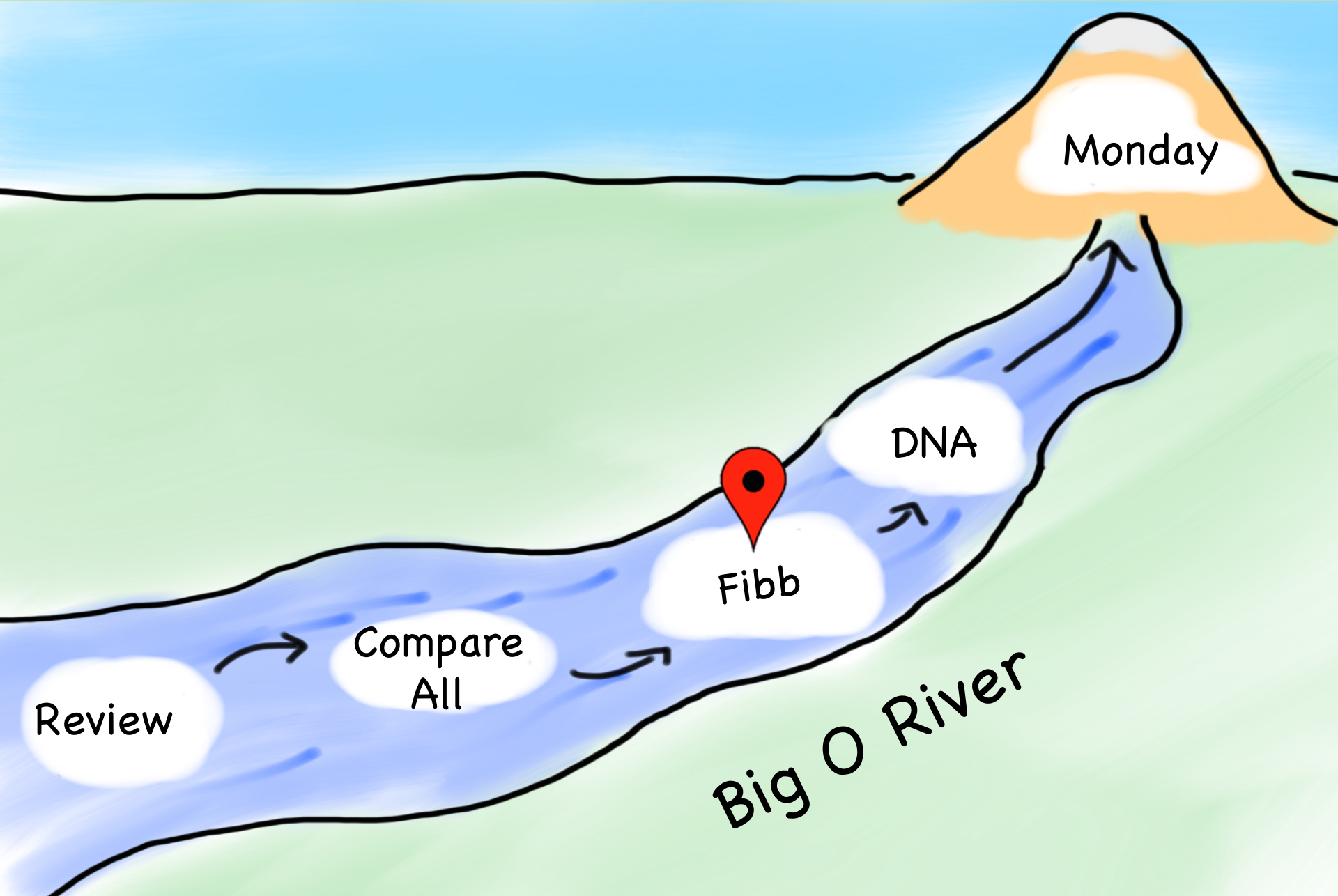
Fast Fibb

```
int fastFibb(Map<int, int>&cache, int n) {
  // base case
  if(cache.containsKey(n)) return cache[n];
  if(n <= 1) return 1;

  // recursive case
  int result = fastFibb(cache, n-1) + fastFibb(cache, n-2);
  cache[n] = result;
  return result;
}

// This is now a wrapper that calls fastFibb
int fibb(int n) {
  Map<int, int> cache;
  return fastFibb(cache, n);
}
```

