CS2040S Tutorial 1

T33 Week 3

Please turn on your cameras, this is supposed to be a f2f class so it would be great if I can get to know your faces

Admin Matters

About me

- Zhong Wei
- Y2 Computer Science
- Email: e0725590@u.nus.edu
    - Only if you can’t reach me via Telegram
    - Slower response time

Asking for help

1. Ask in the tutorial Telegram chat
    - Your tutorial mates will appreciate it :)
2. PM me on Telegram
    - For specific questions (e.g. your problem set submission)
    - Or just too shy for (1) – no one's judging!
3. Coursemology comments
    - Same as (2) but slower

Some things

- If I don’t reply on Telegram for >12 hours, message me again
- Please don’t message me a few hours before your deadlines
  - Your deadlines are usually also my deadlines for other modules :)
  - My response times may be slower

Expectations for Tutorials

- I will allow some time for latecomers. I will appreciate it if you can drop me a text if you are late/not coming so I know when to start
- I intend to use slides (may change!), and I'll send the slides to through the Telegram group once all tutorials are over
- Discussion-based. I will experiment with some tutorial style but please feedback me whether it's suitable
- **Stop me if you are lost.** Your learning is important!!

Structure of Tutorials

- Go through the recent lecture topics in brief detail (~10 mins)
- You guys take turns to answer tutorial questions + other random questions

Tutorial Time

Brief Recap of lectures so far

- Time complexity
- Pre/Post-Conditions + Invariants
- Binary Search/Peak Finding

Time complexity

Eg(T(n) = O/Ω/Θ(f(n))

Upper bound O

- T(n) <= c * f(n)

Lower bound Ω

- T(n) >= c * f(n)

‘Tight’ bound Θ (doesn’t always exist)

- T(n) = Θ(f(n)) iff T(n) = O(f(n)) && T(n) = Ω(f(n))

Time complexity

$$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n) < O(n!)$$

Conditions and Invariants

- Pre condition-> conditions that need to be met before your algo runs
- Post condition -> basically the result of your algo
- Invariants(IMPT) -> This is to prove the correctness of your algo

Peak Finding / Binary Search

- Finding an element in O(logn) time given n elements if conditions are right
- Divide and conquer
- Checks the halfway mark, recurse left or right(based on condition) and repeat
- Invariant(Why does it work) : Your answer will always be within the range of values you recurse on

Tutorial Qn 1 - Java Review

Java and JS = car and carpet

Tutorial Qn 1(a)

What is the difference between a class and an object? Illustrate with an example.

Tutorial Qn 1(a) Answer

A class can be seen as a `template', or a `blueprint', specifying what kind of methods/operations should be supported and its behaviour.

An object is an instance of a class.

Tutorial Qn 1(b)

Why does the main method come with a static modifier?

Tutorial Qn 1(b) Answer

- The *static* keyword is used to denote that the method belongs to the class (as a whole) rather than a particular instance
- You can call a *static* member/method without even creating an instance of the class
- Essentially, the main method being static allows it to be called without creating an instance of the object
- You'll learn more in CS2030S!

Tutorial Qn 1(c)

Give an example class (or classes) that uses the modifier private incorrectly (i.e., the program will not compile as it is, but would compile if private were changed to public).

Tutorial Qn 1(c) Answer

```
class SecretHolder {
        private int secret;

        public SecretHolder(int value){
                this.secret = value;
        }
}
```

```
class Test {
        public static void main(String[] args){
        SecretHolder holder = new SecretHolder(5);
        holder.secret = 6; // Compile-time error!
}
```

Tutorial Qn 1(d)(i)

Why do we use interfaces?

Tutorial Qn 1(d)(i) Answer

An interface can be seen as a `contract' that is signed by a class whenever

it implements the interface. The reason for using interfaces is that whenever we see a class that implements that interface, we know for certain that it supports the operations specified by that interface.

Tutorial Qn 1(d)(ii)

Give an example of using an interface.

Tutorial Qn 1(d)(ii) Answer

Problem Set 1.

```
public interface ILFShiftRegister {

    // Sets the value of the shift register to the specified seed.
    public void setSeed(int[] seed);

    // Shifts the register one time, returning the low-order bit.
    public int shift();

    // Shifts the register k times, returning a k-bit integer.
    public int generate(int k);

}
```

Tutorial Qn 1(d)(iii)

Can a method return an interface?

Tutorial Qn 1(d)(iii) Answer

Yes

```
public class ShiftRegisterTest {

    /**
     * getRegister returns a shiftregister to test
     * @param size
     * @param tap
     * @return a new shift register
     * Description: to test a shiftregister, update this function
     * to instantiate the shift register
     */
    ILFShiftRegister getRegister(int size, int tap) { return new ShiftRegister(size, tap); }
```

Tutorial Qn 1(e)

Refer to IntegerExamination.java.

Without running the code, predict the output of the main method.

Can you explain the outputs?

```
public static void main(String[] args) {
    int i = 7;
    myInteger j = new myInteger( k: 7);
    myInteger k = new myInteger( k: 7);

    addOne( i: 7);
    myIntAddOne(j);
    myOtherIntAddOne(k);

    System.out.println(i);
    System.out.println(j);
    System.out.println(k);
}
```

Tutorial Qn 1(e) Answer

I am in addOne. The value of i is 8

I am in myIntAddOne. The value of j is 8

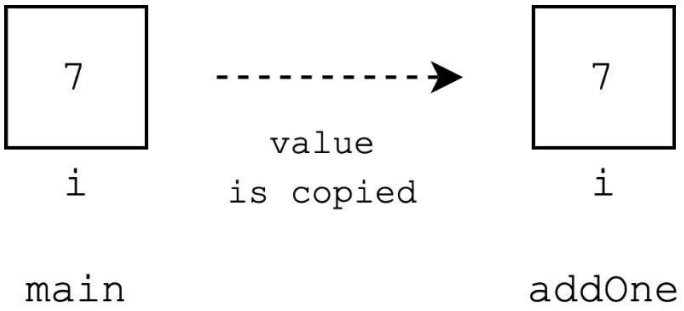
I am in myOtherIntAddOne. The value of k is 8

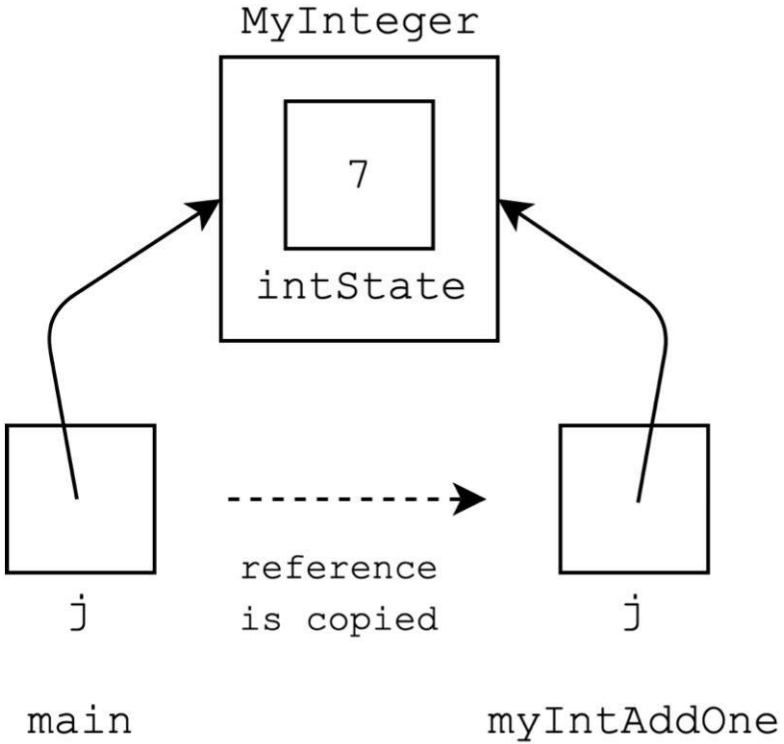
The final value of i back in main is 7

The final value of j back in main is 8

The final value of k back in main is 7

Pass by value





Tutorial Qn 1(f)

Can a variable in a parameter list for a method have the same name as a member (or static) variable in the class? If yes, how is the conflict of names resolved?

