Announcement:

HW1 online : Due. 02/17

Programming Assignment 2 online. : Due 02/22

Designing Recursive Algorithms.

Recursive Algorithms.
2 components.

① solves one part of the problem.
Ex. fact(0)

② Reduces the size of the problem.

Ex. $fact(n) = n * fact(n-1)$

① Base Case → should not call the algo.
② General Case. → does call the algo again (recursive)

Ex. $fact(n) = \frac{fact(n+1)}{n+1}$

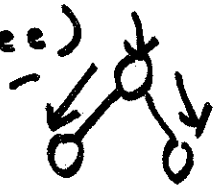
Rules for designing a recursive algo.
① Determine the base case.
② " the general case.
③ combine them into an algorithm.

Limitations: Extensive overhead. (time & memory)

- slower than iterative (loop) approach
- function calls. (how many?)
- how deep. (d).

Advantages. Elegant, Easy to read,

- some algo are inherently recursive. [O(log(N))]
- some DS " " " (tree)



fact(10) - recursive design is better.

fact(100) - iterative " " " . ✓

Ex. Print Reverse.

- Reads # data from keyboard (input)
- once input finishes, then prints the data in reverse order.

Q. Is DS or Algo. naturally suited for recursion?

list

O(N)

[O(N) function calls]

Q. Is the algo simpler to understand? Y.

Iterative approach is more suitable.

Fibonacci Numbers.

0 1 1 2 3 5 8 . . .

recursive.

$Fib(0) = 0$

$Fib(1) = 1$

$Fib(2) = 1 = Fib(0) + Fib(1)$

$Fib(n) = Fib(n-2) + Fib(n-1) \quad \forall\, n \geq 2.$

(Iterative)

0 1 1 2 3 5

second last   last

$fib(0) = 0$

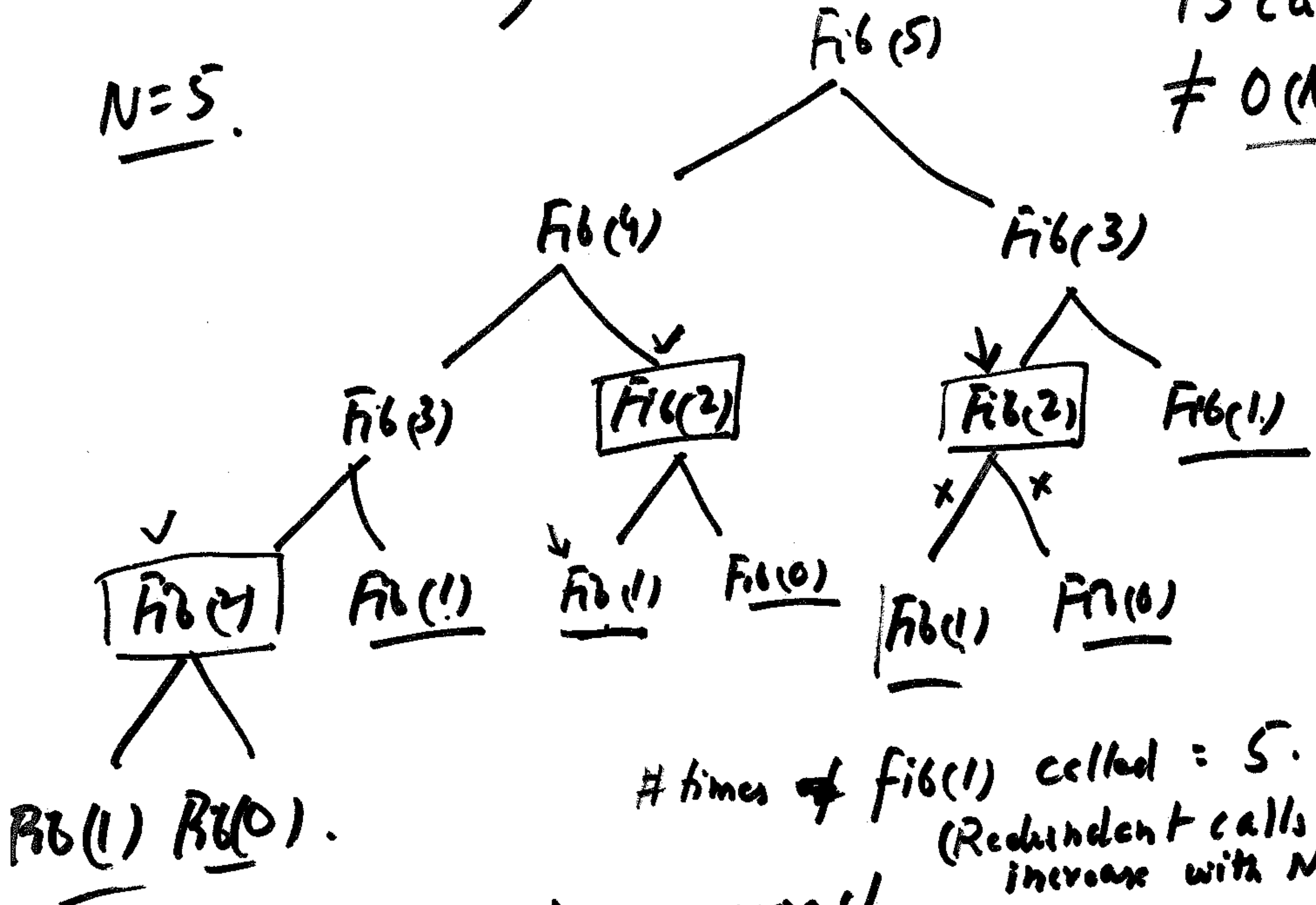
$fib(1) = 1$

```
while ( i )
{   result = second last + last;
    secondlast = last
}   last = result.
```