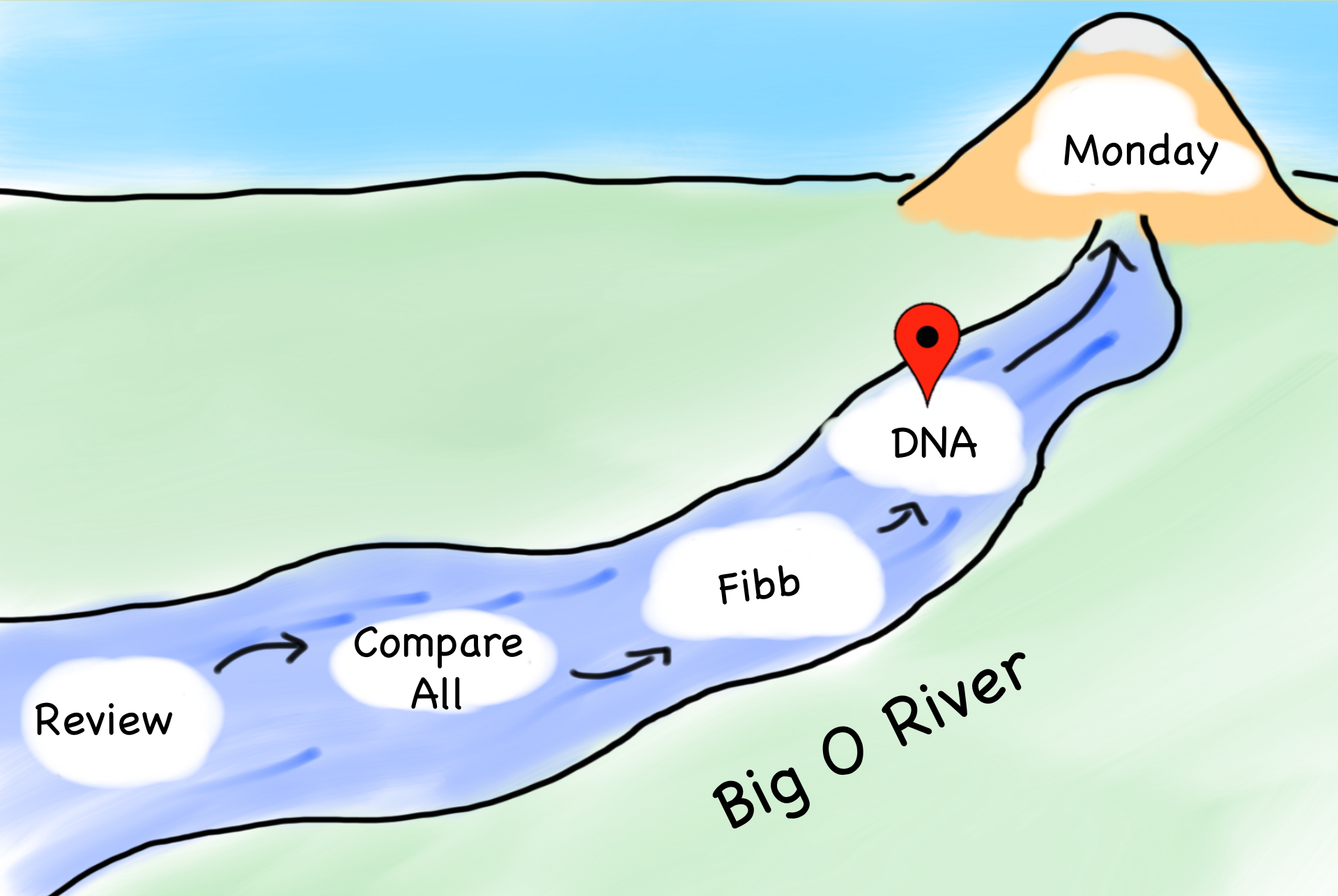
Fast Fibb

$$\mathcal{O}(n)$$

Today's Goals



Today's Goals



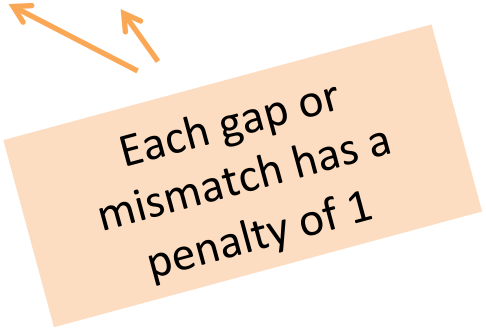
DNA Alignment

