BEST & WORST CASE ANALYSIS RUNNING TIME OF BST OPERATIONS

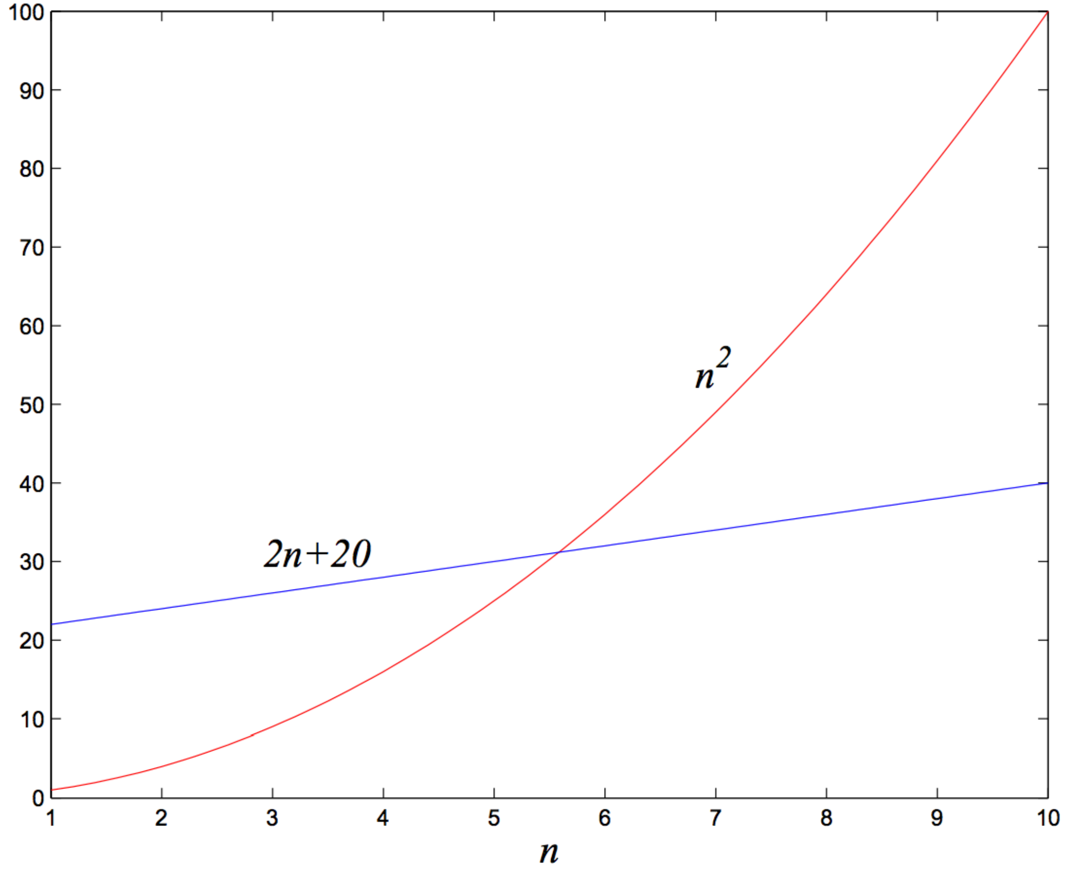
Problem Solving with Computers-II

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 $f(n) \le c \cdot g(n)$ for all $n >= k$.

