RUNNING TIME ANALYSIS - PART 2

Problem Solving with Computers-II

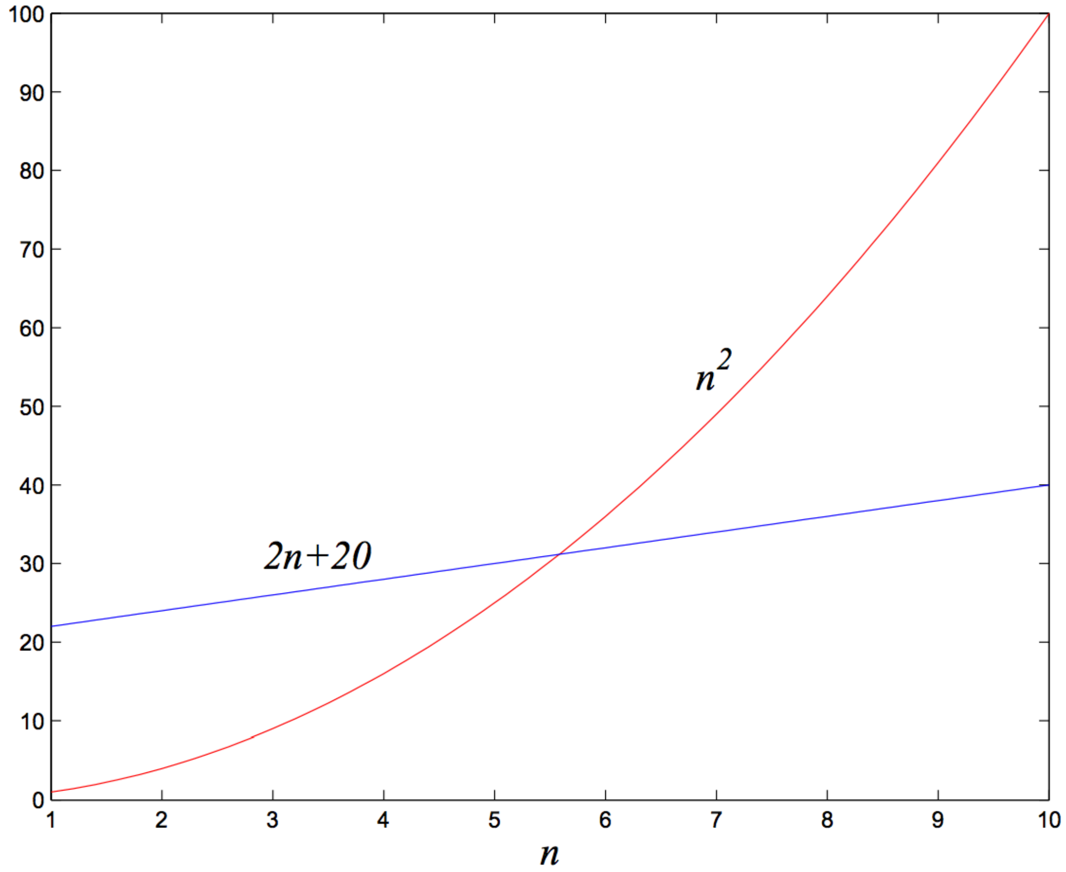
```
#include <iostream>
using namespace std;

int main(){
    cout<<"Hola Facebook\n";
    return 0;

}
```

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"



What is the Big O running time of sumArray2

A. $O(n^2)$
B. O(n)
C. O(n/2)
D. O(log n)
E. None of the array

```
/* n is the length of the array*/
int sumArray2(int arr[], int n)
{
        int result = 0;
        for(int i=0; i < n; i=i+2)
                result+=arr[i];
        return result;
}
```

What is the Big O of sumArray3

A. $O(n^2)$
B. O(n)
C. O(n/2)
D. O(log n)
E. None of the array

```
/* N is the length of the array*/
int sumArray3(int arr[], int n)
{
        int result = 0;
        for(int I = 1; i < n; i=i*2)
                result+=arr[i];
        return result;
}
```

Given the step counts for different algorithms, express the running time complexity using Big-O

1. 10000000
2. 3*n
3. 6*n-2
4. 15*n + 44
5. 50*n*log(n)
6. n^2
7. n^2-6n+9
8. $3n^2$+4*log(n)+1000

For polynomials, use only leading term, ignore coefficients: linear, quadratic

Common sense rules of Big-O

1. Multiplicative constants can be omitted: $14n^2$ becomes n^2 .
2. n^a dominates n^b if a > b: for instance, n^2 dominates n.
3. Any exponential dominates any polynomial: 3^n dominates n^5 (it even dominates 2^n).

