Function evaluation, recursion and complexity

Jia Xiaodong

Last revised August 22, 2021

Substitution Model

- What is the substitution model?

Substitution Model

- What is the substitution model?
  - Reasoning about programs

Substitution Model

- What is the substitution model?
  - Reasoning about programs
- What is the idea behind it?

Substitution Model

- What is the substitution model?
  - Reasoning about programs
- What is the idea behind it?
  - Certain expressions are *irreducible*.

Substitution Model

- What is the substitution model?
  - Reasoning about programs
- What is the idea behind it?
  - Certain expressions are *irreducible*.
  - Computation continues until we cannot proceed further, i.e. we get something that is irreducible.

Substitution Model

- What is the substitution model?
  - Reasoning about programs
- What is the idea behind it?
  - Certain expressions are *irreducible*.
  - Computation continues until we cannot proceed further, i.e. we get something that is irreducible.
  - By performing repeated reductions, we can simplify and find the result of any given statement.

Applicative Order Reduction

What does this do?

```
12345 % math_pow(10, math_floor(math_log10(12345)));
```

Applicative Order Reduction

What does this do?

```
12345 % math_pow(10, math_floor(math_log10(12345)));
```

Let's try:

Applicative Order Reduction

What does this do?

```
12345 % math_pow(10, math_floor(math_log10(12345)));
```

Let's try:

- `12345 % math_pow(10, math_floor(math_log10(12345)));`

Applicative Order Reduction

What does this do?

```
12345 % math_pow(10, math_floor(math_log10(12345)));
```

Let's try:

- `12345 % math_pow(10, math_floor(math_log10(12345)));`
- `12345 % math_pow(10, math_floor(4.09...));`

Applicative Order Reduction

What does this do?

```
12345 % math_pow(10, math_floor(math_log10(12345)));
```

Let's try:

- `12345 % math_pow(10, math_floor(math_log10(12345)));`
- `12345 % math_pow(10, math_floor(4.09...));`
- `12345 % math_pow(10, 4);`

Applicative Order Reduction

What does this do?

```
12345 % math_pow(10, math_floor(math_log10(12345)));
```

Let's try:

- `12345 % math_pow(10, math_floor(math_log10(12345)));`
- `12345 % math_pow(10, math_floor(4.09...));`
- `12345 % math_pow(10, 4);`
- `12345 % 10000;`

Applicative Order Reduction

What does this do?

```
12345 % math_pow(10, math_floor(math_log10(12345)));
```

Let's try:

- `12345 % math_pow(10, math_floor(math_log10(12345)));`
- `12345 % math_pow(10, math_floor(4.09...));`
- `12345 % math_pow(10, 4);`
- `12345 % 10000;`
- `2345;`

Normal Order Reduction

What does this do?

```
function sq(x) { return x * x };
function dist(x, y) { return math_sqrt(sq(x) + sq(y)) };
dist(1 + 5, 2 * 10);
```

Normal Order Reduction

What does this do?

```
function sq(x) { return x * x };
function dist(x, y) { return math_sqrt(sq(x) + sq(y)) };
dist(1 + 5, 2 * 10);
```

Let's try:

Normal Order Reduction

What does this do?

```
function sq(x) { return x * x };
function dist(x, y) { return math_sqrt(sq(x) + sq(y)) };
dist(1 + 5, 2 * 10);
```

Let's try:

- `dist(1 + 5, 2 * 10);`

Normal Order Reduction

What does this do?

```
function sq(x) { return x * x };
function dist(x, y) { return math_sqrt(sq(x) + sq(y)) };
dist(1 + 5, 2 * 10);
```

Let's try:

- `dist(1 + 5, 2 * 10);`
- `math_sqrt(sq(1 + 5) + sq(2 * 10))`

Normal Order Reduction

What does this do?

```
function sq(x) { return x * x };
function dist(x, y) { return math_sqrt(sq(x) + sq(y)) };
dist(1 + 5, 2 * 10);
```

Let's try:

- `dist(1 + 5, 2 * 10);`
- `math_sqrt(sq(1 + 5) + sq(2 * 10))`
- `math_sqrt((1 + 5) * (1 + 5) + (2 * 10) * (2 * 10))`