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

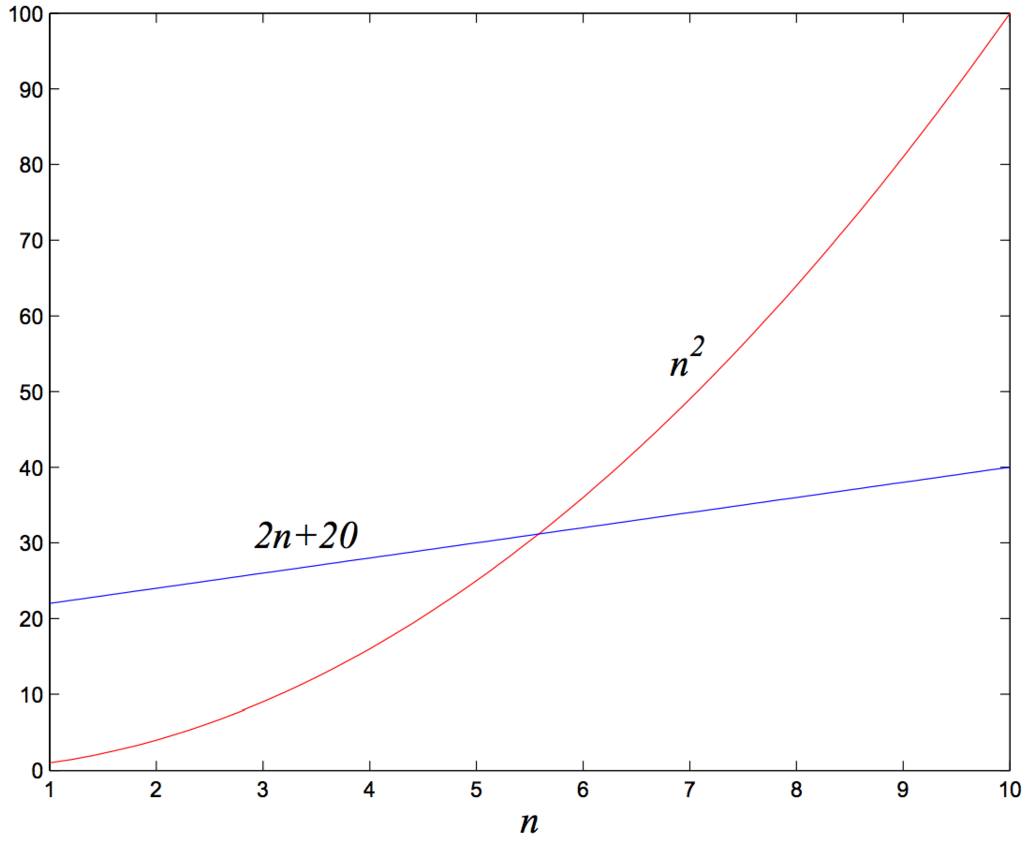


Big-Omega

- 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"

