PROGRAMMING ASSIGNMENT - 1
RUNNING TIME ANALYSIS - PART 2

Problem Solving with Computers-II

Pick matching cards

Ordering:

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

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

h 3 > s10

Each player maintains an ordered hand of cards

How is this assignment different from lab4?

1. Keys in the BST are the combination of suit & value

2. A BST can store other types of keys as long as they are comparable

3. Need to use the BST in the implementation of the card game.

4. Need to design & test your own classes

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	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

bob.contains("c a")? NO

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

Bob

```
c 2
d a
h 10

d j
s 10
h a
```

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

```
c 2

d j
s 10
h a
```

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

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

alice.contains(__ha__)?

Alice
h 3
s 10
c a
s 5

	Bob
A	c 2
B	d j
C	s 10
D	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
```

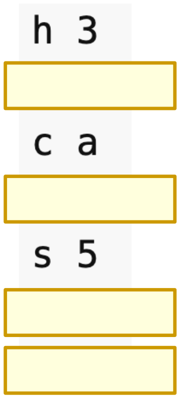
Should Alice take another turn? Yes / No

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

What is the condition to end?

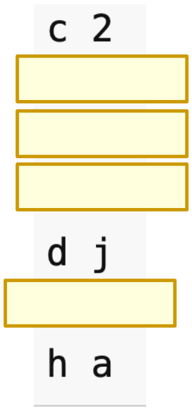
Clubs < diamonds < spades < hearts
ace < 2 < 3 …..< 10 < j < q < k

Alice



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



End game condition

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

Alice