Tutorial Qn 1(f) Answer

```
class Example {
    int value = 5;
    public void clash(int value) {
        // Refers to the `value' argument.
        System.out.println(value);
        // Refers to the `value' member in the class.
        System.out.println(this.value);
    }
}
```

Tutorial Qn 2(a)

$$f_1(n) = 7.2 + 34n^3 + 3254n$$

Tutorial Qn 2(a) Answer

$$f_1(n) = O(n^3)$$

Tutorial Qn 2(b)

$$f_2(n) = n^2 \log n + 25n \log^2 n$$

Tutorial Qn 2(b) Answer

$$f_2(n) = O(n^2 \log n)$$

Tutorial Qn 2(c)

$$f_3(n) = 2^{4 \log n} + 5n^5$$

Tutorial Qn 2(c) Answer

$$2^{4 \log n} = (2^{\log n})^4$$
$$= n^4$$

$$f_3(n) = O(n^5)$$

Tutorial Qn 2(d)

$$f_4(n) = 2^{2n^2+4n+7}$$

Tutorial Qn 2(d) Answer

$$f_4(n) = O(2^{2n^2+4n})$$

Break

Try these for fun:

$$f_5(n) = 1/n \qquad f_6(n) = \log_4 n + \log_8 n \qquad f_7(n) = \log\log\log n + \log\log(n^4)$$

$$f_8(n) = (1 - 4/n)^{2n} \qquad f_9(n) = \log(\sqrt{n}) + \sqrt{\log(n)}$$

I'll go through them at the end if we have time

Tutorial Qn 3(a)

$$f(n) = O(n)$$

$$g(n) = O(\log n)$$

Tutorial Qn 3(a)

$$h_1(n) = f(n) + g(n)$$

Tutorial Qn 3(a) Answer

$$h_1(n) \leq c_1 n + c_2 \log n$$
$$= O(n)$$

Tutorial Qn 3(b)

$$h_2(n) = f(n) \times g(n)$$

Tutorial Qn 3(b) Answer

$$
\begin{aligned}
h_2(n) &\leq c_1 n \cdot c_2 \log n \\
&= c_1 c_2 n \log n \\
&= O(n \log n)
\end{aligned}
$$

Tutorial Qn 3(c)

$$h_3(n) = \max(f(n), g(n))$$

Tutorial Qn 3(c) Answer

$$
\begin{aligned}
h_3(n) &= \max(f(n), g(n)) \\
&= O(f(n) + g(n)) \\
&= O(n)
\end{aligned}
$$

Tutorial Qn 3(d)

$$h_4(n) = f(g(n))$$

Tutorial Qn 3(d) Answer

$$
\begin{aligned}
h_4(n) &\leq c_1 g(n) \\
&\leq c_1 c_2 \log n \\
&= O(\log n)
\end{aligned}
$$

Tutorial Qn 3(e)

$$h_5(n) = f(n)^{g(n)}$$

Tutorial Qn 3(b) Answer

$$h(n) = O(n^{c_2 \log n})$$

Tutorial Qn 4

Given a sorted array of n - 1 unique elements in the range [1, n], find the missing element?

Discuss possible naive solutions and possibly faster solutions.

Tutorial Qn 4 Answer

Naive: Go from 1- n, when the index does not match == missing element

Faster: Binary search -> check for $\lfloor n/2 \rfloor$ if index matches the element, left side has no missing element

Tutorial Qn 5

Basically binary search again, but with a twist.

Not your usual if-else cases that you have learnt in the lecture.

Tutorial Qn 5 Answer

Things you know for sure:

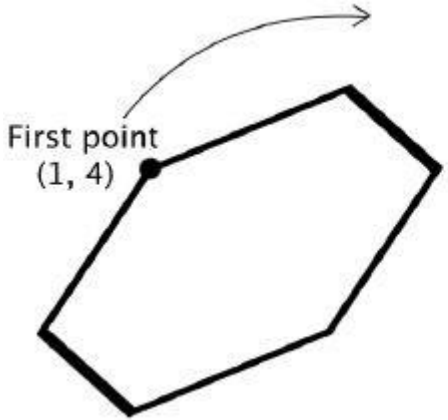
- Minimum k is 1, otherwise you will not do any pieces of homework at all
- Maximum k is the max of all piles of homework. Any number greater will not make sense as this particular k already allow you to finish each pile in 1 hour.

Tutorial Qn 5 Answer

Naïve solution: Linear search 1 -> n

Faster: binary search: Find middle value $\lfloor (1 + n)/2 \rfloor$ of *k*, check if value is valid(finish within *h* hours.
If valid, recurse on left half, to find smaller *k*
If not valid, recurse on right half to find larger *k*

Tutorial Qn 5 Qn



(1, 4)	(4, 5)	(5, 4)	(4, 2)	(2, 1)	(1, 2)

Find bounding box

Tutorial Qn 5 Answer

Find bounding box -> min, max x and y values.

Look for the pattern in the pairs. (aka property of the polygon)

X coords and Y coords can only have 4 graph shapes

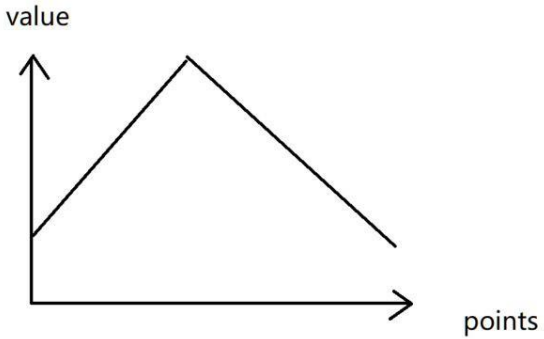
Tutorial Qn 5 Answer

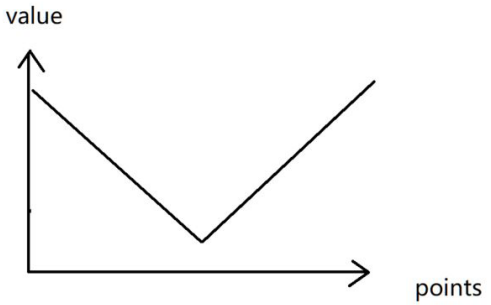
Peak find/binary search on the graph. (ps2)

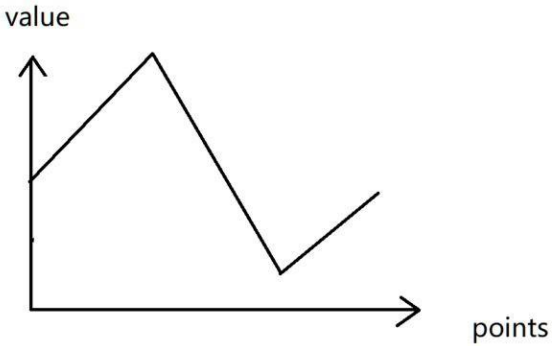
For the zigzag graph, note that since it is a **closed** polygon, the start point will not cross once more with the end point.

