Recursion Vs Iteration

Exercise

- Write the pseudocode for a recursive function that will take a tree and will return true if the tree is full and false otherwise

Recall our process:

- Input/output?
- Examples:
  - Simplest example of calling the function
  - A more complex example
- Edit the template
  - Rename function & data
  - Edit the basecase
  - Deal with one data piece
  - Update to smaller problem for recursive call

HINT: - if you have more than one smaller problem, you can make a recursive call on each of them

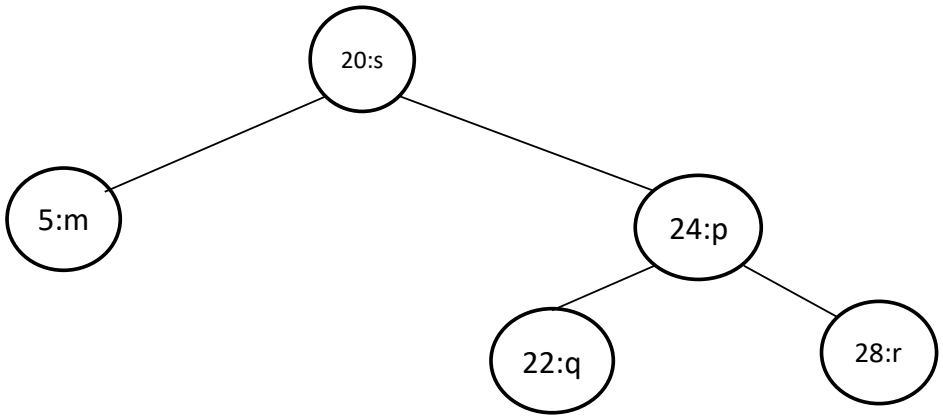
```
function(data)
    if (smallestPossibleProblem? data)
        the simple answer
        return ...
    else
        first part of data ...
        function(smallerProblem(data))
        return ...
```

Solution

```
isFull(tree)
    if (tree is empty)
        return true
    else
        if (height(tree's left subtree) NOT= height(tree's right subtree))
            return false

        else
            if (isFull(tree's left subtree) AND (isFull(tree's right subtree)
                return true
            else
                return false
```

Will this solution work for this tree?




```
isFull(tree)
    if(tree is empty)
        return true
    else
        if (tree's left subtree is empty AND tree's right subtree is not empty)
            return false
        if (tree's right subtree is empty AND tree's left subtree is not empty)
            return false
        else
            if (isFull(tree's left subtree) AND (isFull(tree's right subtree)
                return true
            else
                return false
```

exercise

- The Fibonacci Sequence is a series of numbers.
- The first two numbers in the sequence are 0 and 1
- The next number in the sequence is found by adding up the two numbers before it (0 + 1 = 1)
- The the first 7 values in the sequence are:
  0, 1, 1, 2, 3, 5, 8,…

- Write a method called fibonacci that takes an integer (n) and computes n^{th} number in the Fibonacci sequence
- For example,
  - fibonacci(0) would return 0
  - fibonacci(1) would return 1
  - fibonacci(2) would return 1
  - fibonacci(3) would return 2
  - fibonacci(4) would return 3
  - fibonacci(5) would return 5
  - fibonacci(6) would return 8

iterative solution

```
fibonacci(n)
    if(n==0)
        return 0
    else
        previous = 0
        current = 1
```

iterative solution

```
fibonacci(n)
    if(n==0)
        return 0
    else
        previous = 0
        current = 1
        for(count=1 to n(not inclusive))
            newcurrent = previous + current
            previous = current
            current = newcurrent
    return current
```

Recursive solution

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

What is the efficiency?

```
fibonacci(n)
    if(n==0)
        return 0
    else
        previous = 0
        current = 1
        for(count=1 to n(not inclusive))
            newcurrent = previous + current
            previous = current
            current = newcurrent
    return current
```

