PROGRAMMING ASSIGNMENT - 1
RUNNING TIME ANALYSIS - PART 2

Problem Solving with Computers-II

Pick matching cards

Clubs < diamonds < spades < hearts
ace < 2 < 3 …..< 10 < j < q < k

Alice

```
h 3
s 10
c a
c 3
s 5
h 10
d a
```

Bob

```
c 2
d a
h 10
c 3
d j
s 10
h a
```

Each player maintains an ordered hand of cards

How is this assignment different from lab4?

Alice	Bob
h 3	c 2
s 10	d a
c a	h 10
c 3	c 3
s 5	d j
h 10	s 10
d a	h a

Requirement: Store each hand in a BST

On Alice's turn

Clubs < diamonds < spades < hearts
ace < 2 < 3 …..< 10 < j < q < k

Alice

```
h 3
s 10
c a
c 3
s 5
h 10
d a
```

bob.contains("c a")? NO

Bob

```
c 2
d a
h 10
c 3
d j
s 10
h a
```

Alice iterates through her cards from smallest to largest until she finds a matching card in Bob's hand

On Alice's turn

Alice

```
h 3
s 10
c a
c 3
s 5
h 10
d a
```

bob.contains("c a")? NO
bob.contains("c 3")? Yes

Bob

```
c 2
d a
h 10
c 3
d j
s 10
h a
```

Alice iterates through her cards from smallest to largest until she finds a matching card in Bob's hand

On Alice's turn

Alice

```
h 3
s 10
c a

s 5
h 10
d a
```

Bob

```
c 2
d a
h 10

d j
s 10
h a
```

bob.contains("c a")? NO
bob.contains("c 3")? Yes
print message
alice. delete("c 3")
bob.delete("c 3")

Print message
Delete the card from both hands
Now its bob's turn

```
Alice picked matching card c 3
```

On Bob’s turn

Clubs < diamonds < spades < hearts
ace < 2 < 3 …..< 10 < j < q < k

Alice

```
h 3
s 10
c a

s 5
h 10
d a
```

Bob starts from largest card

alice.contains(“h 10”)? Yes
print message
bob. delete(“h 10”)
alice.delete(“h 10”)

Bob

```
c 2
d a
h 10

d j
s 10
h a
```

```
Alice picked matching card c 3
```

On Bob's turn

Alice

```
h 3
s 10
c a

s 5

d a
```

Repeat the same process

alice.contains("h 10")? Yes
print message
bob. delete("h 10")
alice.delete("h 10")

Bob

```
c 2
d a


d j
s 10
h a
```

```
Alice picked matching card c 3
Bob picked matching card h 10
```

Alice's turn

Clubs < diamonds < spades < hearts
ace < 2 < 3 …..< 10 < j < q < k

Alice

```
h 3
s 10
c a

s 5

d a
```

bob.contains("c a")? NO
bob.contains("d a")? Yes
print message

Bob

```
c 2
d a


d j
s 10
h a
```

```
Alice picked matching card c 3
Bob picked matching card h 10
Alice picked matching card d a
```

Alice's turn

Clubs < diamonds < spades < hearts
ace < 2 < 3 …..< 10 < j < q < k

Alice

```
h 3
s 10
c a

s 5
```

bob.contains("c a")? NO
bob.contains("d a")? Yes
print message
alice. delete("d a")
bob.delete("d a")

Bob

```
c 2



d j
s 10
h a
```

```
Alice picked matching card c 3
Bob picked matching card h 10
Alice picked matching card d a
```

Bob's turn

Clubs < diamonds < spades < hearts
ace < 2 < 3 …..< 10 < j < q < k

Alice

```
h 3
s 10
c a

s 5
```

What card should Bob check for in Alice's hand?

alice.contains(____)?