```
h 3
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
```

```
Alice's cards:
c a
s 5
h 3

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

Bob

```
c 2
d j
h a
```

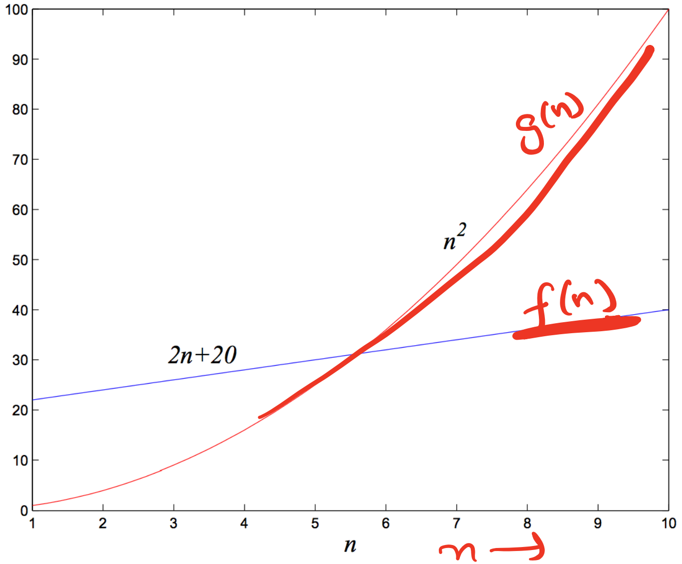
Definition of Big-O

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

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

$f(n) = O(n^2)$

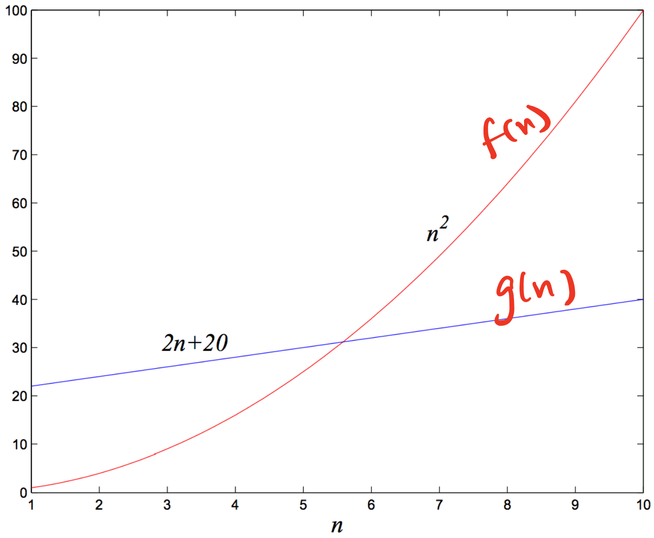


Big-Omega

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

$\boxed{f = \Omega(g)}$
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

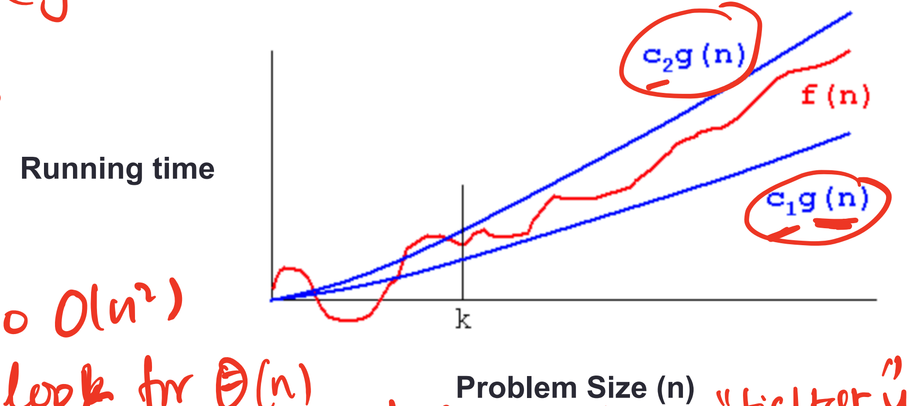
$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$, for $n >= k$

$f(n) = \Theta(g(n))$

$f(n) = 5n + 6$

$= O(n)$

$= \Theta(n)$



Although f(n) is also $O(n^2)$

in practice we look for $\Theta(n)$ and choose the "tighter" upper bound

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?

Best case - find the min key $O(1)$

Worst case - find the max key $O(n)$
or key doesn't exist

…using binary search?

Best case - find the mid value key $O(1)$

key doesn't exist - $O(\log n)$

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
begin						end	mid							end