A. $O(1)$

B. $O(n)$

C. $O(\log n)$

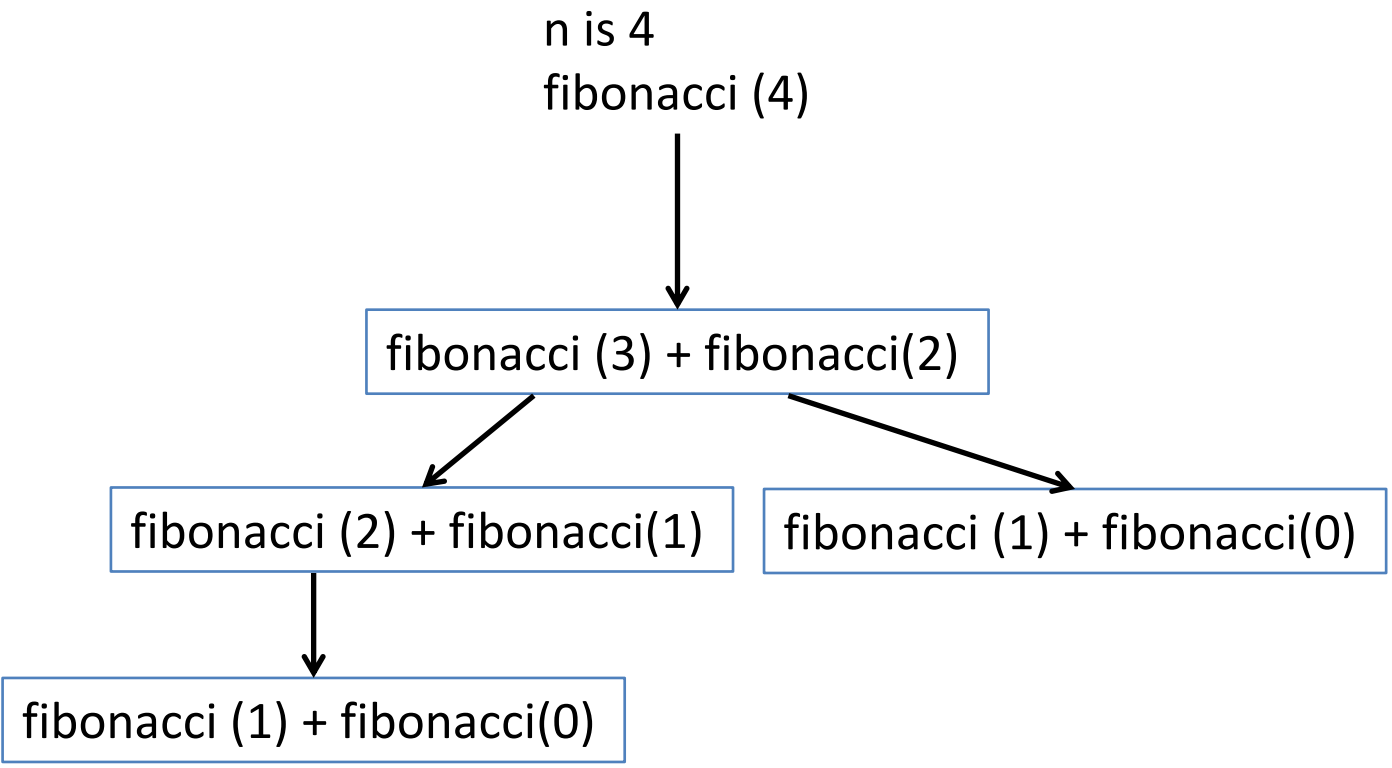