Bob

```
c 2

d j
s 10
h a
```

```
Alice picked matching card c 3
Bob picked matching card h 10
Alice picked matching card d a
```

Bob's turn

Clubs < diamonds < spades < hearts
ace < 2 < 3 …..< 10 < j < q < k

Alice

```
h 3
s 10
c a

s 5
```

```
Alice picked matching card c 3
Bob picked matching card h 10
Alice picked matching card d a
Bob picked matching card s 10
```

Bob

```
c 2



d j
s 10
h a
```

Should Alice take another turn? Yes / No

Alice

h 3

c a

s 5

Bob

c 2

d j

h a

```
Alice picked matching card c 3
Bob picked matching card h 10
Alice picked matching card d a
Bob picked matching card s 10
```

What is the condition to end?

Clubs < diamonds < spades < hearts
ace < 2 < 3 …..< 10 < j < q < k

Alice

h 3

c a

s 5

A. Player has no cards left
B. Player iterated through their cards and found no matching card
C. A or B
D. Something else

Bob

c 2

d j

h a

End game condition

Clubs < diamonds < spades < hearts
ace < 2 < 3 …..< 10 < j < q < k

Alice

h 3

c a

s 5

Bob

c 2

d j

h a

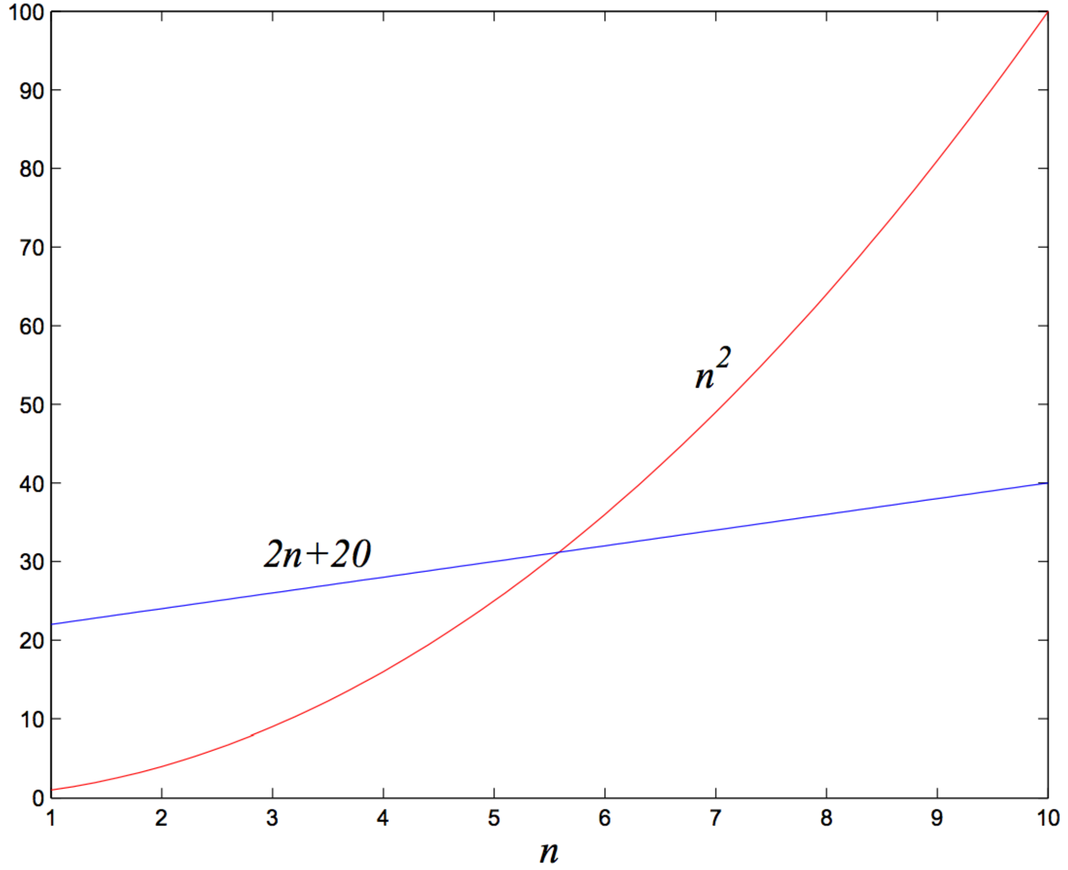
```
Alice picked matching card c 3
Bob picked matching card h 10
Alice picked matching card d a
Bob picked matching card s 10

Alice's cards:
c a
s 5
h 3

Bob's cards:
c 2
d j
h a
```

Definition of Big-O

f(n) and g(n) map positive integer inputs to positive reals.