Normal Order Reduction

What does this do?

```
function sq(x) { return x * x };
function dist(x, y) { return math_sqrt(sq(x) + sq(y)) };
dist(1 + 5, 2 * 10);
```

Let's try:

- `dist(1 + 5, 2 * 10);`
- `math_sqrt(sq(1 + 5) + sq(2 * 10))`
- `math_sqrt((1 + 5) * (1 + 5) + (2 * 10) * (2 * 10))`
- `math_sqrt((6) * (1 + 5) + (2 * 10) * (2 * 10))`

Normal Order Reduction

What does this do?

```
function sq(x) { return x * x };
function dist(x, y) { return math_sqrt(sq(x) + sq(y)) };
dist(1 + 5, 2 * 10);
```

Let's try:

- `dist(1 + 5, 2 * 10);`
- `math_sqrt(sq(1 + 5) + sq(2 * 10))`
- `math_sqrt((1 + 5) * (1 + 5) + (2 * 10) * (2 * 10))`
- `math_sqrt((6) * (1 + 5) + (2 * 10) * (2 * 10))`
- `math_sqrt((6) * (6) + (2 * 10) * (2 * 10))`

Normal Order Reduction

What does this do?

```
function sq(x) { return x * x };
function dist(x, y) { return math_sqrt(sq(x) + sq(y)) };
dist(1 + 5, 2 * 10);
```

Let's try:

- `dist(1 + 5, 2 * 10);`
- `math_sqrt(sq(1 + 5) + sq(2 * 10))`
- `math_sqrt((1 + 5) * (1 + 5) + (2 * 10) * (2 * 10))`
- `math_sqrt((6) * (1 + 5) + (2 * 10) * (2 * 10))`
- `math_sqrt((6) * (6) + (2 * 10) * (2 * 10))`
- etc.

Exercise

Ex. 1.5

```
function p() {
    return p();
}

function test(x, y) {
    return x === 0 ? 0 : y;
}

test(0, p());
```

What does this evaluate to?

Recursion

From Wikipedia:

Recursion (adjective: recursive) occurs when a thing is defined in terms of itself or of its type.

Recursion

From Wikipedia:

Recursion (adjective: recursive) occurs when a thing is defined in terms of itself or of its type.

This gives rise to a way of solving certain problems. Certain problems exhibit the property of *optimal substructure*. This means that the method to solve a large problem is by breaking it up and solving the smaller sub-problems. Then you piece it back together.

Recursion

- Things we need:

Recursion

- Things we need:
  - The base case, or the trivial case.

Recursion

- Things we need:
  - The base case, or the trivial case.
  - A relationship between the large problem and the smaller sub problems.

Recursion

- Things we need:
  - The base case, or the trivial case.
  - A relationship between the large problem and the smaller sub problems.

Ex: Listing out $\mathbb{N}$

$$s_0 = 0 \quad s_n = s_{n-1} + 1$$

Recursion

- Things we need:
  - The base case, or the trivial case.
  - A relationship between the large problem and the smaller sub problems.

Ex: Listing out $\mathbb{N}$

$$s_0 = 0 \quad s_n = s_{n-1} + 1$$

Ex: Fib(n)

$$F_1 = 1, F_2 = 1 \quad F_n = F_{n-1} + F_{n-2}$$

Time and Space Complexities

- Why do we care?

Time and Space Complexities

- Why do we care?
  - We need an abstract way to talk about resources consumed.

Time and Space Complexities

- Why do we care?
  - We need an abstract way to talk about resources consumed.
  - We do not want to care about worldly problems like programming languages, computer architecture, CPU speed, etc.

Time and Space Complexities

- Why do we care?
  - We need an abstract way to talk about resources consumed.
  - We do not want to care about worldly problems like programming languages, computer architecture, CPU speed, etc.
  - We want to know how input affects it.

Time Complexity

- Some abstract measure of time taken for the program to run.

Time Complexity

- Some abstract measure of time taken for the program to run.
- How do we characterize it?

Time Complexity

- Some abstract measure of time taken for the program to run.
- How do we characterize it?
  - Number of operations performed.

Time Complexity

- 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.

Time Complexity

- 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:

Time Complexity

- 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`

Time Complexity

- 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;`

Time Complexity

- 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)`

Time Complexity

- 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)`
- An asymptotic bound on the number of primitive operations by *nice* functions [1].