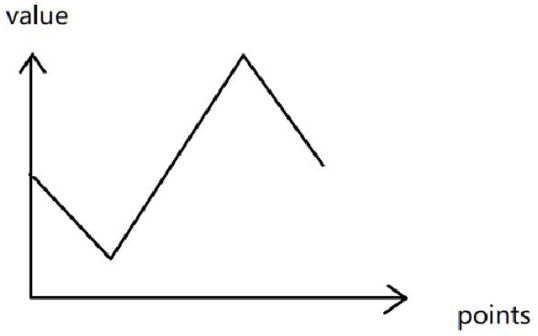
Therefore, by doing bin search, u can guarantee that the *local* maximum/minimum -> global maximum/minimum

Tutorial Qn 5 Answer









More Bonus stuff

$$f_5(n) = 1/n$$

More Bonus stuff answer

$$f_5(n) = O(1)$$

More Bonus stuff

$$f_6(n) = \log_4 n + \log_8 n$$

More Bonus stuff answer

$$f_6(n) = O(\log n)$$

More Bonus stuff

$$f_7(n) = \log\log\log n + \log\log(n^4)$$

More Bonus stuff answer

$$f_7(n) = O(\log \log n)$$

More Bonus stuff

$$f_8(n) = (1 - 4/n)^{2n}$$

More Bonus stuff answer

$$f_8(n) = O(1)$$

More Bonus stuff

$$f_9(n) = \log(\sqrt{n}) + \sqrt{\log(n)}$$

More Bonus stuff answer

$$f_9(n) = O(\log n)$$

More Bonus stuff

$$f(n, m) = n^2 + m \log n + 17$$

More Bonus stuff answer

$O(n^2)$ W.R.T to n

$O(m)$ W.R.T to m

$O(n^2 + m \log n)$ W.R.T to n and m

Appendix: Complexities

Credits to Hanming for the subsequent slides

Complexities

COMMON MISCONCEPTION

Big-O = Worst Case

Big-Omega = Best Case

Big-Theta = Average Case

Not correct!!

Complexities

Correct Interpretation

Big-O = Asymptotic upper bound for some specific case

Big-Omega = Asymptotic lower bound for some specific case

Big-Theta = Asymptotic tight bound for some specific case

Complexities

Correct Interpretation

Big-O = Asymptotic upper bound for some specific case

Big-Omega = Asymptotic lower bound for some specific case

Big-Theta = Asymptotic tight bound for some specific case

This means we can calculate the Big-O, Big-Omega and Big-Theta (if it exists) for any case, e.g. worst case, best case and average case!

Complexities

Correct Interpretation

Big-O = Asymptotic upper bound for some specific case

Big-Omega = Asymptotic lower bound for some specific case

Big-Theta = Asymptotic tight bound for some specific case

This means we can calculate the Big-O, Big-Omega and Big-Theta (if it exists) for any case, e.g. worst case, best case and average case!

i.e. the bounds are independent of the type of case

Best case, Worst case, Average case?

For a given input of size n, complexity also depends on the exact data

Best case, Worst case, Average case?

For a given input of size n, complexity also depends on the exact data

Best case: Minimum time/space needed among all possible inputs of size n

Best case, Worst case, Average case?

For a given input of size n, complexity also depends on the exact data

Best case: Minimum time/space needed among all possible inputs of size n

Worst case: Maximum time/space needed among all possible inputs of size n

Best case, Worst case, Average case?

For a given input of size n, complexity also depends on the exact data

Best case: Minimum time/space needed among all possible inputs of size n

Worst case: Maximum time/space needed among all possible inputs of size n

Average case: Average time/space needed over some finite set of inputs, where each input is of size n. The “finite set” may or may not be the set of all inputs of size n.

Best case, Worst case, Average case?

For a given input of size n, complexity also depends on the exact data

Best case: Minimum time/space needed among all possible inputs of size n

Worst case: Maximum time/space needed among all possible inputs of size n

Average case: Average time/space needed over some finite set of inputs, where each input is of size n. The “finite set” may or may not be the set of all inputs of size n.

Generally, we are most interested in worst-case complexity.

Representing complexity

Generally, we represent complexity as a function of the input size n

e.g. $f(n) = 45n^2 + 23n + 5$

Representing complexity

Generally, we represent complexity as a function of the input size n

$$\text{e.g. } f(n) = 45n^2 + 23n + 5$$

We would then simplify such functions using asymptotic notations:

$$f(n) = \Theta(n^2)$$

Representing complexity

Generally, we represent complexity as a function of the input size n

$$\text{e.g. } f(n) = 45n^2 + 23n + 5$$

We would then simplify such functions using asymptotic notations:

$$f(n) = \Theta(n^2)$$

Why is this okay?

Representing complexity

Generally, we represent complexity as a function of the input size n

$$\text{e.g. } f(n) = 45n^2 + 23n + 5$$

We would then simplify such functions using asymptotic notations:

$$f(n) = \Theta(n^2)$$

*The **time complexity** can be viewed as the number of machine-level instructions (rather than for example, seconds) and that the precise number of instructions for a given algorithm actually depends on the compiler, CPU architecture, and so on. So f(n) can never be exact.*

Representing complexity

Generally, we represent complexity as a function of the input size n

$$\text{e.g. } f(n) = 45n^2 + 23n + 5$$

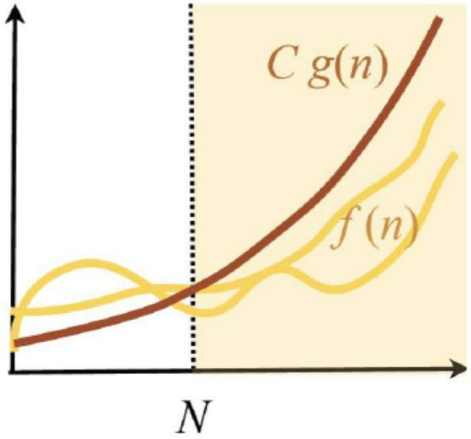
We would then simplify such functions using asymptotic notations:

$$f(n) = \Theta(n^2)$$

*The **time complexity** can be viewed as the number of machine-level instructions (rather than for example, seconds) and that the precise number of instructions for a given algorithm actually depends on the compiler, CPU architecture, and so on. So f(n) can never be exact.*

*Similarly, the **space complexity** depends on the compiler, CPU architecture, and so on.*