Original DNA strings:

```
s1 = GAATTC
s2 = GATTA
```

Optimal Alignment

```
s1 = GAATTC
s2 = GA TTA
```



Dissimilarity

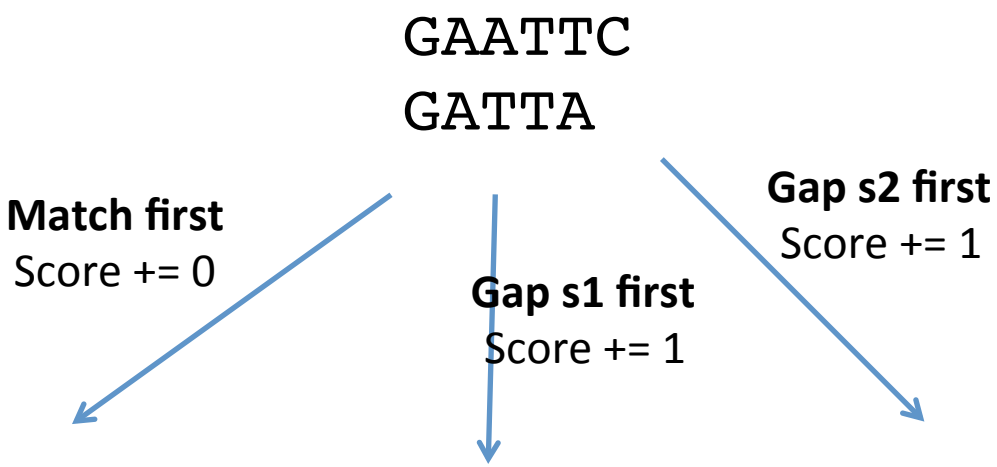