[1] You can forget about this entirely, you will never meet a bad function in this course. This is however usually enforced because there exist pathological functions that really mess up complexity classes.

Space Complexity

- Some abstract measure of space taken for the program to run.

Space Complexity

- Some abstract measure of space taken for the program to run.
- How do we characterize it?

Space Complexity

- Some abstract measure of space taken for the program to run.
- How do we characterize it?
  - Number of symbols created.

Space Complexity

- Some abstract measure of space taken for the program to run.
- How do we characterize it?
  - Number of symbols created.
  - Maximum number of "simple" symbols created.

Space Complexity

- Some abstract measure of space taken for the program to run.
- How do we characterize it?
  - Number of symbols created.
  - Maximum number of “simple” symbols created.
- An asymptotic bound on the space required relative to input size.

Big O notation

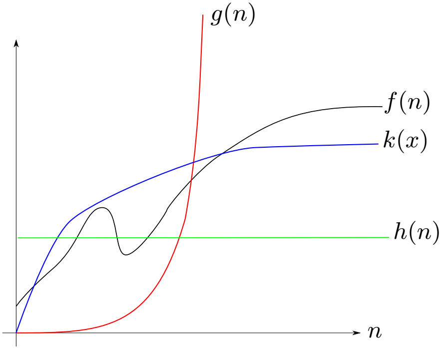
To accomplish these things we use the Big O asymptotic notation.

Name	Definition	Meaning
$f(n) = O(g(n))$	$\exists k > 0, \exists N, \forall n > N, f(n) \leq k \cdot g(n)$	f is bounded above by g.
$f(n) = \Omega(g(n))$	$\exists k > 0, \exists N, \forall n > N, f(n) \geq k \cdot g(n)$	f is bounded below by g.
$f(n) = \Theta(g(n))$	$\exists k_1, k_2 > 0, \exists N, \forall n > N,$ $k_1 \cdot g(n) \leq f(n) \leq k_2 \cdot g(n)$	f is bounded by g.

We can find some constant factor(s) such that regardless of how large the input gets (asymptotic) we can provide a bound on the function.

Big O notation

Graphical illustration



S3 Q1

```
function f1(rune_1, n, rune_2) {
    return n === 0
        ? rune_2
        : f1(rune_1, n - 1, beside(rune_1,stack(□, rune_2)));
}
```

Evaluate `f1(■, 3, ♡)` using the substitution model.

```
f1(■, 3, ♡)
```

S3 Q1

```
function f1(rune_1, n, rune_2) {
    return n === 0
        ? rune_2
        : f1(rune_1, n - 1, beside(rune_1,stack(□, rune_2)));
}
```

Evaluate `f1(■, 3, ♡)` using the substitution model.

```
f1(■, 3, ♡)

f1(■, 2, beside(■, stack(□, ♡)))
```

S3 Q1

```
function f1(rune_1, n, rune_2) {
    return n === 0
        ? rune_2
        : f1(rune_1, n - 1, beside(rune_1,stack(□, rune_2)));
}
```

Evaluate `f1(■, 3, ♡)` using the substitution model.

`f1(■, 3, ♡)`

`f1(■, 2, beside(■, stack(□, ♡)))`

f1(■, 2, beside(■, $\genfrac{}{}{0pt}{}{\square}{\heartsuit}$))

S3 Q1

```
function f1(rune_1, n, rune_2) {
    return n === 0
        ? rune_2
        : f1(rune_1, n - 1, beside(rune_1,stack(□, rune_2)));
}
```

Evaluate `f1(■, 3, ♡)` using the substitution model.

```
f1(■, 3, ♡)

f1(■, 2, beside(■, stack(□, ♡)))
```

`f1(■, 2, beside(■,`  `))`

`f1(■, 2,`  `)`

S3 Q1

```
function f1(rune_1, n, rune_2) {
    return n === 0
        ? rune_2
        : f1(rune_1, n - 1, beside(rune_1,stack(□, rune_2)));
}
```

Evaluate `f1(■, 3, ♡)` using the substitution model.

```
f1(■, 3, ♡)
```

```
f1(■, 2, beside(■, stack(□, ♡)))
```