Best case and worst case running times

Operations on sorted arrays of size n

- Min :
- Max:
- Median:
- Successor:
- Predecessor:
- Search**:**
- 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

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;
}
```

Binary Search Trees

- WHAT are the operations supported?

- HOW do we implement them?

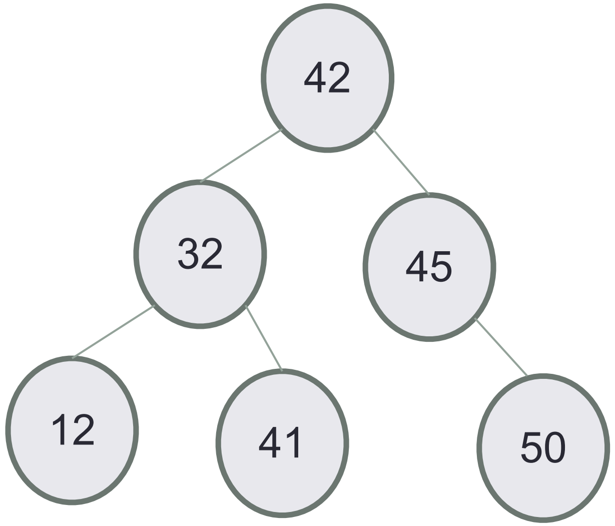
- WHAT are the (worst case) running times of each operation?

Height of the tree

- Path – a sequence of nodes and edges connecting a node with a descendant.
- A path starts from a node and ends at another node or a leaf
- Height of node – The height of a node is the number of edges on the longest downward path between that node and a leaf.