Understanding Big-O - Asymptotic Upper Bounds



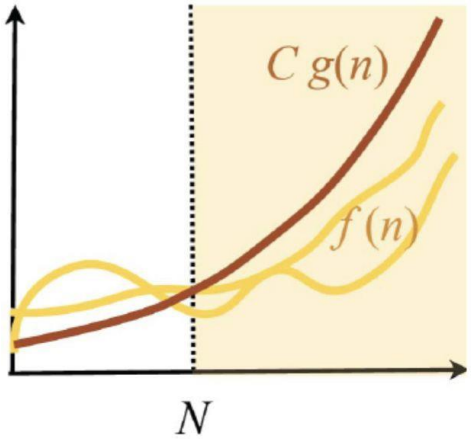
Let f(n) and g(n) be positive valued functions. We say

$$f(n) \in O(g(n))$$

if there exists constants N > 0 and C > 0 such that for all n > N,

$$f(n) \leq Cg(n)$$

Understanding Big-O - Asymptotic Upper Bounds



Let f(n) and g(n) be positive valued functions. We say

$$f(n) \in O(g(n))$$

if there exists constants N > 0 and C > 0 such that for all n > N,

$$f(n) \leq Cg(n)$$

Formally, O(g(n)) is a **set of functions** that contains all functions f(n) satisfying the above conditions. We often abuse notation and say

$$f(n) = O(g(n))$$

Understanding Big-O - Asymptotic Upper Bounds

Quick test: If $f(n) = O(n^2)$, then is $f(n) = O(n^3)$?

Understanding Big-O - Asymptotic Upper Bounds

Quick test: If $f(n) = O(n^2)$, then is $f(n) = O(n^3)$?

Yes, because if we look at $O(n^2)$ and $O(n^3)$ as sets of functions, then we can clearly see that

$$O(n^2) \subseteq O(n^3)$$

Since $f(n) \in O(n^2)$, then $f(n) \in O(n^3)$.

Understanding Big-O - Asymptotic Upper Bounds

Quick test: If $f(n) = O(n^2)$, then is $f(n) = O(n^3)$?

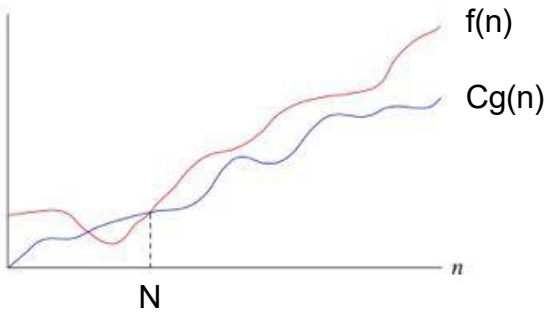
Yes, because if we look at $O(n^2)$ and $O(n^3)$ as sets of functions, then we can clearly see that

$$O(n^2) \subseteq O(n^3)$$

Since $f(n) \in O(n^2)$, then $f(n) \in O(n^3)$.

So this upper bound **may not** be the tightest upper bound!

Understanding Big-Ω - Asymptotic Lower Bounds



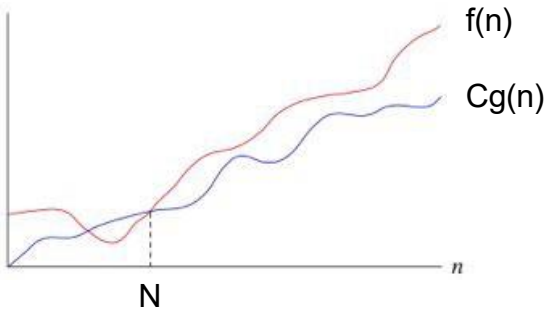
Let f(n) and g(n) be positive valued functions. We say

$$f(n) \in \Omega(g(n))$$

if there exists constants N > 0 and C > 0 such that for all n > N,

$$f(n) \geq Cg(n)$$

Understanding Big-Ω - Asymptotic Lower Bounds



Let f(n) and g(n) be positive valued functions. We say

$$f(n) \in \Omega(g(n))$$

if there exists constants N > 0 and C > 0 such that for all n > N,

$$f(n) \geq Cg(n)$$

Formally, Ω(g(n)) is a **set of functions** that contains all functions f(n) satisfying the above conditions. We often abuse notation and say

$$f(n) = \Omega(g(n))$$

Understanding Big-Ω - Asymptotic Lower Bounds

Quick test: If $f(n) = \Omega(n^2)$, then is $f(n) = \Omega(n)$?

Understanding Big-Ω - Asymptotic Lower Bounds

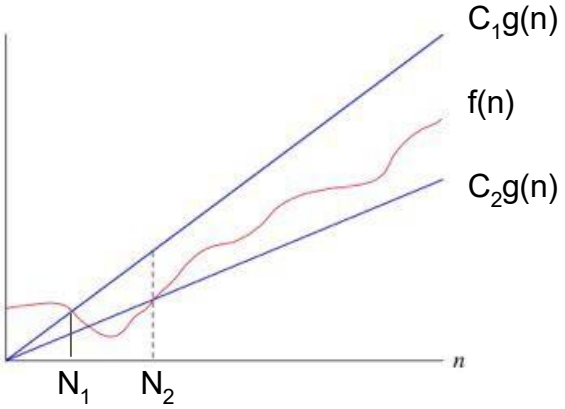
Quick test: If $f(n) = \Omega(n^2)$, then is $f(n) = \Omega(n)$?

Yes, because if we look at $\Omega(n^2)$ and $\Omega(n)$ as sets of functions, then we can clearly see that

$$\Omega(n^2) \subseteq \Omega(n)$$

Since $f(n) \in \Omega(n^2)$, then $f(n) \in \Omega(n)$.

Understanding Big-Θ - Asymptotic Tight Bounds



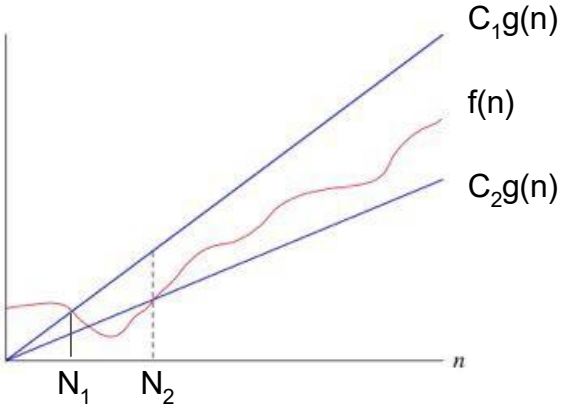
Let f(n) and g(n) be positive valued functions. We say

$$f(n) \in \Theta(g(n))$$

if

$$f(n) \in \Omega(g(n)) \text{ and } f(n) \in O(g(n))$$

Understanding Big-Θ - Asymptotic Tight Bounds



Let f(n) and g(n) be positive valued functions. We say

$$f(n) \in \Theta(g(n))$$

if

$$f(n) \in \Omega(g(n)) \text{ and } f(n) \in O(g(n))$$

Note that in the graph above, N_1 does not need to be the same as N_2, and C_1 does not need to be the same as C_2!

We are just looking at the intersection between the two sets, Ω(g(n)) and O(g(n)).

Appendix: Recursion Trees

Recursion Tree Drawing Technique

What happens during exams when you simply cannot figure out the complexity?