`f1(■, 2, beside(■,`  `))`

`f1(■, 2,`  `)`

`f1(■, 1, beside(■, stack(□,`  `)`

S3 Q1

```
function f1(rune_1, n, rune_2) {
    return n === 0
        ? rune_2
        : f1(rune_1, n - 1, beside(rune_1,stack(□, rune_2)));
}
```

Evaluate `f1(■, 3, ♡)` using the substitution model.

```
f1(■, 3, ♡)
f1(■, 2, beside(■, stack(□, ♡)))
f1(■, 2, beside(■, 
```

```
))
f1(■, 2, 
```

```
)
f1(■, 1, beside(■, stack(□, 
```

```
)
f1(■, 1, 
```

```
)
```

S3 Q1

```
function f1(rune_1, n, rune_2) {
    return n === 0
        ? rune_2
        : f1(rune_1, n - 1, beside(rune_1,stack(□, rune_2)));
}
```

Evaluate `f1(■, 3, ♡)` using the substitution model.

`f1(■, 3, ♡)`

`f1(■, 2, beside(■, stack(□, ♡)))`

`f1(■, 2, beside(■,`  `))`

`f1(■, 2,`  `)`

`f1(■, 1, beside(■, stack(□,`  `)`

`f1(■, 1,`  `)`

`f1(■, 0, beside(■, stack(□,`  `)`

S3 Q1

```
function f1(rune_1, n, rune_2) {
    return n === 0
        ? rune_2
        : f1(rune_1, n - 1, beside(rune_1,stack(□, rune_2)));
}
```

Evaluate `f1(■, 3, ♡)` using the substitution model.

```
f1(■, 3, ♡)

f1(■, 2, beside(■, stack(□, ♡)))
```

f1(■, 2, beside(■, ))

f1(■, 2, )

f1(■, 1, beside(■, stack(□, )

f1(■, 1, )

f1(■, 0, beside(■, stack(□, )

f1(■, 0, )

S3 Q2

```
function f2(rune, n) {
    return n === 0
        ? rune
        : stack(beside(□, f2(rune, n - 1)), ■);
}
```

Evaluate `f2(♡, 3)` using the substitution model.

```
f2(♡, 0);
```

S3 Q2

```
function f2(rune, n) {
    return n === 0
        ? rune
        : stack(beside(□, f2(rune, n - 1)), ■);
}
```

Evaluate `f2(♡, 3)` using the substitution model.

```
f2(♡, 0);
```

♡

S3 Q2

```
function f2(rune, n) {
    return n === 0
        ? rune
        : stack(beside(□, f2(rune, n - 1)), ■);
}
```

Evaluate `f2(♡, 3)` using the substitution model.

```
f2(♡, 0);
```
♡

```
f2(♡, 1);
```

S3 Q2

```
function f2(rune, n) {
    return n === 0
        ? rune
        : stack(beside(□, f2(rune, n - 1)), ■);
}
```

Evaluate `f2(♡, 3)` using the substitution model.

```
f2(♡, 0);
♡

f2(♡, 1);
stack(beside(□, f2(♡, 0)), ■)
```

S3 Q2

```
function f2(rune, n) {
    return n === 0
        ? rune
        : stack(beside(□, f2(rune, n - 1)), ■);
}
```

Evaluate `f2(♡, 3)` using the substitution model.

```
f2(♡, 0);
♡

f2(♡, 1);
stack(beside(□, f2(♡, 0)), ■)
stack(beside(□, ♡), ■)
```

S3 Q2

```
function f2(rune, n) {
    return n === 0
        ? rune
        : stack(beside(□, f2(rune, n - 1)), ■);
}
```

Evaluate `f2(♡, 3)` using the substitution model.

```
f2(♡, 0);
♡
```

```
f2(♡, 1);
stack(beside(□, f2(♡, 0)), ■)
stack(beside(□, ♡), ■)
stack(□♡, ■)
```

S3 Q2

```
function f2(rune, n) {
    return n === 0
        ? rune
        : stack(beside(□, f2(rune, n - 1)), ■);
}
```

Evaluate `f2(♡, 3)` using the substitution model.

```
f2(♡, 0);
♡
```

```
f2(♡, 1);
stack(beside(□, f2(♡, 0)), ■)
stack(beside(□, ♡), ■)
stack(□♡, ■)
```