D. $O(n^2)$

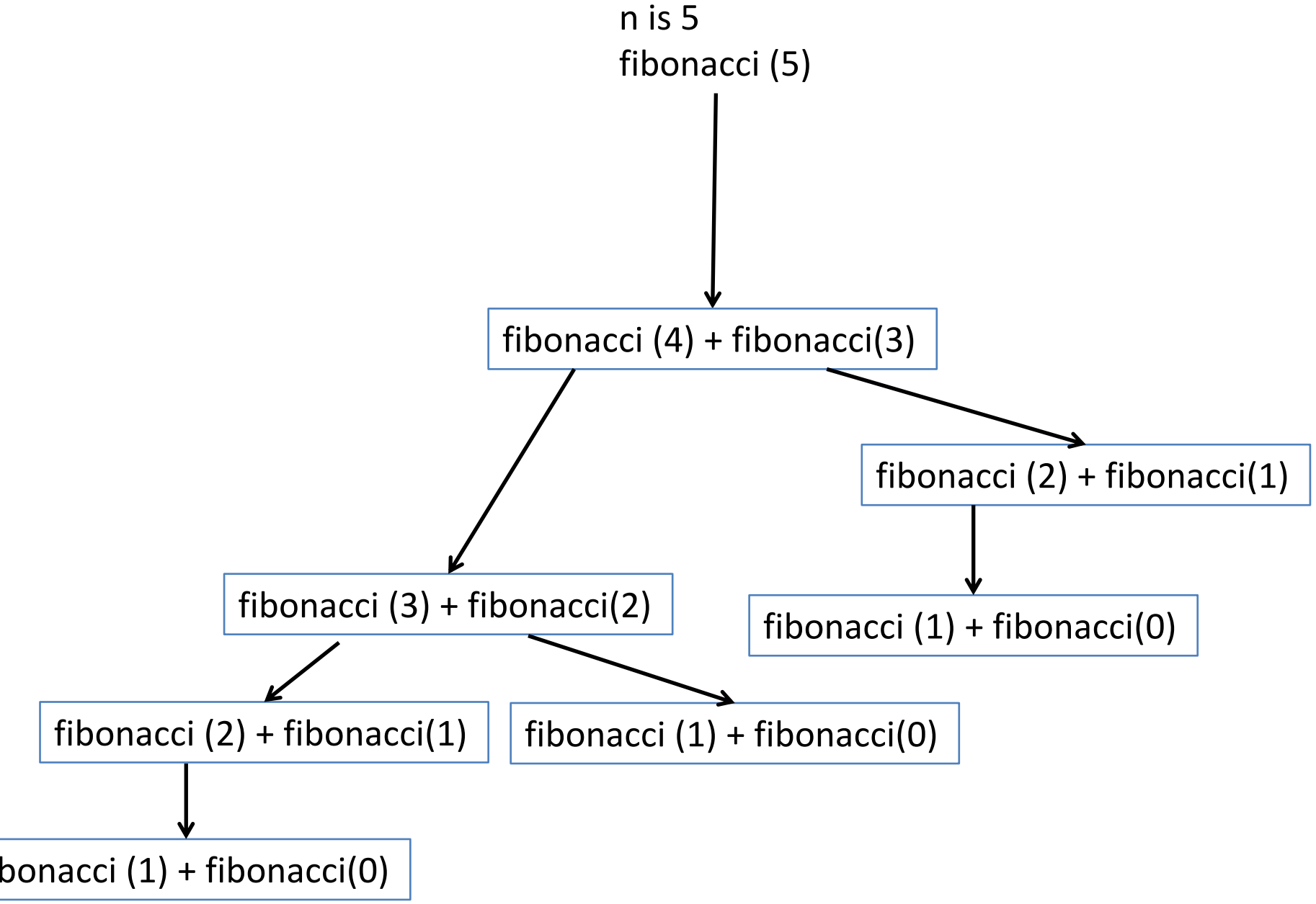
E. $O(2^n)$