We say f = O(g) if there is a constant c > 0 and k>0 such that
f(n) ≤ c · g(n) for all n >= k.

f = O(g)
means that "f grows no faster than g"

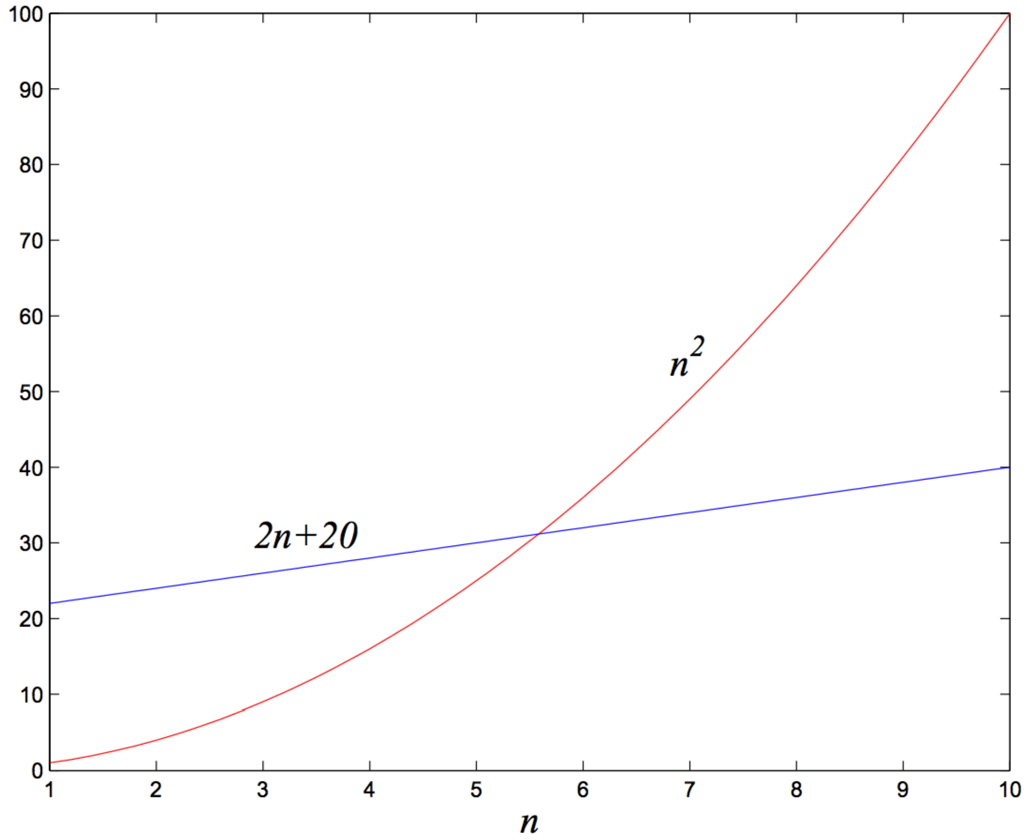


Big-Omega

- f(n) and g(n) map positive integer inputs to positive reals.