2

DNA Decision Tree

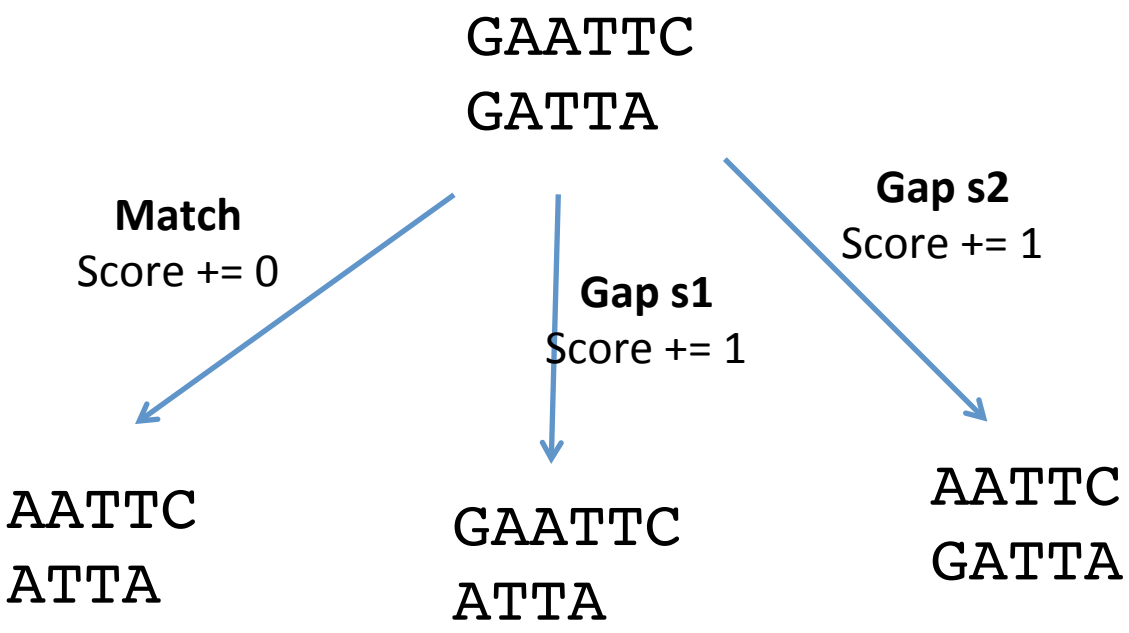
GAATTC

GATTA

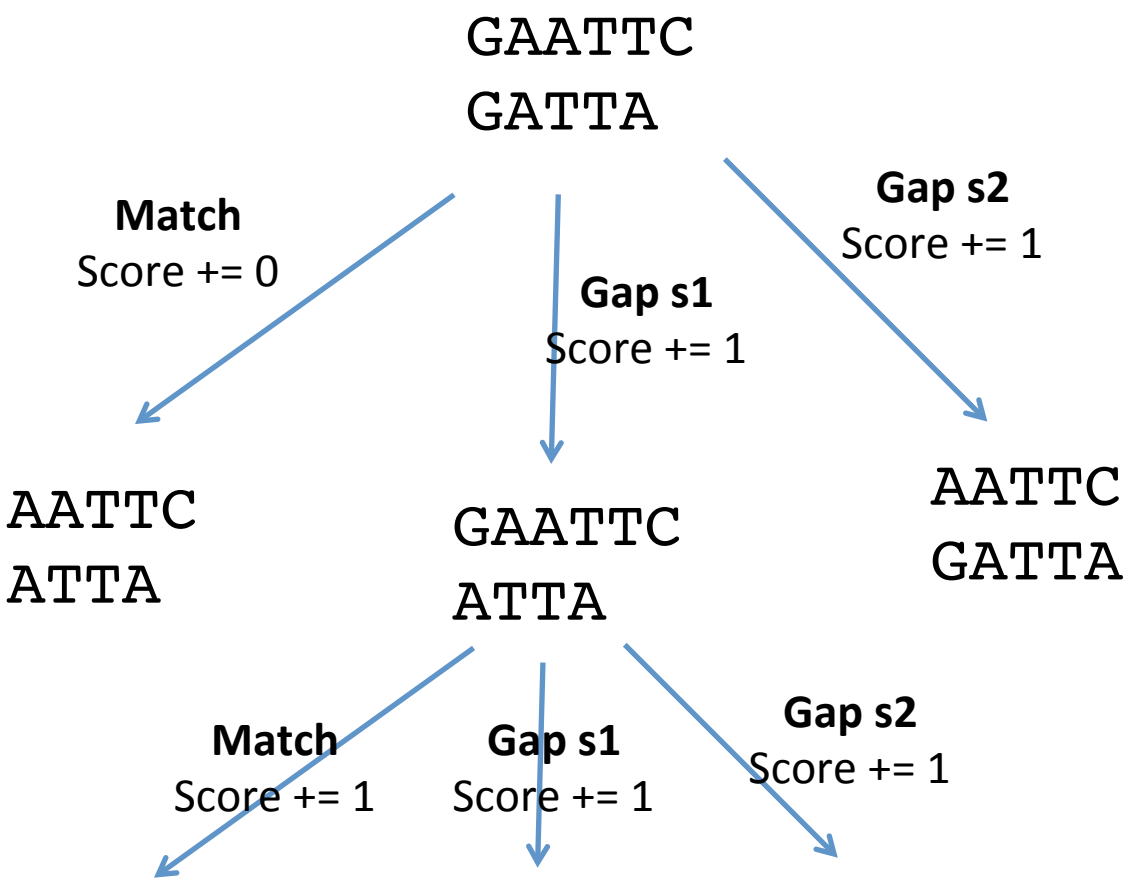
DNA Decision Tree