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

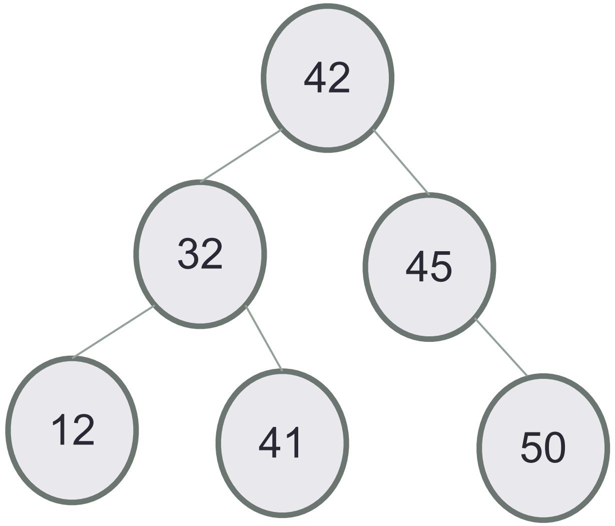
Worst case Big-O of search, insert, min, max



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)

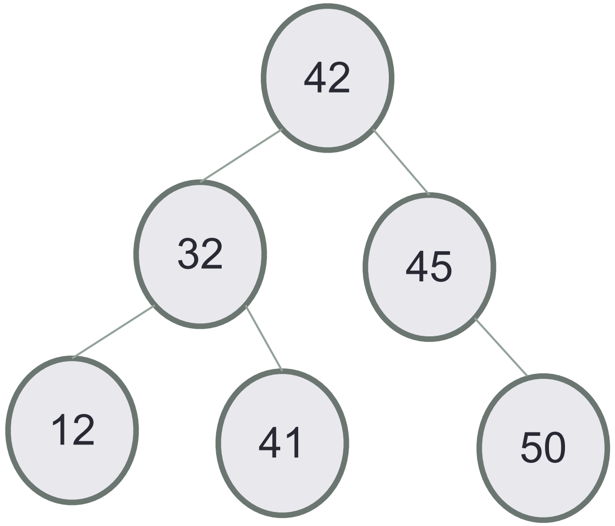
Worst case Big-O of predecessor / successor



- 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)

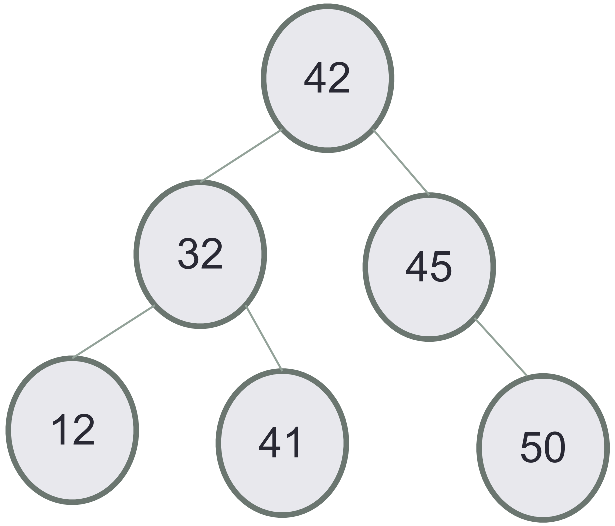
Worst case Big-O of delete



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:

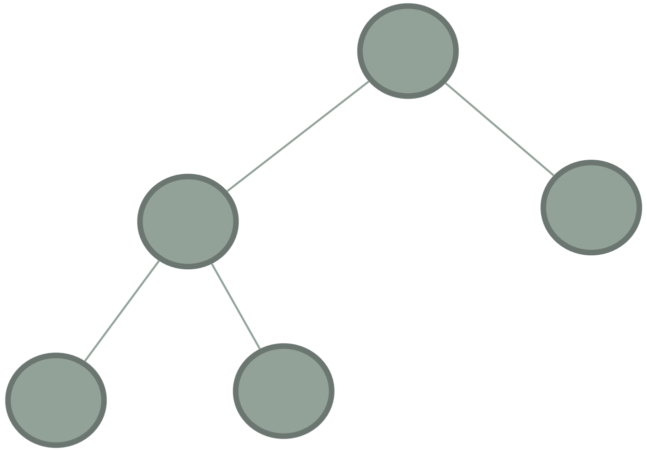
Pre Order:

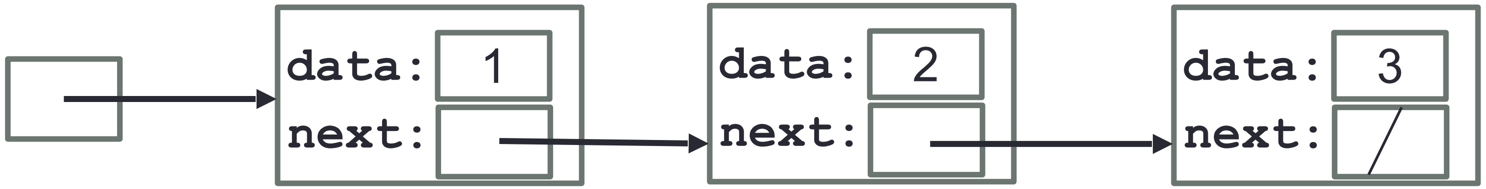
Post Order:

Worst case analysis

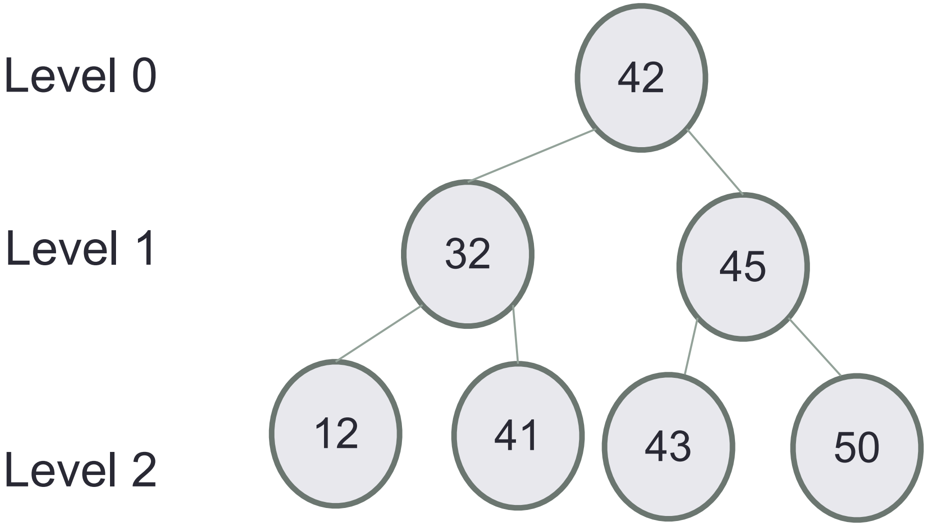