We say $f = \Omega(g)$ if there are constants $c > 0$, $k>0$ such that
$c \cdot g(n) \leq f(n)$ for $n >= k$

$f = \Omega(g)$
means that "f grows at least as fast as g"

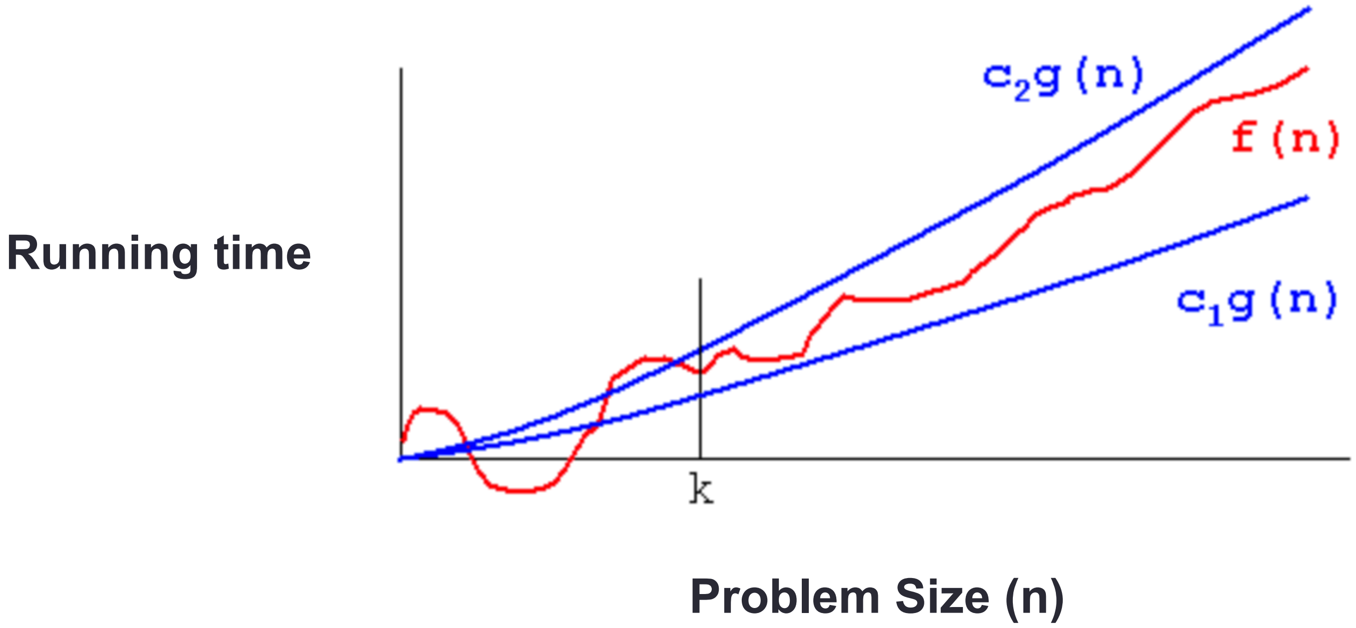


Big-Theta

- f(n) and g(n) map positive integer inputs to positive reals.

We say $f = \Theta(g)$ if there are constants c_1, c_2, k such that

$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$, for n >=k



Best case and worst case analysis

What is the Big-O running time of search in a sorted array of size n?

…using linear search?

…using binary search?

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Worst case analysis of binary search

```
bool binarySearch(int arr[], int element, int n){
//Precondition: input array arr is sorted in ascending order
  int begin = 0;
  int end = n-1;
  int mid;
  while (begin <=  end){
    mid = (end + begin)/2;
    if(arr[mid]==element){
      return true;
    }else if (arr[mid]< element){
      begin = mid + 1;
    }else{
      end = mid - 1;

    }
  }
  return false;
}
```

Best case and worst case : sorted array

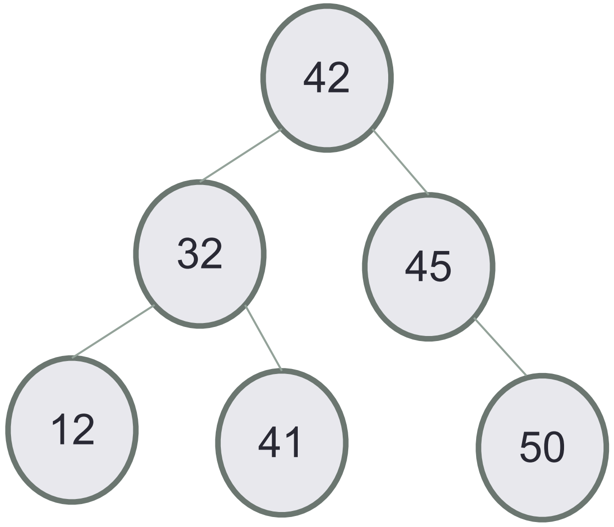
• Search (Binary search)		
• Min/Max		
• Median		
• Successor/Predecessor		
• Insert		
• Delete		