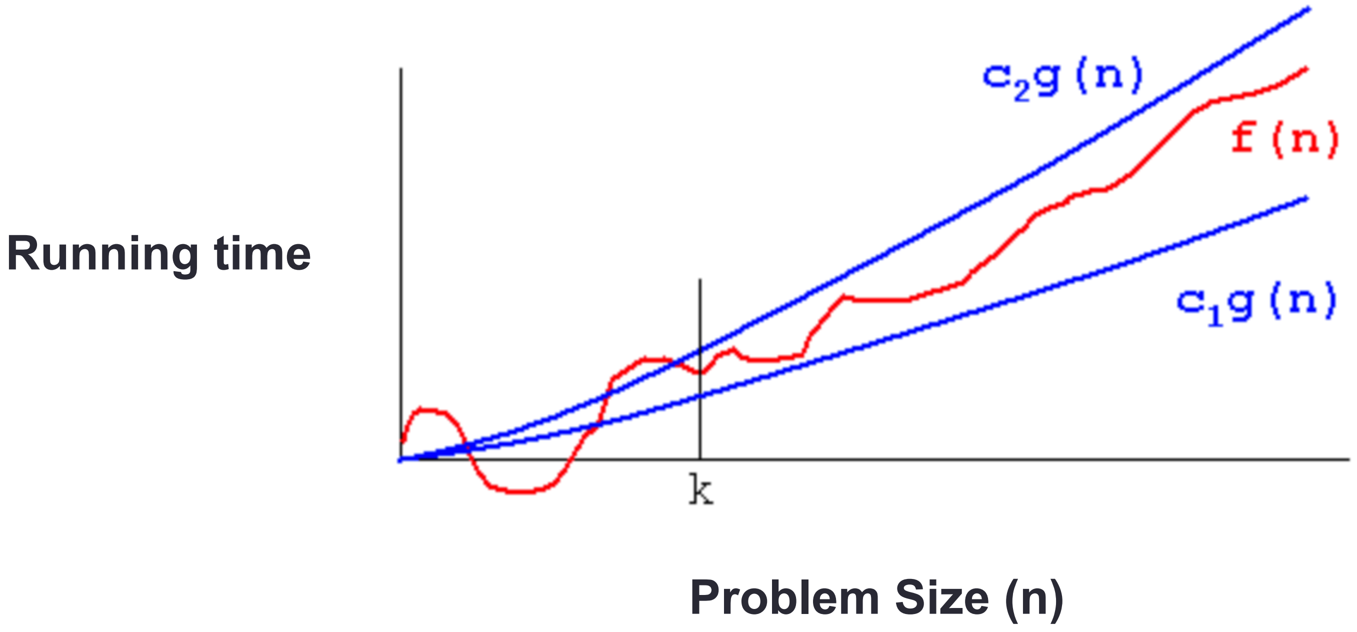


Big-Theta

- 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

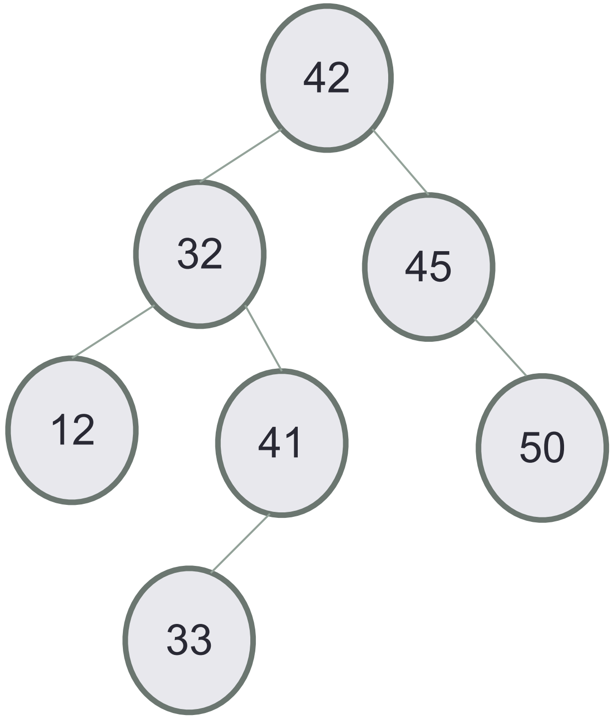
• Search (Binary search)		
• Min/Max		
• Median		
• Successor/Predecessor		
• Insert		
• Delete		

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

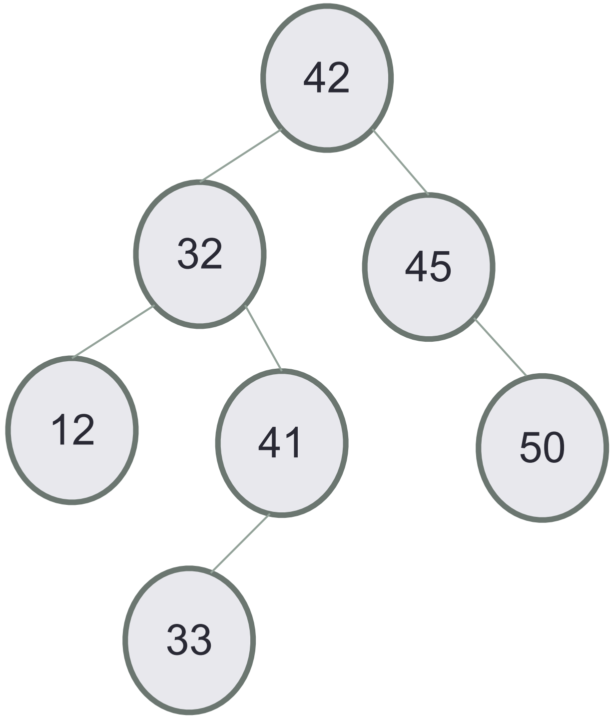
BST search - best case



Given a BST with N nodes, in the best case, which key would be searching for?