S3 Q2

```
function f2(rune, n) {
    return n === 0
        ? rune
        : stack(beside(□, f2(rune, n - 1)), ■);
}
```

Evaluate `f2(♡, 3)` using the substitution model.

```
f2(♡, 0);
```

♡

```
f2(♡, 1);
stack(beside(□, f2(♡, 0)), ■)
stack(beside(□, ♡), ■)
stack(□♡, ■)
```

```
f2(♡, 2);
```

S3 Q2

```
function f2(rune, n) {
    return n === 0
        ? rune
        : stack(beside(□, f2(rune, n - 1)), ■);
}
```

Evaluate `f2(♡, 3)` using the substitution model.

```
f2(♡, 0);
♡
```

```
f2(♡, 1);
stack(beside(□, f2(♡, 0)), ■)
stack(beside(□, ♡), ■)
stack(□♡, ■)
□♡
■
```

```
f2(♡, 2);
stack(beside(□, f2( □♡/■ , 1)), ■)
```

S3 Q2

```
function f2(rune, n) {
    return n === 0
        ? rune
        : stack(beside(□, f2(rune, n - 1)), ■);
}
```

Evaluate `f2(♡, 3)` using the substitution model.

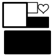
```
f2(♡, 0);
```

♡

```
f2(♡, 1);
stack(beside(□, f2(♡, 0)), ■)
stack(beside(□, ♡), ■)
stack(□♡, ■)
```

```
f2(♡, 2);
stack(beside(□, f2(
```

```
, 1)), ■)
```



S3 Q2

```
function f2(rune, n) {
    return n === 0
        ? rune
        : stack(beside(□, f2(rune, n - 1)), ■);
}
```

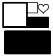
Evaluate `f2(♡, 3)` using the substitution model.

```
f2(♡, 0);
♡
```

```
f2(♡, 1);
stack(beside(□, f2(♡, 0)), ■)
stack(beside(□, ♡), ■)
stack(□♡, ■)
```

```
f2(♡, 2);
stack(beside(□, f2(
```

```
, 1)), ■)
```



```
f2(♡, 3);
```

S3 Q2

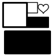
```
function f2(rune, n) {
    return n === 0
        ? rune
        : stack(beside(□, f2(rune, n - 1)), ■);
}
```

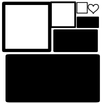
Evaluate `f2(♡, 3)` using the substitution model.

```
f2(♡, 0);
```
♡

```
f2(♡, 1);
stack(beside(□, f2(♡, 0)), ■)
stack(beside(□, ♡), ■)
stack(□♡, ■)
```

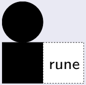

```
f2(♡, 2);
```
stack(beside(□, f2( , 1)), ■)



```
f2(♡, 3);
```


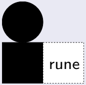
Q1

Write a function `moony_1(rune)` that outputs this:



Q1

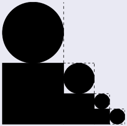
Write a function `moony_1(rune)` that outputs this:




```
function moony_1(rune) {
    return stack(beside(circle, blank),
        beside(square, rune));
}
```

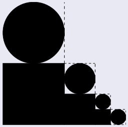
Q2

Write a function `moony_2` to recursively insert circles into the right place. Example output of `moony_2(4)`:



Q2

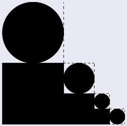
Write a function `moony_2` to recursively insert circles into the right place. Example output of `moony_2(4)`:



```
function moony_2(n) {
```

Q2

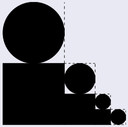
Write a function `moony_2` to recursively insert circles into the right place. Example output of `moony_2(4)`:



```
function moony_2(n) {
    return n === 1
        ? circle
```

Q2

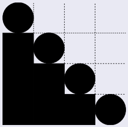
Write a function `moony_2` to recursively insert circles into the right place. Example output of `moony_2(4)`:



```
function moony_2(n) {
    return n === 1
        ? circle
        : moony_1(moony_2(n - 1));
}
```