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

- Path – a sequence of (zero or more) connected nodes.
- Length of a path - number of edges traversed on the path
- Height of node – Length of the longest path from the node to a leaf node.
- **Height of the tree** - Length of the longest path from the **root** to a leaf node.

BSTs of different heights are possible with the same set of keys
Examples for keys: 12, 32, 41, 42, 45

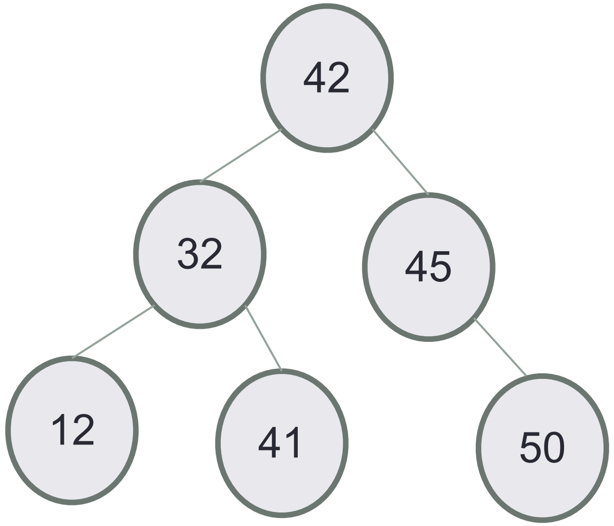
Worst case Big-O of search, insert, min, max



Given a BST of height H with N nodes, what is the worst case complexity of searching for a key?

A. O(1)
B. O(log H)
C. O(H)
D. O(H*log H)
E. O(N)

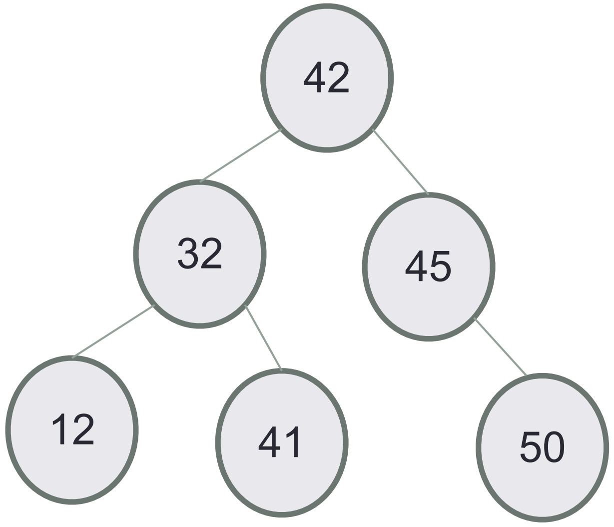
Worst case Big-O of predecessor / successor



Given a BST of height H and N nodes, what is the worst case complexity of finding the predecessor or successor key?

A. O(1)
B. O(log H)
C. O(H)
D. O(H*log H)
E. O(N)

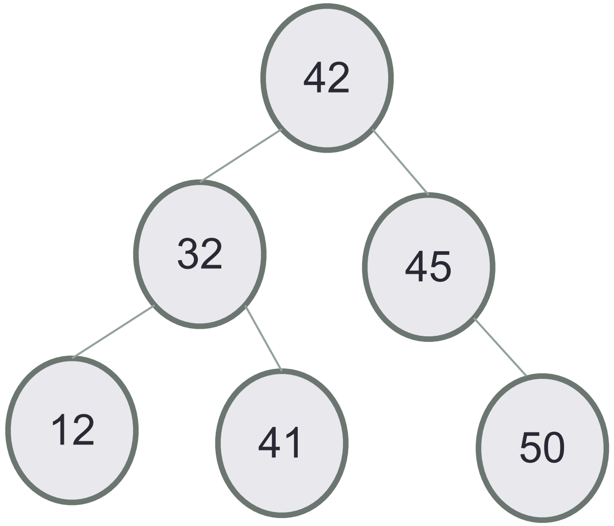
Worst case Big-O of delete



Given a BST of height H and N nodes, what is the worst case complexity of deleting a node?