This is especially for questions involving recursion!

Recursion Tree Drawing Technique

What happens during exams when you simply cannot figure out the complexity?

This is especially for questions involving recursion!

Common question:

- How can I calculate the complexity of function g if g calls itself?
  I need the complexity of g to calculate the complexity of g!

Recursion Tree Drawing Technique

We can draw the recursion tree! (Note: this is not exactly taught in CS2040S, but it may make things a lot easier to see.)

4-step process:

1. Draw the recursion tree
2. Find the height of the recursion tree
3. Find the work done at every node in the tree
4. Find the work done at every level of the tree
    a. This is just the sum of work done by each node in each level
    b. See if you can spot some kind of pattern

Demo of the Technique

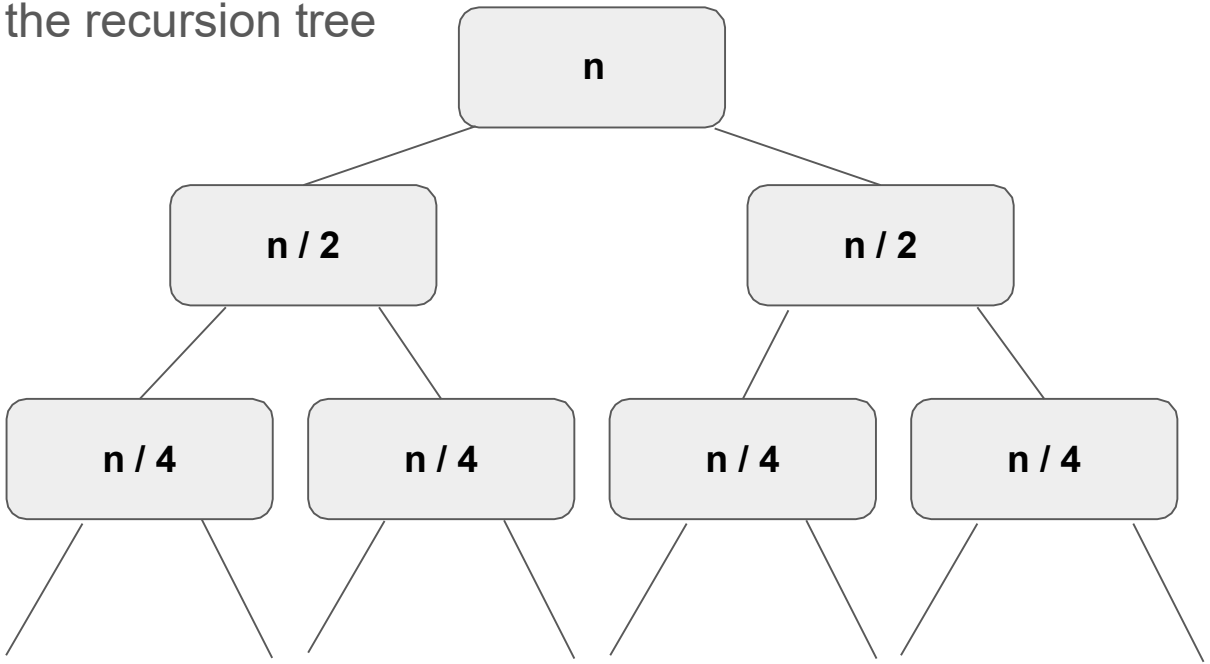
Let’s say we have the function below:

```
Foo(n):
    if n <= 1 then
        return
    else
        DoThetaOne(); // runs in Θ(1) time
        Foo(n/2);
        Foo(n/2);
    end
end
```

What’s the time complexity of this function?