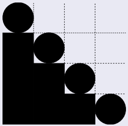
Q3

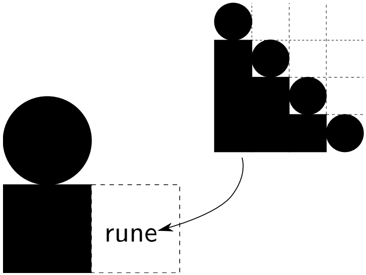
Now make the circles have the same diameters:



Q3

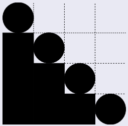
Now make the circles have the same diameters:

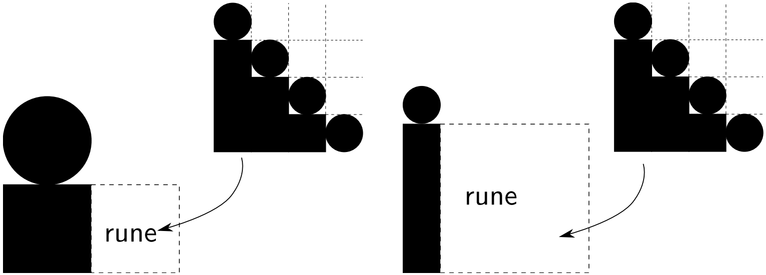




Q3

Now make the circles have the same diameters:





Q3

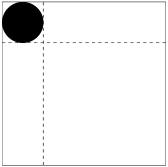
Cont.

```
function moony(n) {
    return n === 1
        ? circle
```

Q3

Cont.

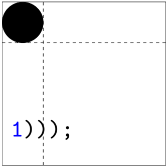
```
function moony(n) {
    return n === 1
        ? circle
        : stack_frac(1 / n,
            beside_frac(1/n, circle, blank),
```

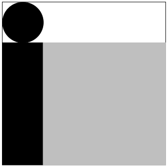


Q3

Cont.

```
function moony(n) {
    return n === 1
        ? circle
        : stack_frac(1 / n,
            beside_frac(1/n, circle, blank),
            beside_frac(1/n, square, moony(n - 1)));
}
```





Q4

Do your functions give rise to recursive or iterative processes? What is the time and space complexities of your `moony`?

Name	Process	Space	Time
`moony_1`	—	$\Theta(1)$	$\Theta(1)$
`moony_2`	Recursive	$\Theta(n)$	$\Theta(n)$
`moony`	Recursive	$\Theta(n)$	$\Theta(n)$

Q1

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

Q1

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

- `5 * expt(b, 4)`

Q1

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

- `5 * expt(b, 4)`
- `5 * 5 * expt(b, 3)`

Q1

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

- `5 * expt(b, 4)`
- `5 * 5 * expt(b, 3)`
- `5 * 5 * 5 * expt(b, 2)`

Q1

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

- `5 * expt(b, 4)`
- `5 * 5 * expt(b, 3)`
- `5 * 5 * 5 * expt(b, 2)`
- `5 * 5 * 5 * 5 * expt(b, 1)`

Q1

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

- `5 * expt(b, 4)`
- `5 * 5 * expt(b, 3)`
- `5 * 5 * 5 * expt(b, 2)`
- `5 * 5 * 5 * 5 * expt(b, 1)`
- ...

Q1

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

- `5 * expt(b, 4)`
- `5 * 5 * expt(b, 3)`
- `5 * 5 * 5 * expt(b, 2)`
- `5 * 5 * 5 * 5 * expt(b, 1)`
- ...

- Time: $\Theta(e)$
- Space: $\Theta(e)$

Q2

```
function fast_power(b, e) {
    return e === 0
        ? 1
        : is_even(e)
            ? fast_power(b * b, e / 2)
            : b * fast_power(b, e - 1);
}
```

Q2

```
function fast_power(b, e) {
    return e === 0
        ? 1
        : is_even(e)
            ? fast_power(b * b, e / 2)
            : b * fast_power(b, e - 1);
}
```

- `fast_power(3, 4)`

Q2

```
function fast_power(b, e) {
    return e === 0
        ? 1
        : is_even(e)
            ? fast_power(b * b, e / 2)
            : b * fast_power(b, e - 1);
}
```

- `fast_power(3, 4)`
- `fast_power(9, 2)`

Q2

```
function fast_power(b, e) {
    return e === 0
        ? 1
        : is_even(e)
            ? fast_power(b * b, e / 2)
            : b * fast_power(b, e - 1);
}
```

