

CS1101S - Programming Methodology I

Studio 3

Theodore Leebrant

Tutorial Group 8D

Admin Stuff

Attendance Taking

Make sure you have taken your temperature.

We will take photo when everyone is present.

Regarding SourceAcademy tasks

- By now, you should have done 4 paths:
 1. Elements of Programming (ungraded)
 2. Runes (140XP)
 3. Substitution model and recursion (140XP)
 4. Orders of Growth (160XP)
- These will be autograded at a later time.
- Paths are meant to assess your basic understanding of the topics taught in lectures and briefs. Please try to do them on time.

- Most, if not all of you, have completed the Rune Reading Mission.
- Some things to note:
 - Read the comments
 - Please make your code readable! This is for your future projects as well as my sanity in marking.
 - Read the Source styleguide. Some common mistakes include:

Common mistakes

- Off by 1 error - Pay attention to the constraints! Example
- Using unnecessary functions when better alternatives exist. Example
- Put your comments for a function outside the function, not inside the function body. Example
- For things other than functions, put the comments either on the line or after the line. Example
- Indentation style: try to stick to one tab (not two) - this one is more forgivable, but you might get some trouble when it got to longer code, because
- You need to keep to 80 characters per line. Example
- Also check the styleguide for indentation, as there are quite a bit on it.

- Learn to test your own solutions.
- **Always** test the given case >:(
- Consider base cases. $n = 0$? $n = 1$? Test them. (**Common mistake**)
- Consider extreme cases. $n = \text{big ass number}$? (A little harder for runes)
- Don't just test n Test other parameters as well. Asymmetric runes? Different colors?

Recap

Substitution Model

- Reason about programs
- Replace eligible sub-expression with result until you can't anymore (fully simplified/reduced) - some expressions are *irreducible*.
- By performing repeated reductions, we can simplify and find the result of any given statement.

Applicative Order Reduction

Works like your normal arithmetic: evaluate from the left, and the deepest expression.

- Evaluate arguments first.
- Then substitute function call(s) with body.

Example: This will be evaluated as:

Normal Order Reduction

Do not evaluate arguments unless absolutely needed.

- Substitute function call(s) with body.
- Then evaluate the arguments.

Normal Order Reduction (Example)

Example:

This program will be executed as follows:

Applicative vs Normal Order Reduction

Recursive Problem-solving

1. Base Case - the simplest problem you can think of
2. Think of one slightly simpler function call. Can you build the desired function call from this one?

Example: Tower of Hanoi

The thing with recursion is belief!

You need to assume that the smaller problem is correct.

Do not try to trace the recursion.

Recursive vs Iterative Processes

- Both involve calling the function inside the function body (i.e. recursion)
- A recursive process has **deferred operations**
- An iterative process does not

Example: <https://share.sourceacademy.nus.edu.sg/itervsrec>
(Use Stepper for visualisation)

Time Complexity

Why do we care?

- Need an abstract way to talk about resources consumed
- The same for every languages, architecture, CPU
- Some abstract measure of time taken for the program to run.
- How do we characterize it?
 - Number of operations performed.
 - Number of “simple” operations performed for some input size.
 - Simple operations:
 - All arithmetic e.g. $4 * 5$
 - Memory read and write e.g. `const a = 4;`
 - Conditionals e.g. `if (a === 4)`

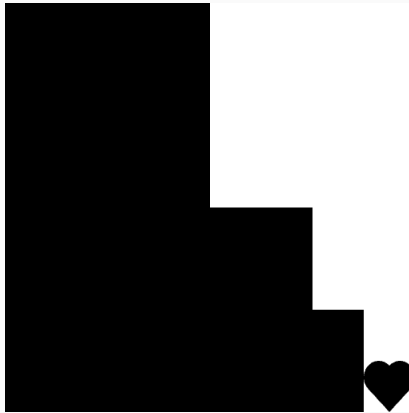
Why do we care?

- We don't want to run out of space :(
- Some abstract measure of space taken for the program to run is needed
- Characterized by maximum number of symbols created at one point in time (usually deferred operator)

Studio Sheet (and Photo Taking)

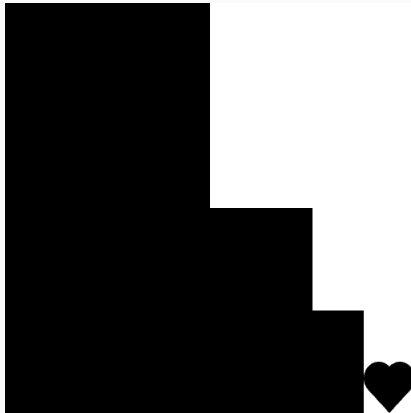
Question 1

Someone try to answer :D



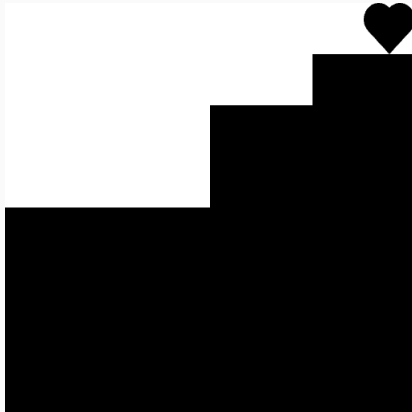
Question 1

Intuitive solution:



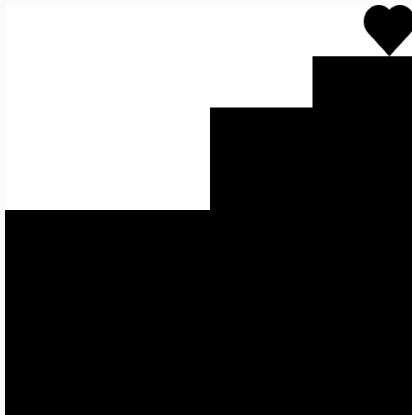
Question 2

The proper solution: [here](#)



Question 2

The intuitive solution: (on the board)



Question 3

Take some time to think about the question, and someone try to explain.

Question 4

```
function power(b, n) {  
    return n === 0 ? 1 : b * power(b, n - 1);  
}
```

- Iterative or recursive process? Can you write it in the other manner?
- Use the Θ notation to characterize the running time and space consumption of *power* as the argument n grows.

Question 4

Consider the following example: `power(2, 3)`

Question 5

Given that

$$b^n = \begin{cases} b^{n/2} b^{n/2} & \text{if } n \text{ is even} \\ b^{(n-1)/2} b^{(n-1)/2} & \text{if } n \text{ is odd} \end{cases}$$

- Implement a function *fast_power*(*b*, *n*) which computes b^n in $O(\log(n))$ time, where n is a natural number. [\[answer\]](#)
- Can you extend this to integer powers?
- Iterative or recursive process? Can you write it in the other manner?
- Use the Θ notation to characterize the running time and space consumption of *fast_power* as the argument n grows.

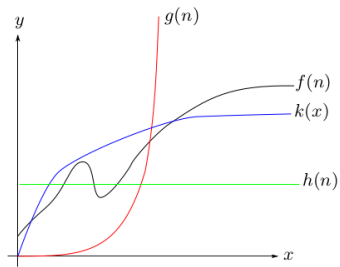
Additional Material

Big-O, Big-Ω, Big-Θ

The intuitive definition of Big-O, Big-Ω, Big-Θ are:
When the input is big enough,

$f(n) = O(g(n))$	f is bounded above by g
$f(n) = \Omega(g(n))$	f is bounded below by g
$f(n) = \Theta(g(n))$	f is bounded by g

Big-O, Big-Ω, Big-Θ



In this example, what is the relation between

- f and g
- f and h
- f and k

Big-O, Big-Ω, Big-Θ

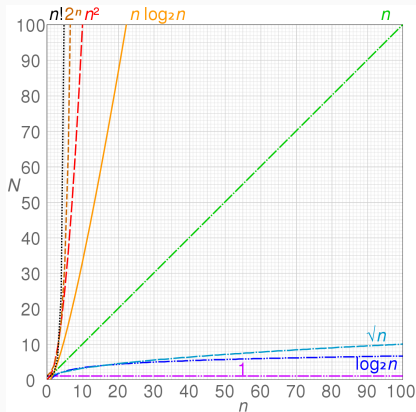


Figure 1: Graphs of commonly encountered time complexities

Optional content - try if you like math, at this juncture.

What are the complexities of:

- $4n^2 - n$
- $5n^2 + n$
- $\sqrt{n} + n$
- $3^n n^2$

Verify the following:

- $\log_5 n = \Theta(\log n)$
- $10n \log n = O(n^2)$
- $n^3 = O(2^n)$

Optional content - try if you want some extra practice, at this juncture.

What is the (space and time) complexity of:

Optional content - try if you want some extra practice, at this juncture.

What is the (space and time) complexity of:

Optional content - try if you want some extra practice, at this juncture.

What is the (space and time) complexity of:

Optional content - try if you want some extra practice, at this juncture.

Implement the following using both recursive and iterative processes:

- Factorial
- Fibonacci
- Power
- GCD
- LCM
- Coin-change problem (covered this week in lecture!)
- ...(continued next slide)

Optional content - try if you want some extra practice, at this juncture.

Implement the following using both recursive and iterative processes:

- Pascal triangle (covered this week in lecture!)
- Tower of Hanoi
- Permutations
- Combinations
- $\text{sqrt}(x)$ by Newton-Raphson method (or Newton's method)
- sine approximation (see the book)

Some cannot be implemented using iterative.

Analyze their space and time complexity.

Are there any closed form for these questions?