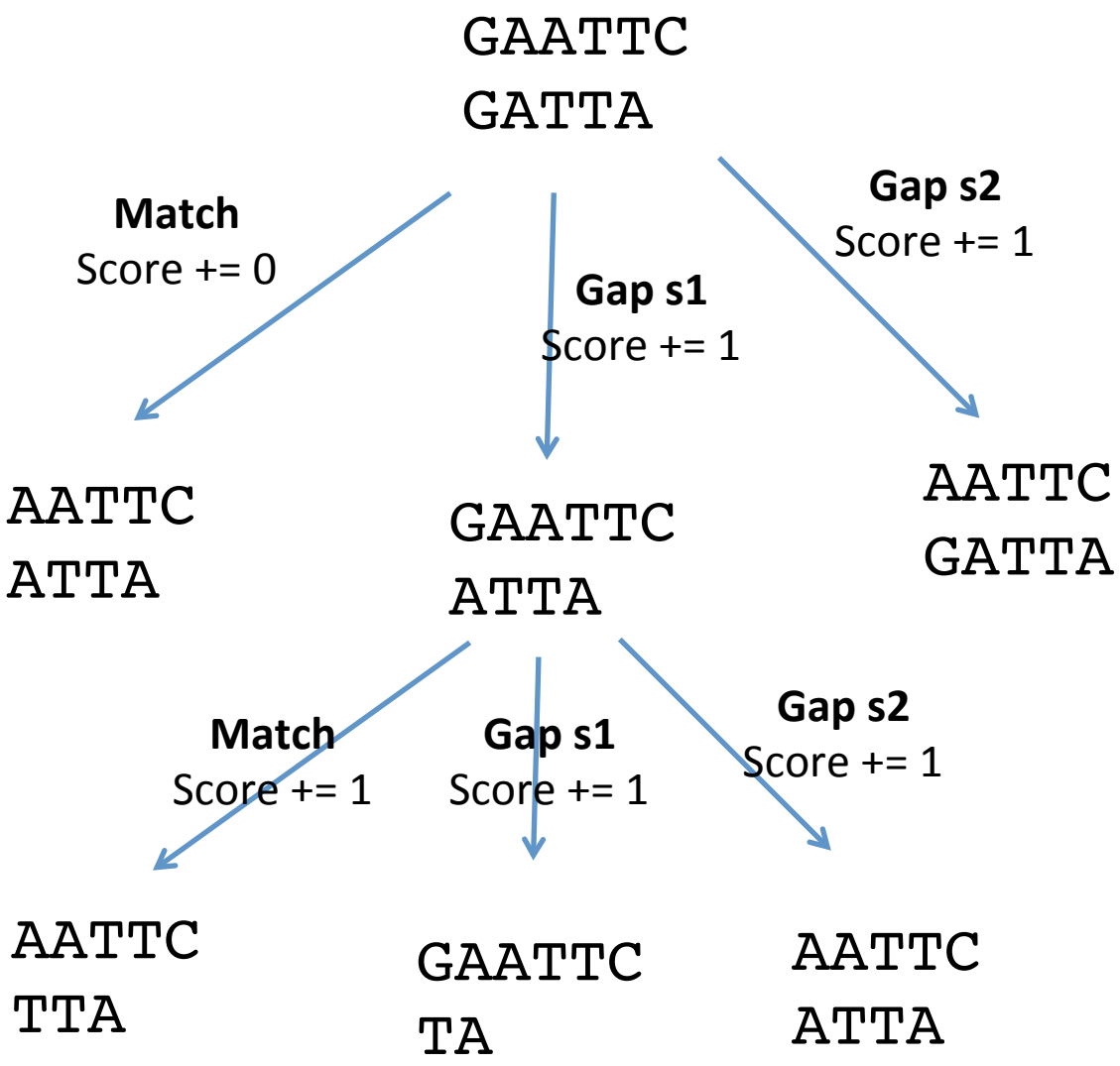
DNA Decision Tree



DNA Decision Tree



DNA Decision Tree



DNA Decision Tree

Try every combination of decisions.