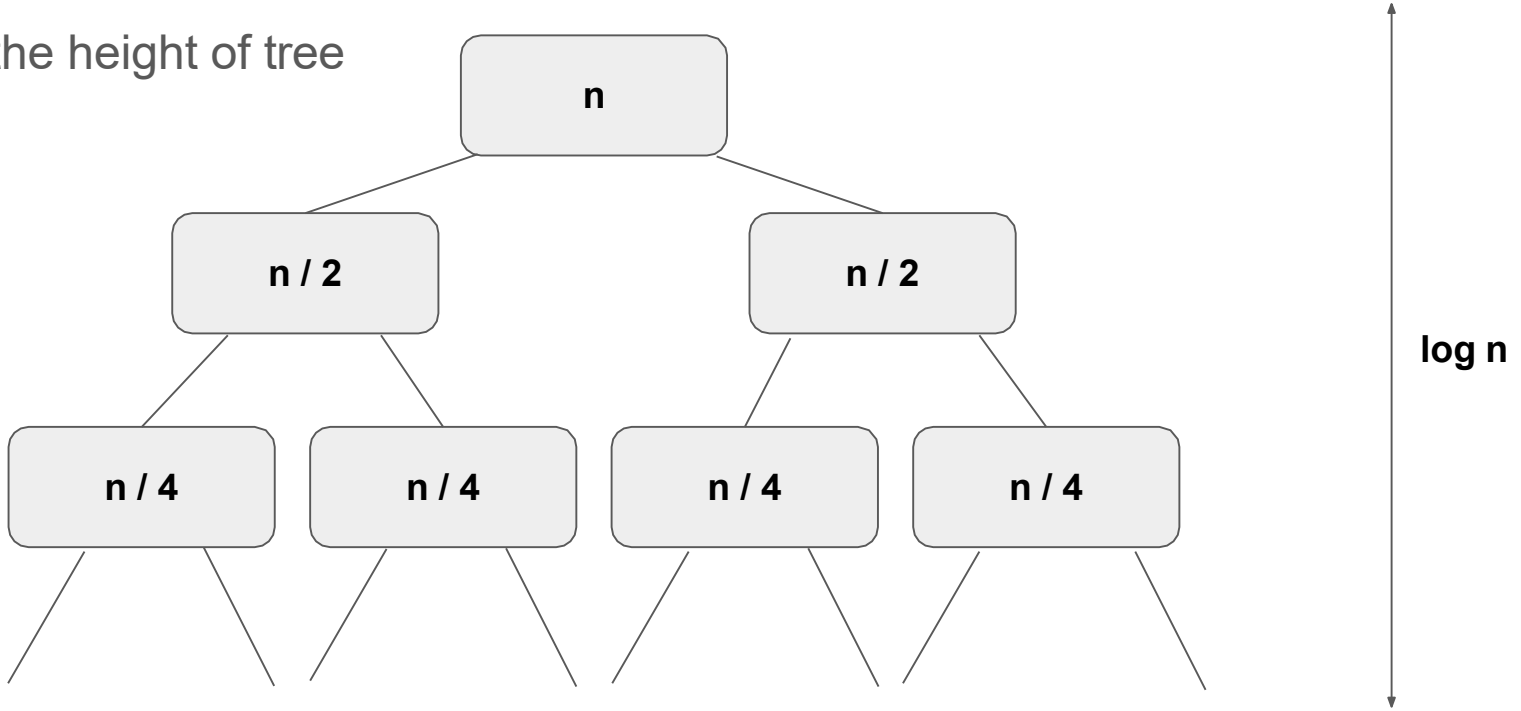
Demo of the Technique

Step 1: Draw the recursion tree



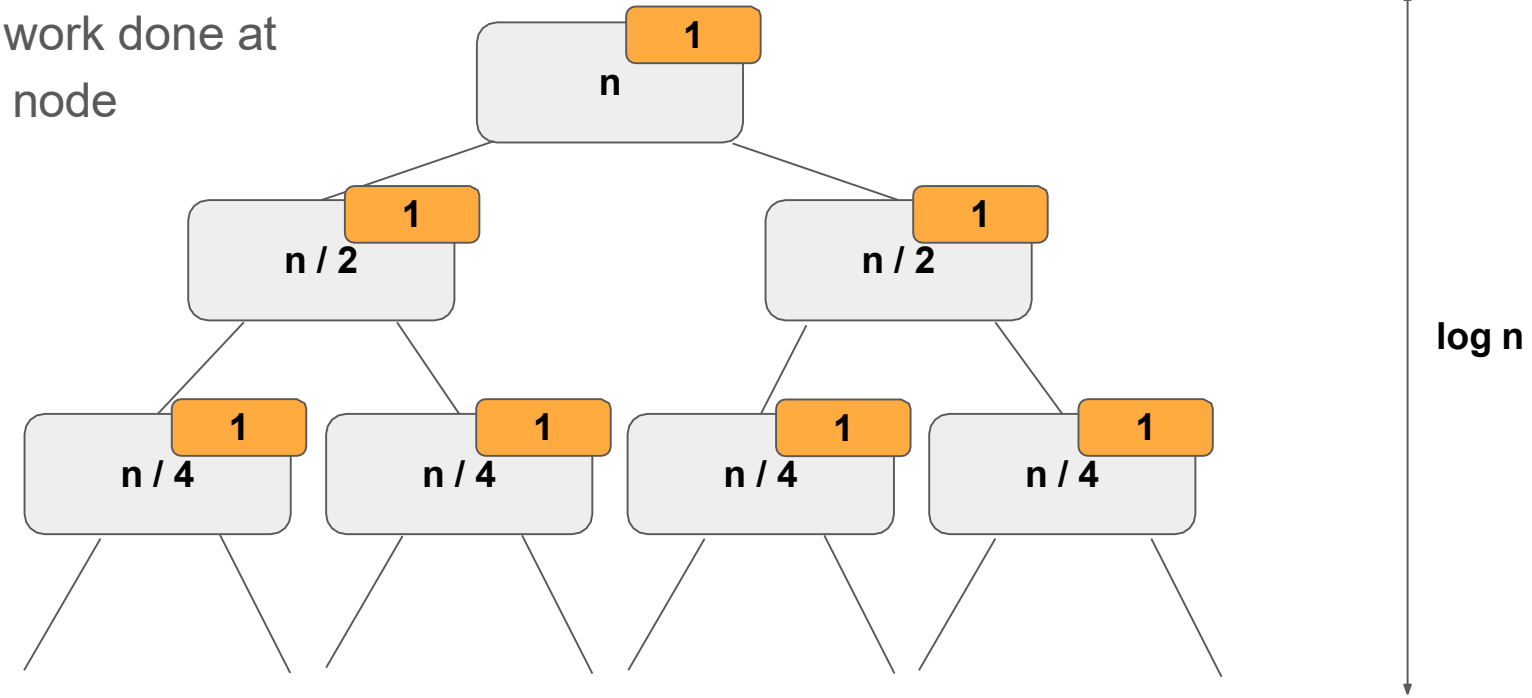
Demo of the Technique

Step 2: Find the height of tree



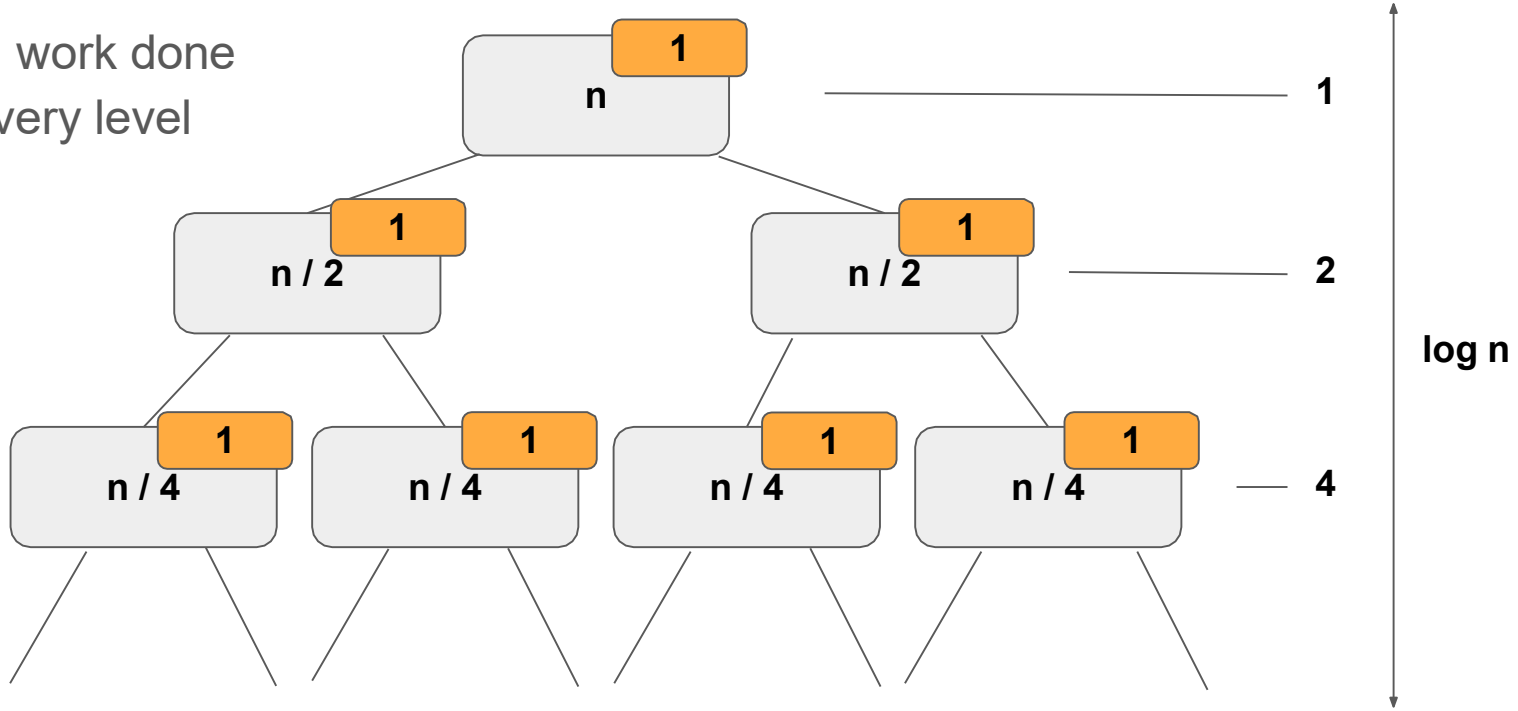
Demo of the Technique

Step 3: Find work done at each node



Demo of the Technique

Step 4a: Find work done at every level



Demo of the Technique

Step 4b: Finding a pattern

At 1st layer of recursion, 1 operation is performed.