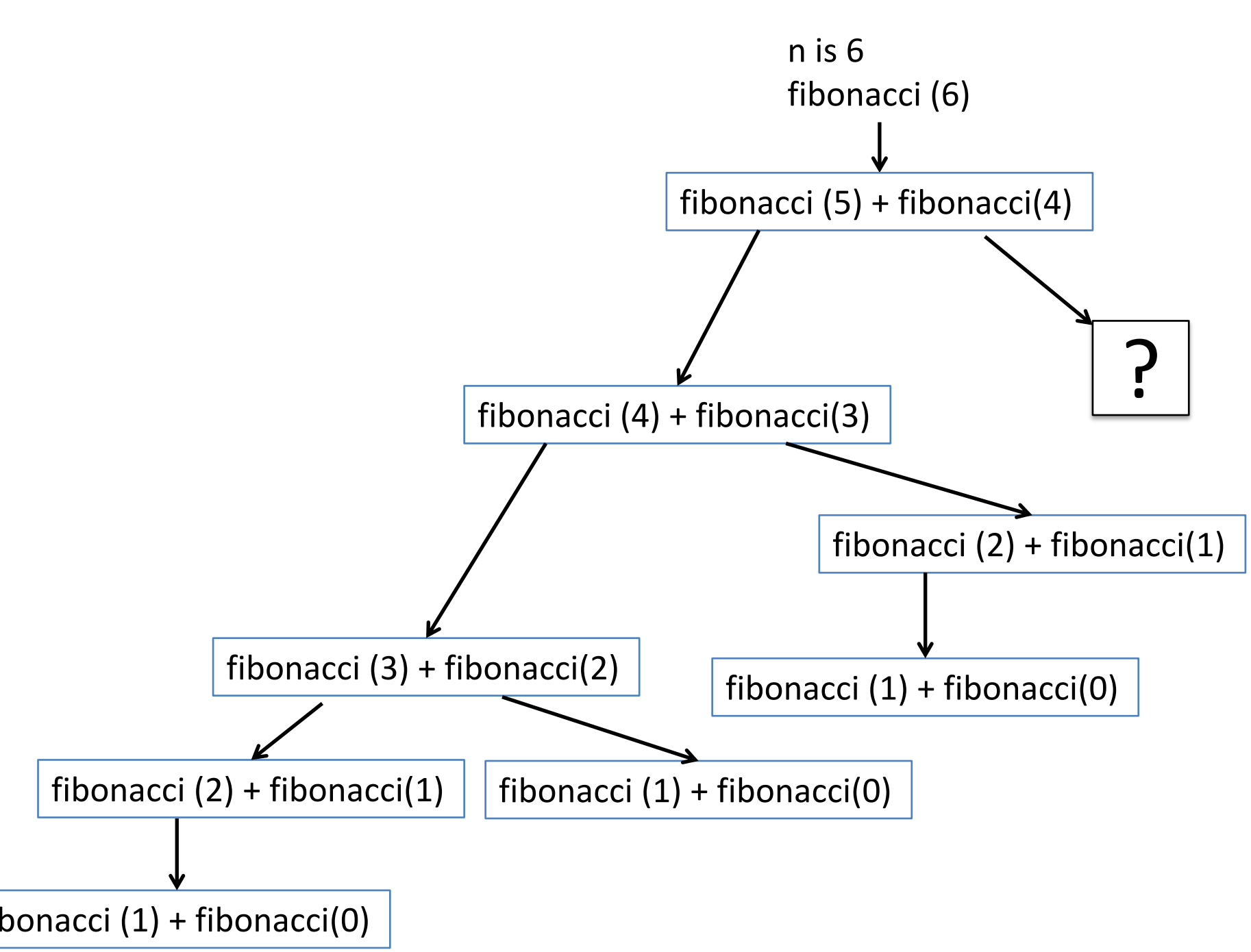
What is the efficiency?