- `fast_power(3, 4)`
- `fast_power(9, 2)`
- `fast_power(81, 1)`

Q2

```
function fast_power(b, e) {
    return e === 0
        ? 1
        : is_even(e)
            ? fast_power(b * b, e / 2)
            : b * fast_power(b, e - 1);
}
```

- `fast_power(3, 4)`
- `fast_power(9, 2)`
- `fast_power(81, 1)`
- `81 * fast_power(81, 0)` ✓

Q2

```
function fast_power(b, e) {
    return e === 0
        ? 1
        : is_even(e)
            ? fast_power(b * b, e / 2)
            : b * fast_power(b, e - 1);
}
```

- `fast_power(3, 4)`
- `fast_power(9, 2)`
- `fast_power(81, 1)`
- `81 * fast_power(81, 0)` ✓

- Time: $\Theta(\log e)$
- Space: $\Theta(1)$

Q9

Cont.

```
    return e === 0
        ? 1
        : is_even(e)
            ? fast_power(b * b, e / 2)
            : b * fast_power(b, e - 1);
```

Q9

Cont.

```
    return e === 0
        ? 1
        : is_even(e)
            ? fast_power(b * b, e / 2)
            : b * fast_power(b, e - 1);
```

- `fast_power(3, 10)`

Q9

Cont.

```
    return e === 0
        ? 1
        : is_even(e)
            ? fast_power(b * b, e / 2)
            : b * fast_power(b, e - 1);
```

- `fast_power(3, 10)`
- `fast_power(9, 5)`

Q9

Cont.

```
return e === 0
    ? 1
    : is_even(e)
        ? fast_power(b * b, e / 2)
        : b * fast_power(b, e - 1);
```

- `fast_power(3, 10)`
- `fast_power(9, 5)`
- `9 * fast_power(9, 4)`

Q9

Cont.

```
    return e === 0
        ? 1
        : is_even(e)
            ? fast_power(b * b, e / 2)
            : b * fast_power(b, e - 1);
```

- `fast_power(3, 10)`
- `fast_power(9, 5)`
- `9 * fast_power(9, 4)`
- `9 * fast_power(81, 2)`

Q9

Cont.

```
return e === 0
    ? 1
    : is_even(e)
        ? fast_power(b * b, e / 2)
        : b * fast_power(b, e - 1);
```

- `fast_power(3, 10)`
- `fast_power(9, 5)`
- `9 * fast_power(9, 4)`
- `9 * fast_power(81, 2)`
- `9 * fast_power(6561, 1)`

Q9

Cont.

```
return e === 0
    ? 1
    : is_even(e)
        ? fast_power(b * b, e / 2)
        : b * fast_power(b, e - 1);
```

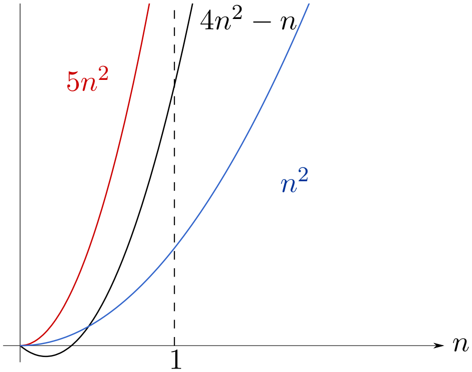
- `fast_power(3, 10)`
- `fast_power(9, 5)`
- `9 * fast_power(9, 4)`
- `9 * fast_power(81, 2)`
- `9 * fast_power(6561, 1)`
- `9 * 6561 * fast_power(6561, 0)` ✓

Q1

Show that $4n^2 - n = \Theta(n^2)$

$n_0 = 1, k_2 = 5, k_1 = 1.$

In fact one can show that for any polynomial $p_m(n)$ of degree m, $p_m(n) = \Theta(n^m)$.