At 2nd layer of recursion, 2 operations are performed.

At 3rd layer of recursion, 4 operations are performed.

…

At the kth layer of recursion, 2^{k-1} operations are performed.

Demo of the Technique

Step 4b: Finding a pattern

At 1st level of recursion, 1 operation is performed.

At 2nd level of recursion, 2 operations are performed.

At 3rd level of recursion, 4 operations are performed.

…

At the $\log n$ th level of recursion, $2^{\log n - 1}$ operations are performed.

We have $\log n$ levels of recursion.

Demo of the Technique

Step 4c: Summing Everything Up

$1 + 2 + 4 + \ldots + 2^{\log n - 1} \leq 2n$ (geometric series)

$= \Theta(n)$

More precisely, we can see that $1 + 2 + \ldots + 2^k = 2^{k+1} - 1$

Example: $1 + 2 + 4 + 8 = 2^0 + 2^1 + 2^2 + 2^3 = 15 = 16 - 1 = 2^4 - 1$

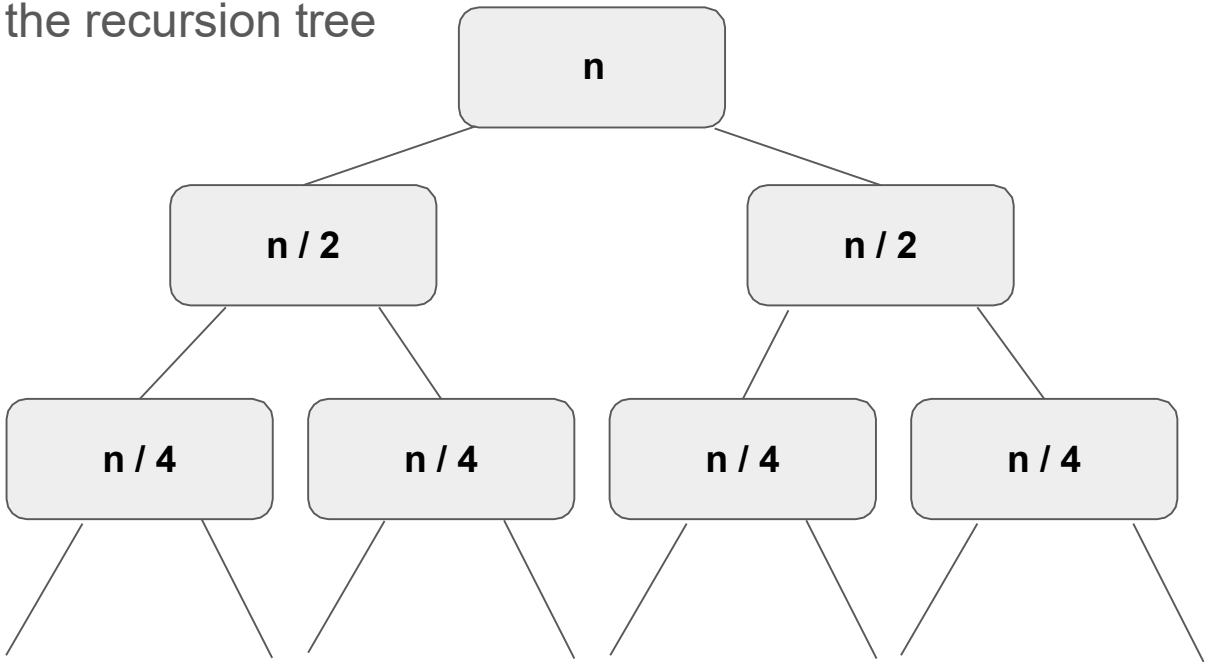
Thus, we have $1 + 2 + 4 + \ldots + 2^{\log n - 1} = 2^{\log n} - 1$