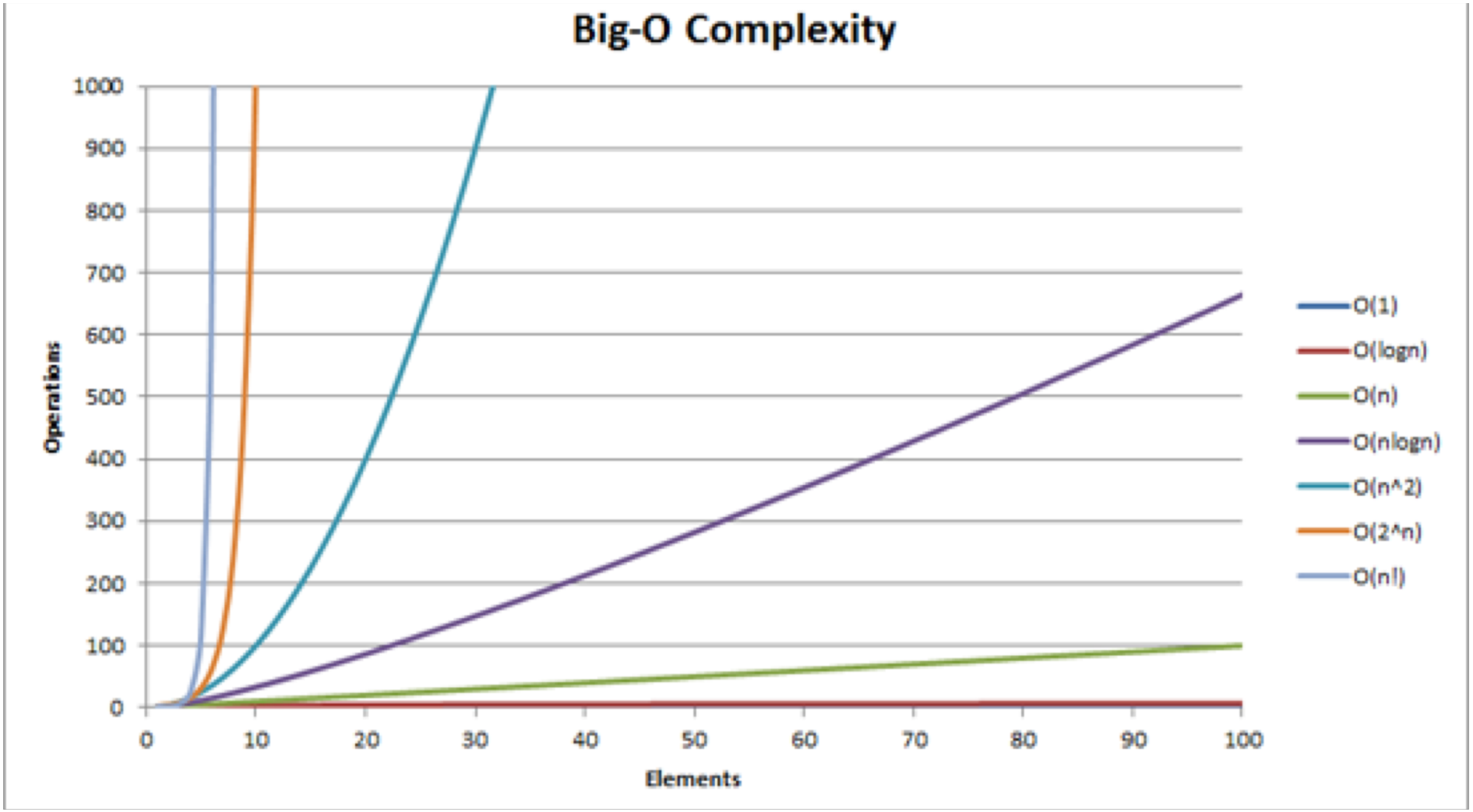
```
fibonacci(n)
    if(n==0 OR n==1)
        return n
    else
        return fibonacci(n-1) + fibonacci(n-2)
```

A. $O(1)$
B. $O(n)$
C. $O(\log n)$
D. $O(n^2)$
E. $O(2^n)$


n is 4
fibonacci (4)
fibonacci (3) + fibonacci(2)
fibonacci (2) + fibonacci(1)
fibonacci (1) + fibonacci(0)
fibonacci (1) + fibonacci(0)


n is 5
fibonacci (5)
fibonacci (4) + fibonacci(3)
fibonacci (2) + fibonacci(1)
fibonacci (3) + fibonacci(2)
fibonacci (1) + fibonacci(0)
fibonacci (2) + fibonacci(1)
fibonacci (1) + fibonacci(0)
bonacci (1) + fibonacci(0)


n is 6
fibonacci (6)
fibonacci (5) + fibonacci(4)
?
fibonacci (4) + fibonacci(3)
fibonacci (2) + fibonacci(1)
fibonacci (3) + fibonacci(2)
fibonacci (1) + fibonacci(0)
fibonacci (2) + fibonacci(1)
fibonacci (1) + fibonacci(0)
bonacci (1) + fibonacci(0)



http://www.daveperrett.com/articles/2010/12/07/comp-sci-101-big-o-notation/