Return the one with the smallest score.

DNA Decision Tree

Without memoization:

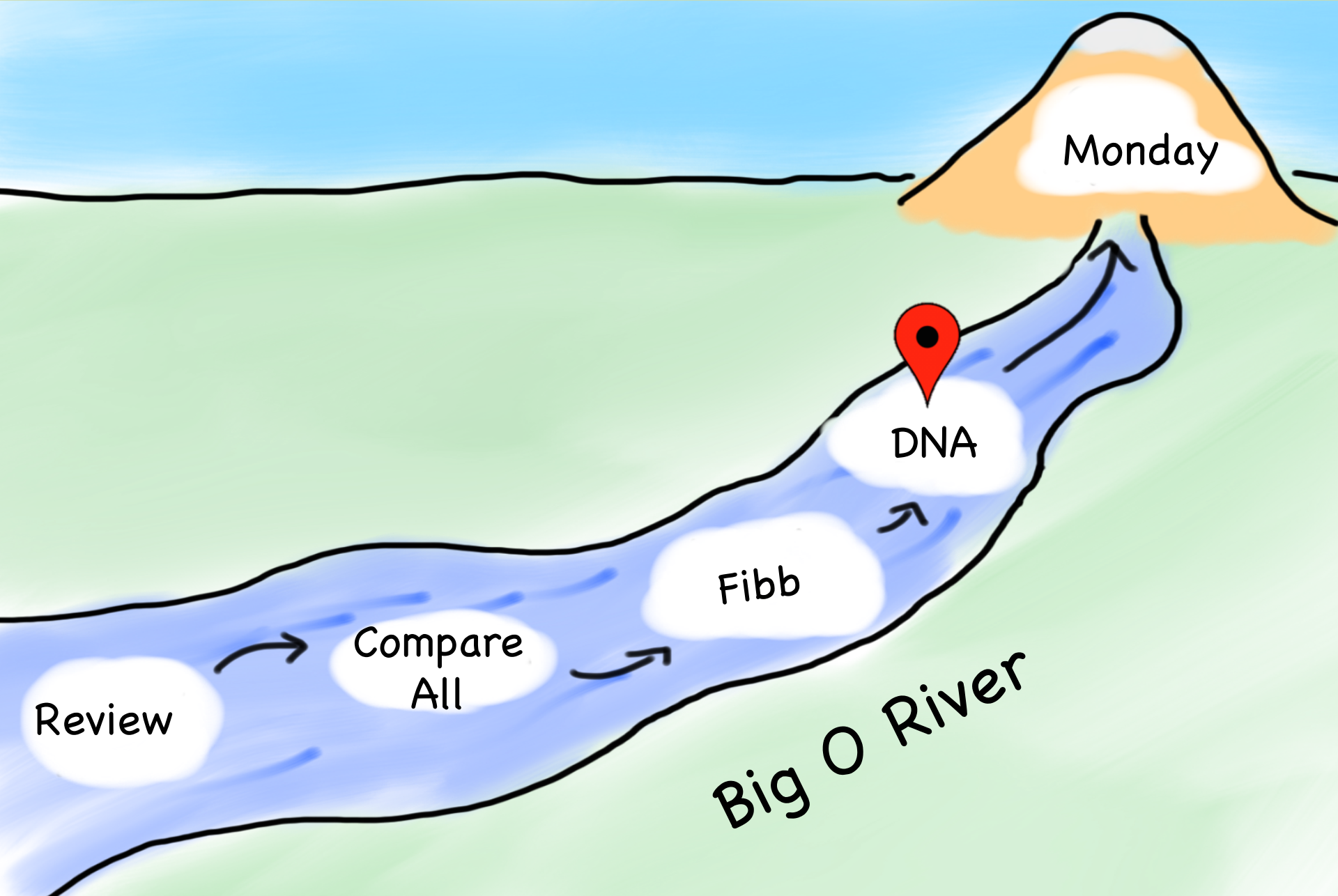
$$\mathcal{O}(3^{m+n})$$ LOL

With memoization:

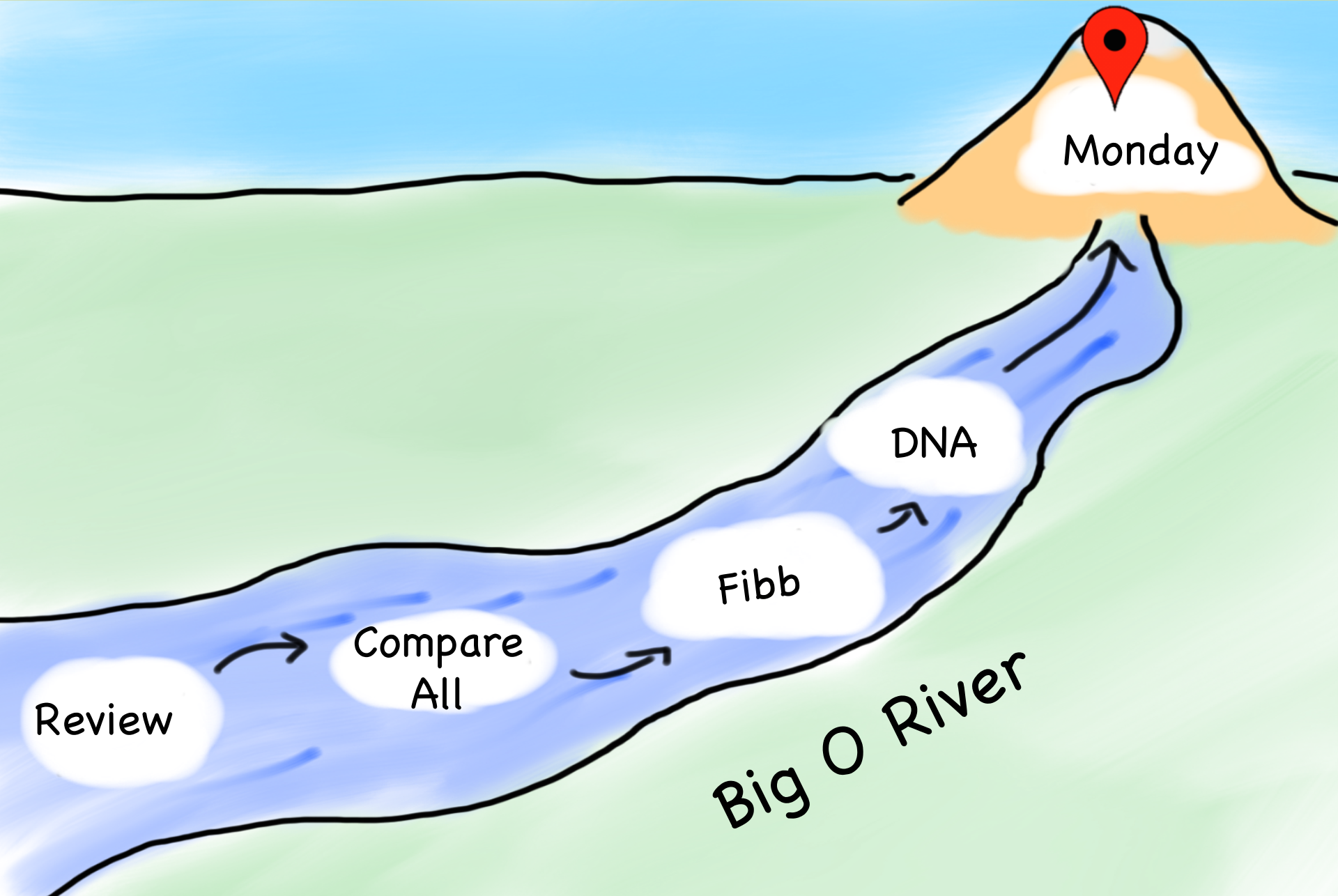
$$\mathcal{O}(m \cdot n)$$ Slow

Where m is the length of str1 and
n is the length of str2

Today's Goals



Today's Goals



Today's Goals

1. Feel Comfort with Big O
2. Understand the benefit of memoization