A. root node (e.g. 42)
B. any leaf node (e.g. 12 or 33 or 50)
C. leaf node that is on the longest path from the root (e.g. 33)
D. any key, there is no best or worst case

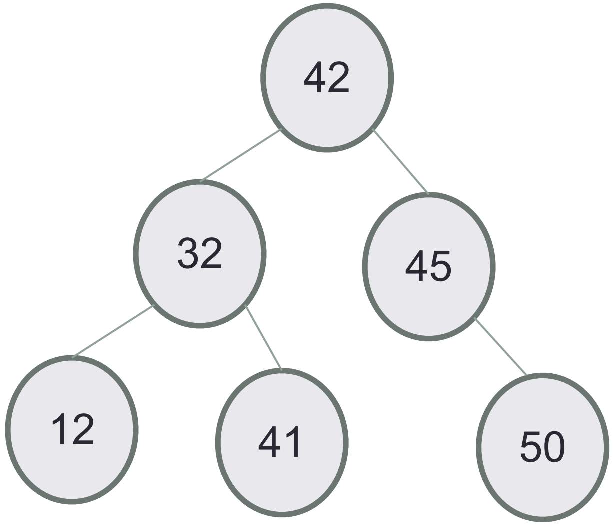
BST search - worst case



Given a BST with N nodes, in the worst case, which key would be searching for?

A. root node (e.g. 42)
B. leaf node (e.g. 12 or 41 or 50)
C. leaf node that is on the longest path from the root (e.g. 33)
D. a key that doesn’t exist in the tree

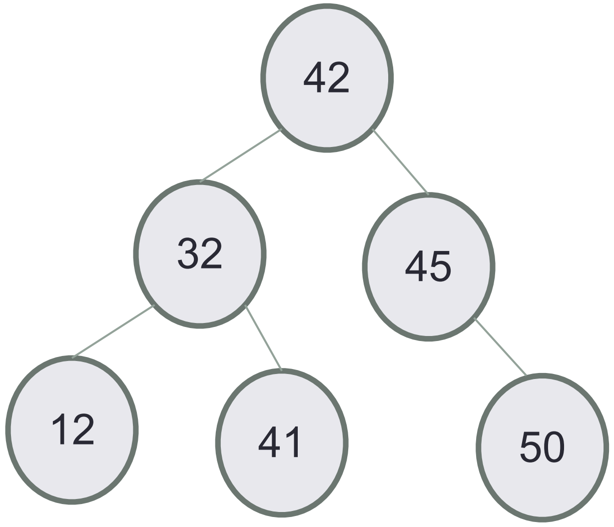
Worst case Big-O of search, insert, min, max



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

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

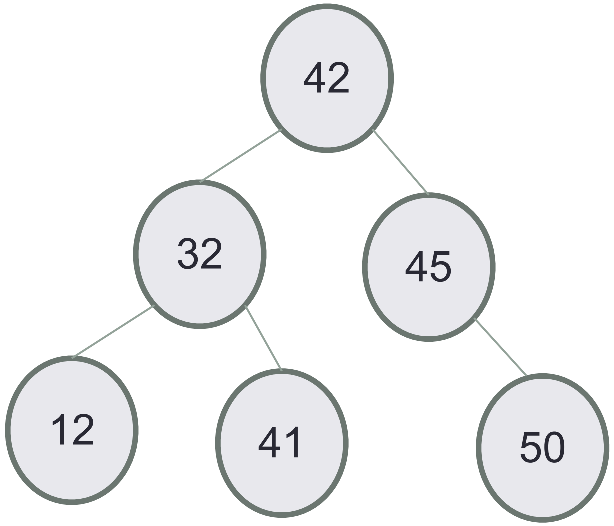
BST operations (worst case)



Given a BST of height H and N nodes, which of the following operations has a complexity of O(H)?