Q. How many function calls. are made to function fib. to compute fib (5). using recursive design.

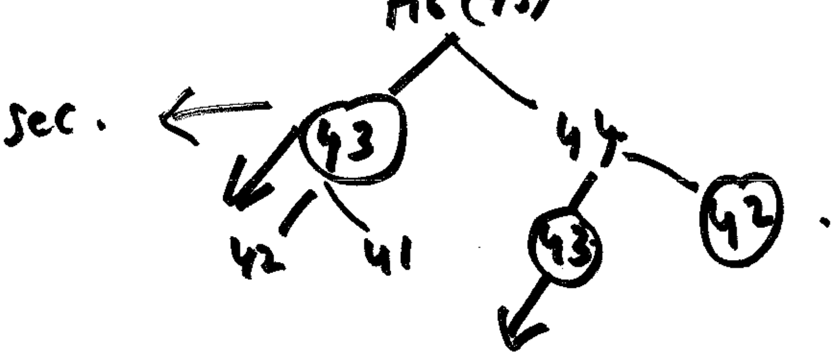
15 calls.
≠ O(N)

N=5.



times of fib(1) called : 5.
(Redundent calls increase with N)

Idea : keep recursive approach.
(good idea) However, store ~~all computed~~ Fib numbers. computed once.



original recursive approach

- # calls to fib. grows exponentially with N.

new recursive approach.

- # calls to ~~fig~~ fib is linear in N.

iterative approach is still better.

- for large n, recursive f^n is not efficient.
  (O(N) efficiency, O(N) f^n calls)

<u>1.13</u>. for (i=1; i < n; i *= 2) (Multiply loop)

<u>doIt</u> (..) → (Given) efficiency $O(n^2)$

Q. What is the efficiency of the above code segment?

for → multiple loop / divide loop] logarithmic loops. $O(\log N)$

Ans: $n^2 \cdot \log n$

2.1. ~~d~~

2.1. algorithm fun1 (x)

1. if (x < 5)

2. return (3 * x)

3 else.

4 return(2 * fun1(x-5) + 7)

6 end fun1

(A) fun1 (10) = ?

return (2 * fun1(5) + 7) → 21

↓ ↑ 7

fun1 (5) return (2 * fun1(0) + 7)

↓

fun1 (0) return 0

(B) fun1(2) = ? (6).

x
0
7
14
17
41
54.

(c) $fun1(11) = ?$ (33)