Q2

Show that $\log_5(n) = \Theta(\ln n)$

Q2

Show that $\log_5(n) = \Theta(\ln n)$

$$\log_a n = \frac{\log_b n}{\log_b a}$$

Q3

$$10n \log n \stackrel{?}{=} O(n^2)$$

Q3

$$10n \log n \stackrel{?}{=} O(n^2)$$

$$10n \log n \stackrel{?}{\leq} 2n^2$$

$$\log n \leq n$$

Q4

$$n^3 \stackrel{?}{=} O(2^n)$$

Q4

$$n^3 \stackrel{?}{=} O(2^n)$$

$$n^3 \stackrel{?}{\leq} k \cdot 2^n$$

$$\log n^3 \stackrel{?}{\leq} \log k + \log 2^n$$

$$3 \log n \leq \log k + n \log 2$$

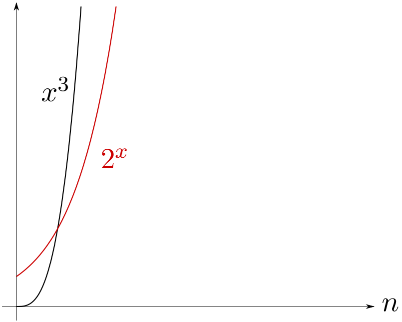
Q4

$$n^3 \stackrel{?}{=} O(2^n)$$

$$n^3 \stackrel{?}{\leq} k \cdot 2^n$$

$$\log n^3 \stackrel{?}{\leq} \log k + \log 2^n$$

$$3 \log n \leq \log k + n \log 2$$



Q5

a) $5n^2 + n = \Theta(?)$
b) $\sqrt{n} + n = \Theta(?)$
c) $3^n n^2 = \Theta(?)$

Q5

a) $5n^2 + n = \Theta(?)$
b) $\sqrt{n} + n = \Theta(?)$
c) $3^n n^2 = \Theta(?)$

a) $\Theta(n^2)$

Q5

a) $5n^2 + n = \Theta(?)$
b) $\sqrt{n} + n = \Theta(?)$
c) $3^n n^2 = \Theta(?)$

a) $\Theta(n^2)$
b) $\Theta(n)$

Q5

a) $5n^2 + n = \Theta(?)$
b) $\sqrt{n} + n = \Theta(?)$
c) $3^n n^2 = \Theta(?)$

a) $\Theta(n^2)$
b) $\Theta(n)$
c) $\Theta(3^n n^2)$

For the next few questions, give the space and time complexities of the functions presented.

Q6

```
function factorial(n) {
    return n === 1
        ? 1
        : n * factorial(n - 1);
}
```

Q6

```
function factorial(n) {
    return n === 1
        ? 1
        : n * factorial(n - 1);
}
```

- `5 * factorial(4)`

Q6

```
function factorial(n) {
    return n === 1
        ? 1
        : n * factorial(n - 1);
}
```

- `5 * factorial(4)`
- `5 * 4 * factorial(3)`

Q6

```
function factorial(n) {
    return n === 1
        ? 1
        : n * factorial(n - 1);
}
```

- `5 * factorial(4)`
- `5 * 4 * factorial(3)`
- `5 * 4 * 3 * factorial(2)`

Q6

```
function factorial(n) {
    return n === 1
        ? 1
        : n * factorial(n - 1);
}
```

- `5 * factorial(4)`
- `5 * 4 * factorial(3)`
- `5 * 4 * 3 * factorial(2)`
- `5 * 4 * 3 * 2 * factorial(1)`

Q6

```
function factorial(n) {
    return n === 1
        ? 1
        : n * factorial(n - 1);
}
```

- `5 * factorial(4)`
- `5 * 4 * factorial(3)`
- `5 * 4 * 3 * factorial(2)`
- `5 * 4 * 3 * 2 * factorial(1)`
- `5 * 4 * 3 * 2 * 1`

Q6

```
function factorial(n) {
    return n === 1
        ? 1
        : n * factorial(n - 1);
}
```

- `5 * factorial(4)`
- `5 * 4 * factorial(3)`
- `5 * 4 * 3 * factorial(2)`
- `5 * 4 * 3 * 2 * factorial(1)`
- `5 * 4 * 3 * 2 * 1`

- Time: $\Theta(n)$
- Space: $\Theta(n)$

Q7

Write `factorial` that gives rise to an iterative process.

Q7

Write `factorial` that gives rise to an iterative process.

```
function _(n, res) {
    return n === 1
        ? res
        : _(n - 1, n * res);
}

function factorial(n) {
    return _(n, 1);
}
```