W1 → exam (always)

Code → RR → time O(?)

Recurrence Relation

```
def func_one(n):
    if n == 0:
        return 0
    else:
        return func_one(n-1)
```

} Base

10, 100, 100

R {

① Base: $T(0) = a$

Recursive: $T(n) = T(n-1) \boxed{+b}$

② Telescoping #3 times

$T(n) = T(n-1) + b$

$T(n-1) = T(n-2) + b$

$T(n) = T(n-2) + b + b$

$T(n) = T(n-2) + 2b$

$T(n-2) = T(n-3) + b$

$T(n) = T(n-3) + 3b$

③ Find k

base

$T(n) = \boxed{T(n-k)} + kb$

④ Subbing in base case

$T(0) = a$

$n - k = 0$

$\boxed{k = n}$

$T(n-k) = T(0)$

$T(n) = T(n-k) + kb$

$T(n) = T(n-n) + nb$

$T(n) = T(0) + nb$

$T(n) = a + nb$

⑤ $T(n) = a + nb$

$O(n)$

```
def func_two(n):
    if n == 0:
        return 0
    else:
        return func_two(n//2) + 10
```

1

unsolvable

$T(0)$

$T(0) = T(\frac{n}{2^k})$

$\frac{n}{2^k} = 0$

$\boxed{n = 0}$

① Base: $T(1) = a$ ← $T(1)$

Recursive: $T(n) = T(\frac{n}{2}) \underline{+b}$

② $T(n) = T(\frac{n}{2}) + b$

$T(\frac{n}{2}) = T(\frac{\frac{n}{2}}{2}) + b$

$T(\frac{n}{2}) = T(\frac{n}{4}) + b$

$T(n) = T(\frac{n}{4}) + 2b$

$T(\frac{n}{4}) = T(\frac{\frac{n}{4}}{2}) + b$

$T(\frac{n}{4}) = T(\frac{n}{8}) + b$

$T(n) = T(\frac{n}{8}) + 3b$

③ $T(n) = T(\frac{n}{8}) + 3b$

$T(n) = T(\frac{n}{2^3}) + 3b$

$T(n) = \boxed{T(\frac{n}{2^k})} + kb$

④ $T(1) = T(\frac{n}{2^k})$

$\frac{n}{2^k} = 1$

$n = 2^k$

$k = \log_2 n$

$T(n) = T(\frac{n}{2^{\log_2 n}}) + b\log_2 n$

$T(n) = a + b\log_2 n$

⑤ $T(n) = \underbrace{a + b\log_2 n}$

$O(\log n)$

```
def fibonacci(n):
    if n == 0:
        return 0
    if n == 1:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)
```

← twice

① Base: $T(0) = a$

$T(1) = b$

Recursive: $T(n) = T(n-1) + T(n-2) + c$

② $T(n) = \underbrace{T(n-1) + T(n-2)} + c$

#

$T(n) < 2T(n-1) + c$

$T(n-1) < 2T(n-2) + c$

$T(n) < 2[2T(n-2) + c] + c$

$T(n) < 4T(n-2) + 2c + c$

$T(n-2) < 2T(n-3) + c$

$T(n) < 4[2T(n-3) + c] + 3c$

$T(n) < 8T(n-3) + \boxed{7c}$

③ $T(n) < 2^3 T(n-3) + (2^3 - 1)c$

$T(n) < 2^k T(n-k) + (2^k - 1)c$

④ $T(0) = T(n-k)$

$n - k = 0$

$k = n$

$T(n) < 2^n \boxed{T(n-n)} + (2^n - 1)c$

$T(n) < 2^n a + 2^n c - c$

⑤ $O(2^n + 2^n) \rightarrow O(2^n)$

```
def func_three(n, m):
    if n == 1:
        return m
    else:
        x = 3 * func_three(n//3, m)
```

1/100 ...