A. O(1)
B. O(log H)
C. O(H)
D. O(H*log H)
E. O(N)

Big O of traversals

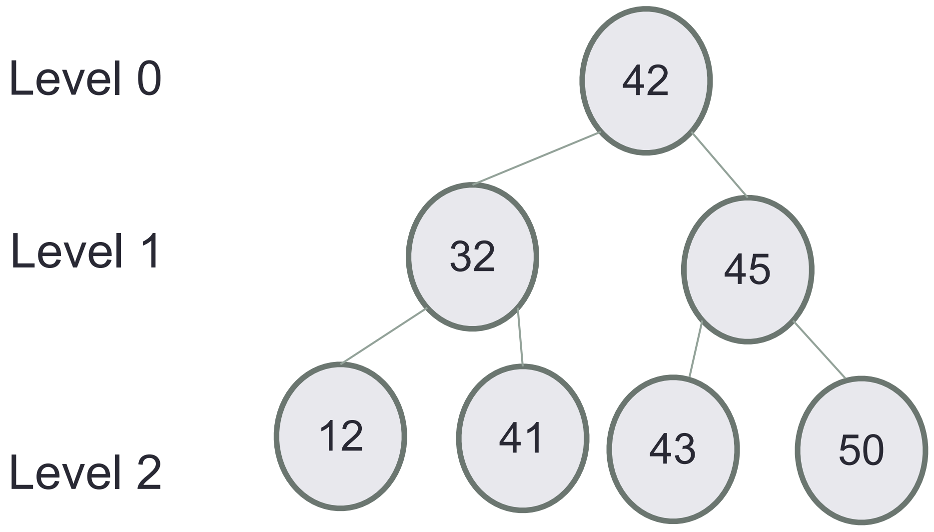


In Order:

Pre Order:

Post Order:

Types of BSTs

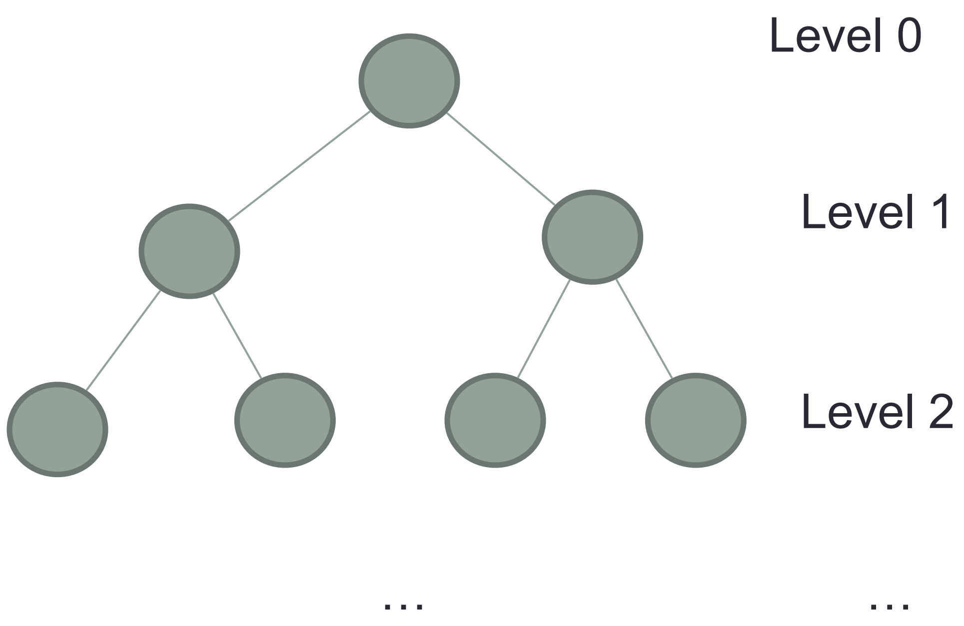


Balanced BST:

Complete Binary Tree: Every level, except possibly the last, is completely filled, and all nodes are as far left as possible

Full Binary Tree: A complete binary tree whose last level is completely filled

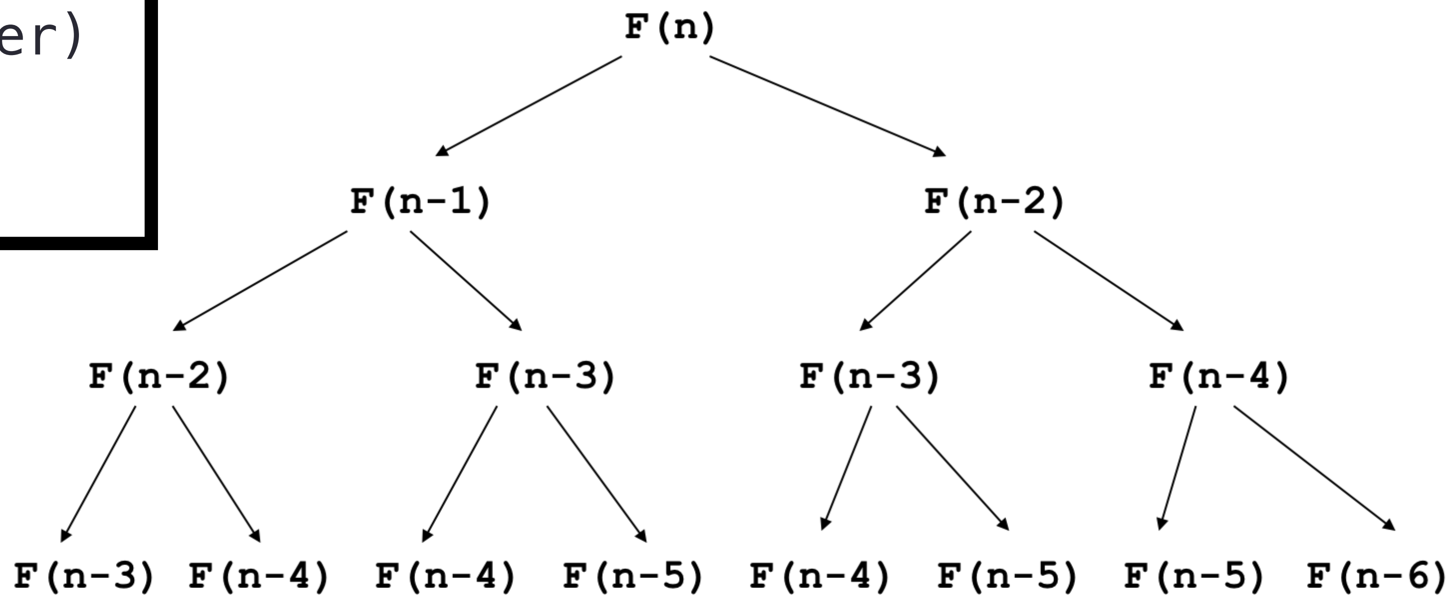
Relating H (height) and n (#nodes) for a full binary tree



Big-O analysis

```
procedure F(n: a positive integer)
    if(n <= 2) return 1
    return F(n–1) + F(n–2)
```

What takes so long? Let's unravel the recursion…



The same subproblems get solved over and over again!

Balanced trees

- Balanced trees by definition have a height of O(log n)
- A completely filled tree is one example of a balanced tree
- Other Balanced BSTs include AVL trees, red black trees and so on
- Visualize operations on an AVL tree: https://visualgo.net/bn/bst