A. min or max
B. insert
C. predecessor or successor
D. delete
E. All of the above

Big O of traversals

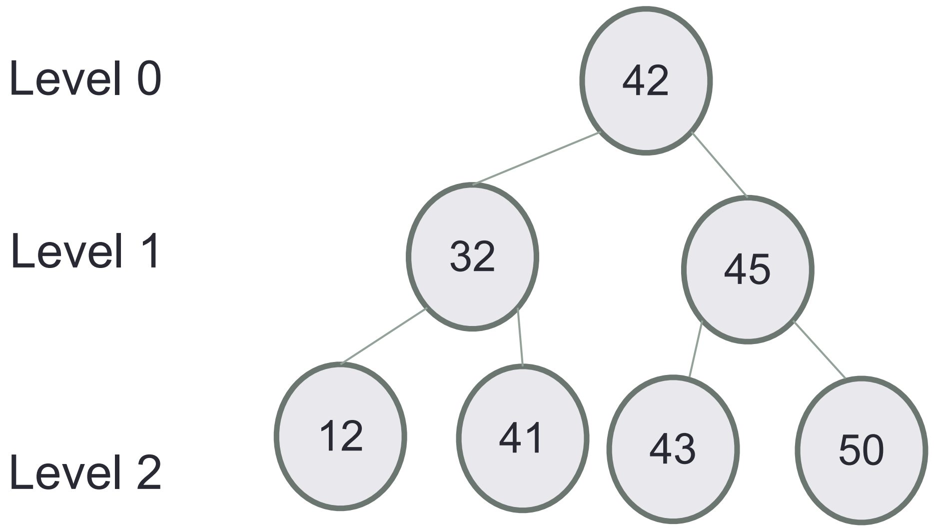


In Order:

Pre Order:

Post Order:

Types of BSTs

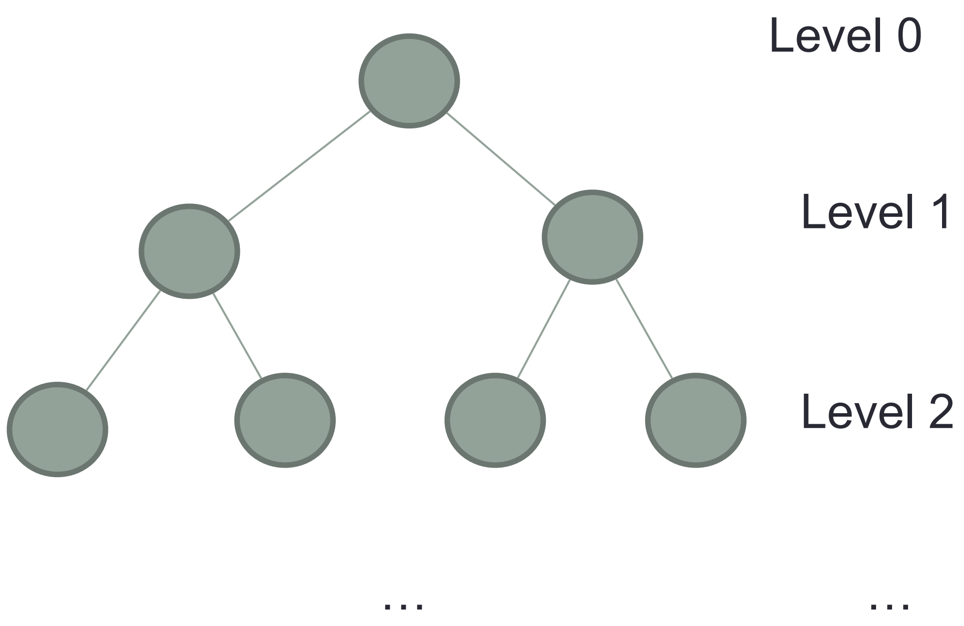


Balanced BST:

Complete Binary Tree: Every level, except possibly the last, is completely filled, and all nodes on the last level 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



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