$\underbrace{3} \times \underbrace{\text{return value}} = 100$

$= 300$

① Base: $T(1) = a$

Recursive: $T(n) = 3\underbrace{T(\frac{n}{3})} + b$

call function 3 times

$T(n) = T(\frac{n}{3}) + b$

☆☆

Aux : exclusive of input

Space : inclusive of input

Auxiliary Space & Space Complexity

```
def aux_one(n):
    arr = [None]*n
    for i in range(n):
        arr[i] = i*2
    return sum(arr)
```

int ⊗

[0, 0, 0, 0, ... n] items O(n)

⊗

⊗

Aux : O(n)

Space : O(n)

```
def aux_two(arr):
    for i in range(len(arr)):
        arr[i] += 1
    return arr
```

array as input O(n)

⊗ O(1)

⊗

Aux : O(1) →
- exclude input
- didn't explicitly create extra space

Space : O(n) → overall space taken

```
def aux_three(arr):
    bucket = [0]*256
    for i in range(len(arr)):
        bucket[ord(arr[i])] += 1
    return bucket
```

O(n)

creating space , O(256) × O(n) → O(1)

O(1) ⊗

O(1) ⊗

Aux : O(1)
- despite explicitly creating space, it is not dependent on n
- Still constant

for i in range(1000):
↑ O(1)

Space : O(n)

arr ~ size(), 1, 100, 1000, 100000...

bucket: 256 256 256 256

```
def aux_four(n):
    matrix = [None]*n
    for i in range(n):
        matrix[i] = [None]*n
    return 1000
```

int, O(1)

O(n)

O(n)

O(1)

$[\ \underbrace{[\ ],\ [\ ],\ [\ ]}_{size = n}\ \ldots\ n\ ]$

$\boxed{n^2}$

Aux : $O(n^2)$

Space : $O(n^2)$

input, array is input

Selection Sort
Space Complexity: O(n)
Auxiliary Space Complexity: O(1) ← swap elements

[-2, 0, -3, -4, 1, -1] X create new list

[-4 | 0, -3, -2, 1, -1]

[-4, -3 | 0, -2, 1, -1]

[-4, -3, -2 | 0, 1, -1]

⋮

in-place algorithms

array or input

Merge Sort
Space Complexity: O(n)
Auxiliary Space Complexity: O(n) → O(n)

O(n^2)

[-4, -2, -3 | 0, -1, -5] O(n)

O(n) → [-4, -2, -3] [0, -1 | -5]

[-4, -2] [-3] [0, -1] [-5]

[-4] [-2] ↓ [0] [-1]

O(n) → [-4, -2] [-3] [-1, 0] [-5]

[-4, -3, -2] [-5, -1, 0]

[-5, -4, -3, -2, -1, 0]

Quicksort
Space Complexity: O(n)
Auxiliary Space Complexity: O(?)

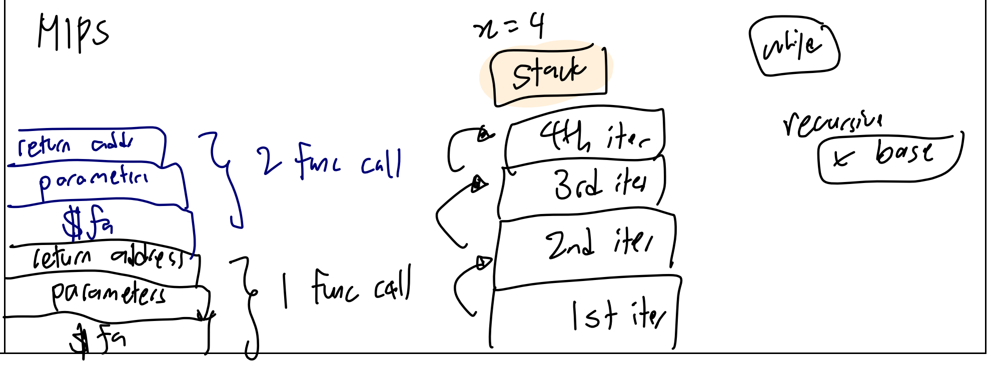
Next Week

Recursive Algorithms?

```
def rec(n): ← int
    if n == 1:
        return 0
    else:
        return rec(n-1)
```

Aux: ? always take up space.

aux space == to the depth of recursive stack


MIPS
return addr
parameters
$fa
return address
parameters
$fa
2 func call
1 func call
n = 4
stack
4th iter
3rd iter
2nd iter
1st iter
while
recursive
< base