$= n - 1$

$= \Theta(n)$

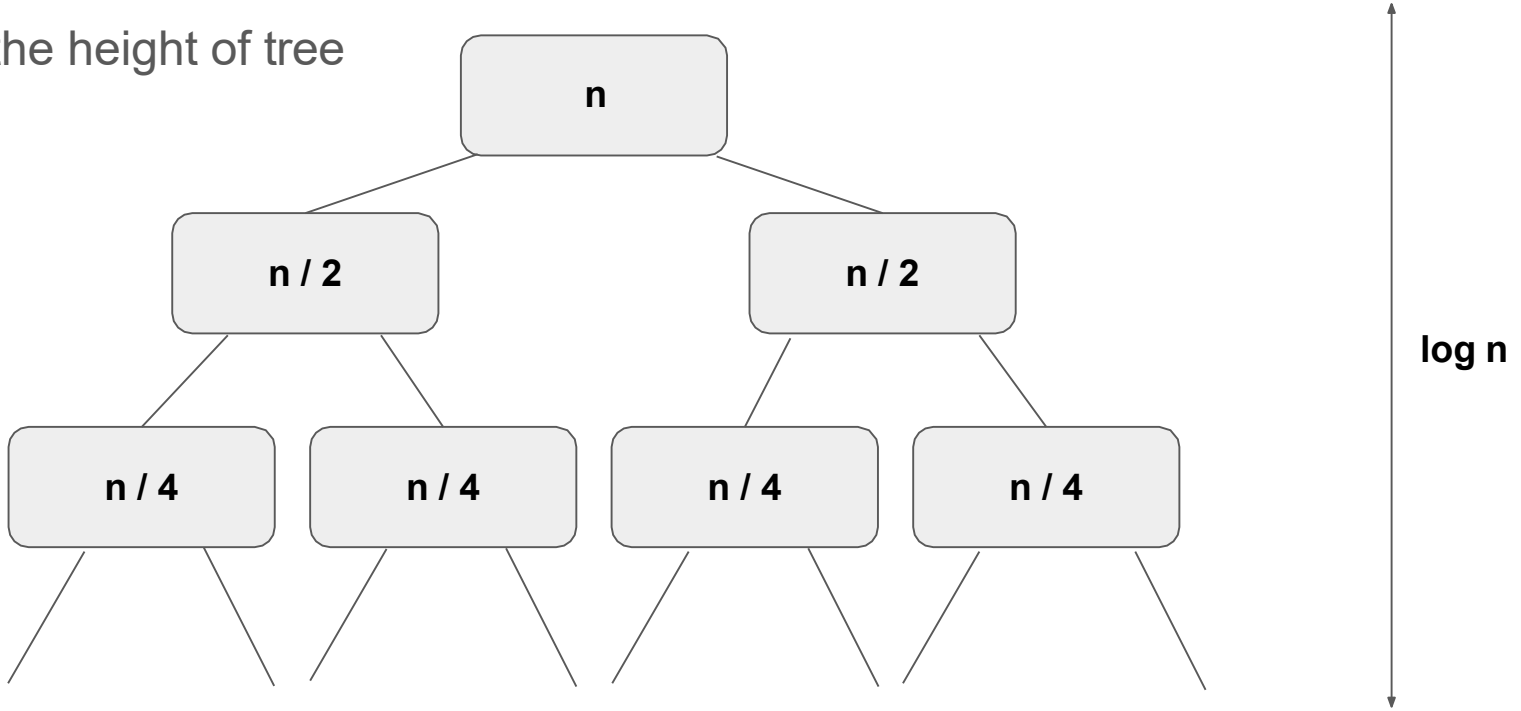
Using Recursion Tree for Merge Sort

Step 1: Draw the recursion tree



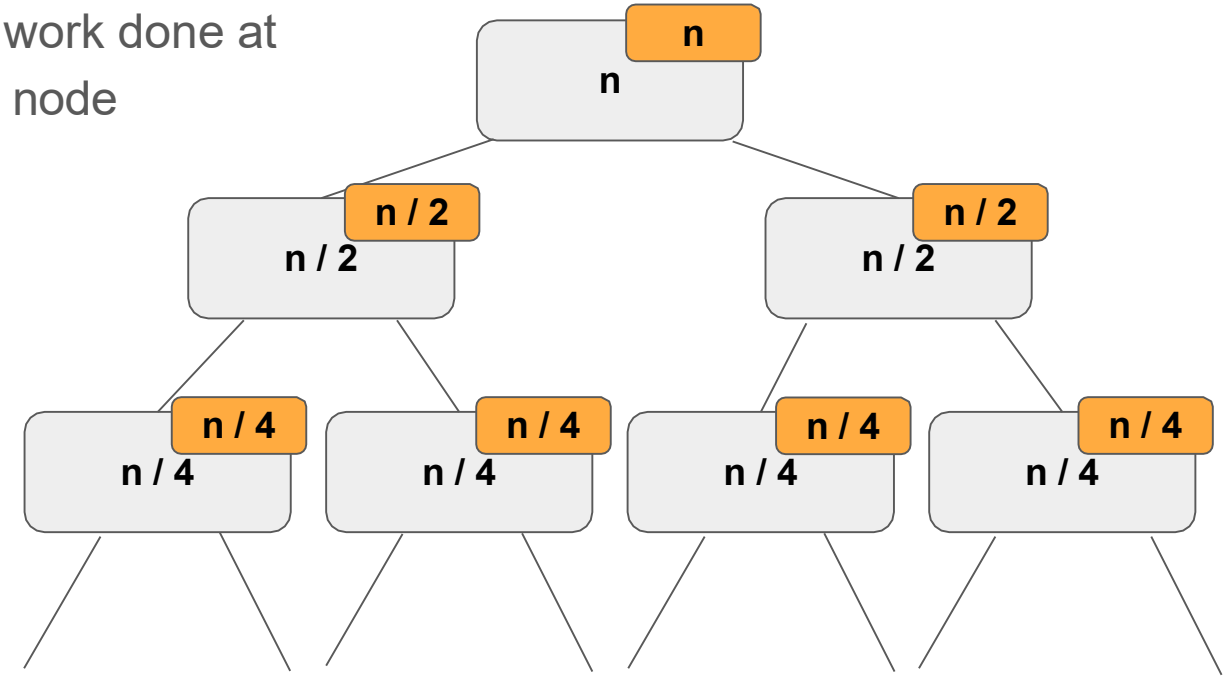
Using Recursion Tree for Merge Sort

Step 2: Find the height of tree



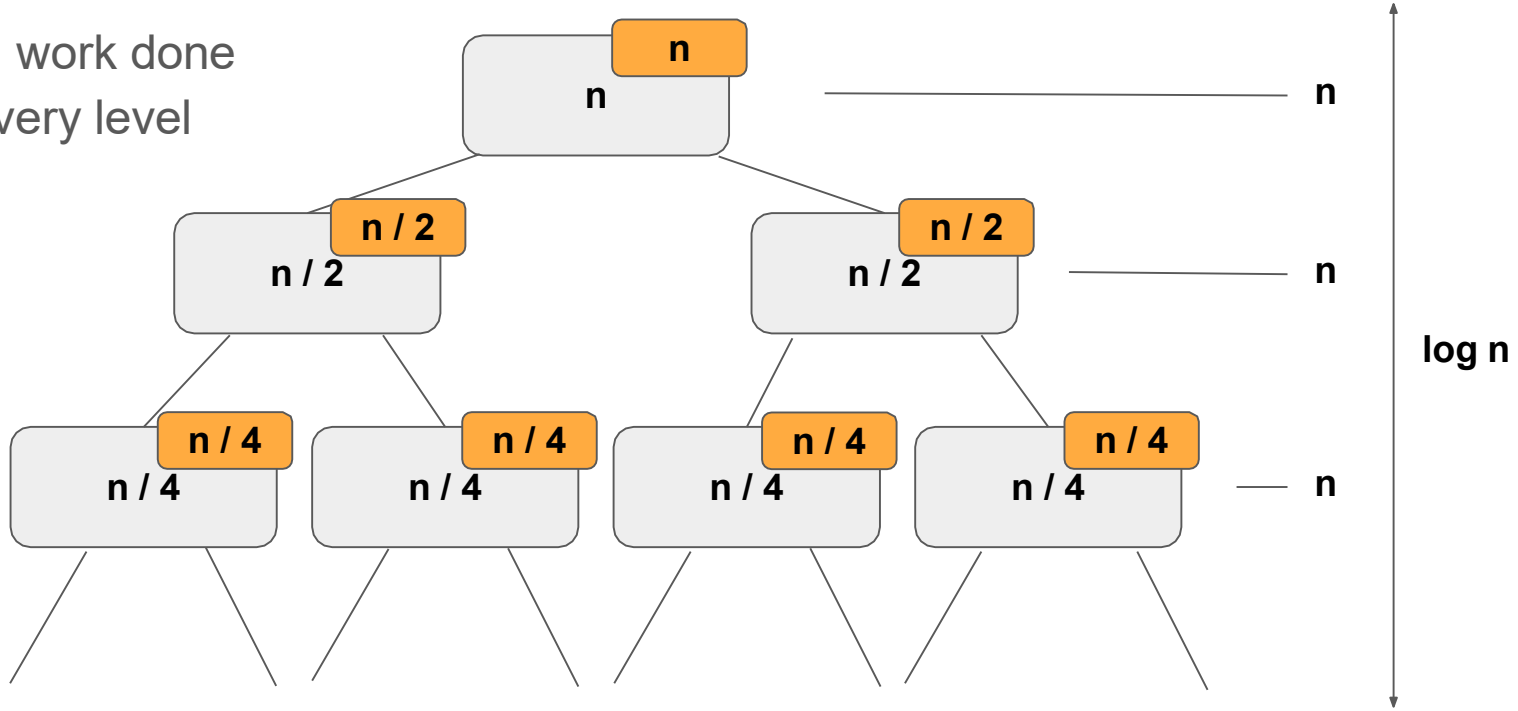
Using Recursion Tree for Merge Sort

Step 3: Find work done at each node



Using Recursion Tree for Merge Sort

Step 4a: Find work done at every level



Using Recursion Tree for Merge Sort

Step 4b: Finding a pattern

At 1st layer of recursion, n operations is performed.

At 2nd layer of recursion, n operations are performed.

At 3rd layer of recursion, n operations are performed.

…

At the log n th layer of recursion, n operations are performed.

Using Recursion Tree for Merge Sort

Step 4c: Summing Everything Up

$$\underbrace{n + n + n + \ldots + n}_{\log n \text{ copies}} = n \log n$$

$$= \Theta(n \log n)$$