Are binary search trees *really* faster than linked lists for finding elements?

- A. Yes
- B. No



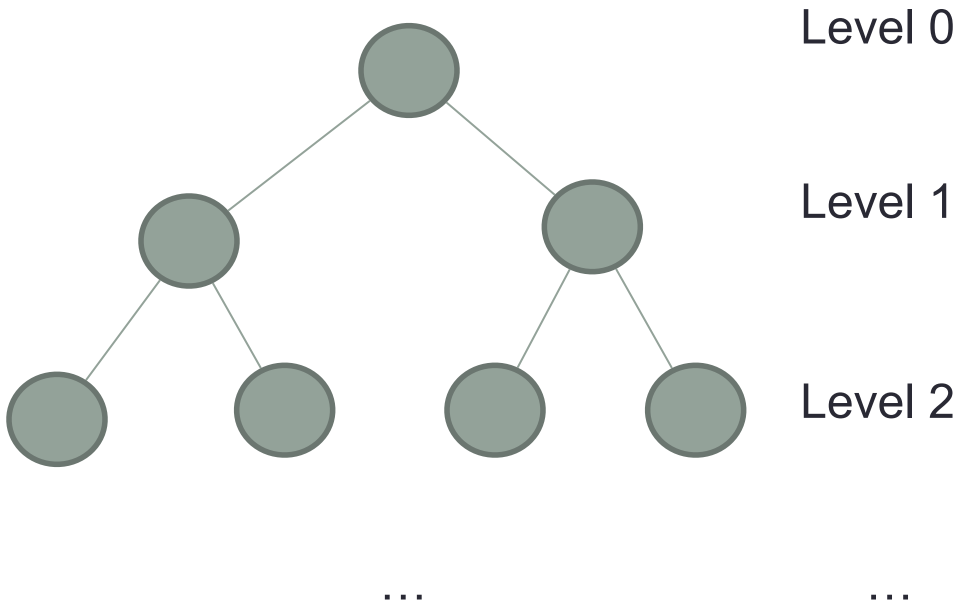


Completely filled binary tree



Nodes at each level have exactly two children, except the nodes at the last level

Relating H (height) and N (#nodes)
find is O(H), we want to find a f(N) = H



How many nodes are on level L in a completely filled binary search tree?

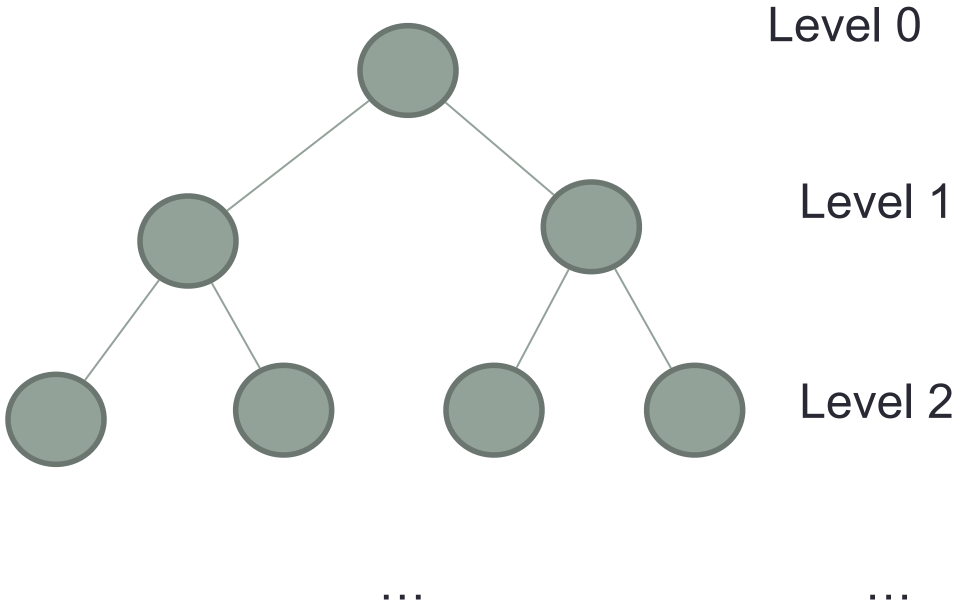
A.2

B.L

C.2*L

D.2^L

Relating H (height) and N (#nodes)
find is O(H), we want to find a f(N) = H



Finally, what is the height (exactly) of the tree in terms of N?

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