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;
}
```

c_1 → We need to determine the number of iterations of while loop

c_2

iteration #	end - begin
1	$(n-1)$
2	$\frac{n-1}{2} - 1$
3	$\frac{n-1}{2^2} - \frac{1}{2} - 1$
⋮	
k	$\frac{n-1}{2^{k-1}} - \left(1 + \frac{1}{2} + \frac{1}{2^2} + \cdots + \frac{1}{2^{k-2}}\right)$

Iteration number $\qquad$ (end - begin)

k $\qquad$ $\frac{n-1}{2^{k-1}} - \left(1 + \frac{1}{2} + \frac{1}{2^2} + \cdots \frac{1}{2^{k-2}}\right)$

$$= \frac{(n-1)}{2^{k-1}} - \frac{\left(1 - \frac{1}{2^{k-1}}\right)}{\left(1 - \frac{1}{2}\right)}$$ (sum of geometric series)

$$= \frac{(n-1)}{2^{k-1}} - 2\left(1 - \frac{1}{2^{k-1}}\right)$$

$$= \frac{n-1+2}{2^{k-1}} - 2$$

$$= \frac{n+1}{2^{k-1}} - 2$$

Stop when (end - begin) < 0

$$\frac{n+1}{2^{k-1}} - 2 < 0$$

$$(n+1) < 2 \cdot 2^{k-1}$$

$$(n+1) < 2^k$$

$$\log(n+1) < k$$

The upper bound on the number of iterations of the while loop is $\log(n+1)$. In each iteration, there are const time operations. (c_2)

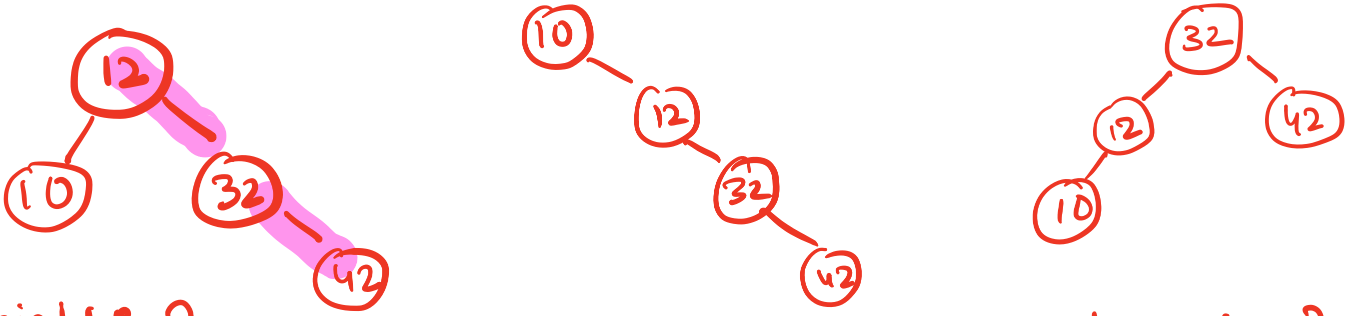
Therefore, running time of binary search, $T(n) \le c_1 + c_2 \log(n+1)$
$= O(\log n)$

Best case and worst case : sorted array

	Best case	Worst case
Search (Binary search)	$O(1)$	$O(\log n)$
Min/Max	$O(1)$	$O(1)$
Median	$O(1)$	$O(1)$
Successor/Predecessor	$O(1)$	$O(1)$
Insert	$O(1)$	$O(n)$
Delete	$O(1)$	$O(n)$

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. Height(32)? 1
- 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.

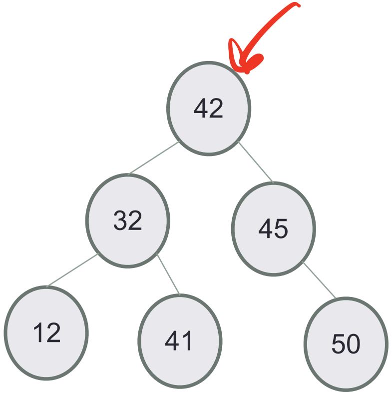


Height: 2 Height : 3 Height : 2

Path 12 → 32 length: 1

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

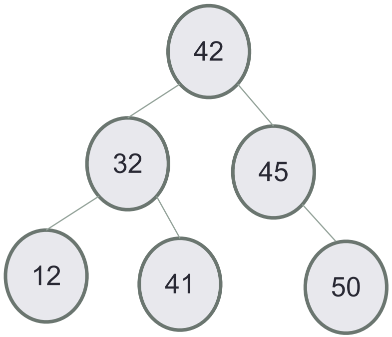


Best case: searching for root key: O(1)

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)

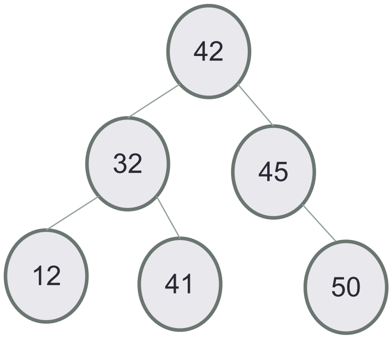
Worst case Big-O of predecessor / successor



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)

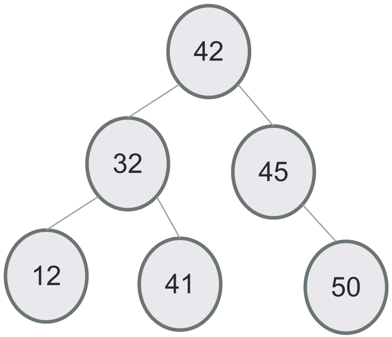
Worst case Big-O of delete



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

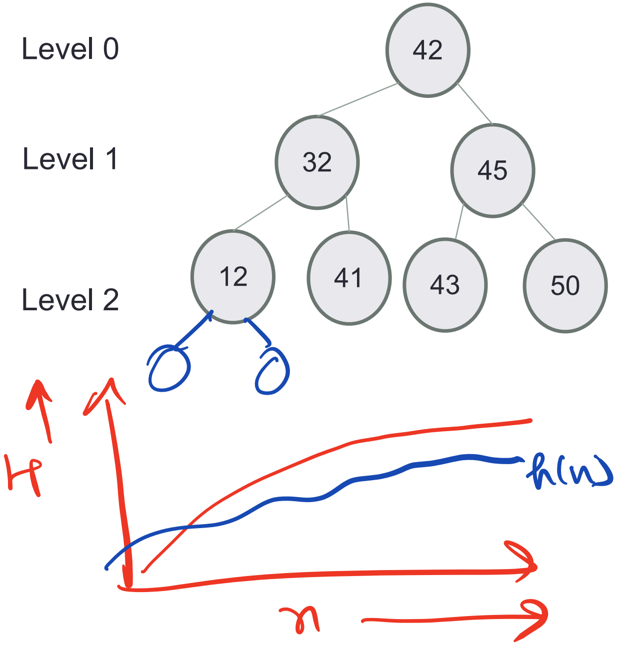


In Order: $O(n)$

Pre Order: $O(n)$

Post Order: $O(n)$

Types of BSTs



Balanced BST: $H = O(\log n)$

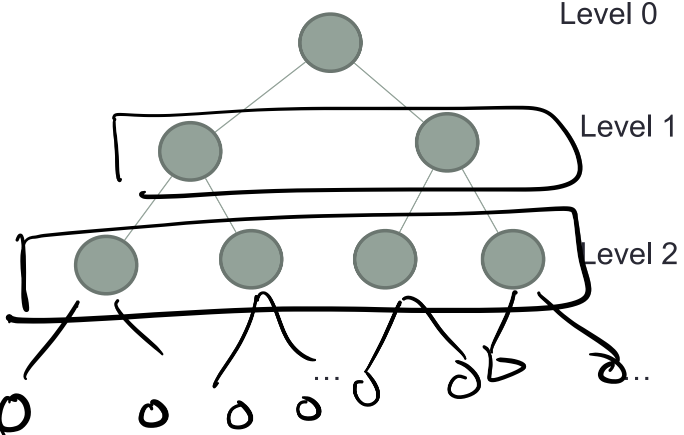
AVL, red-black

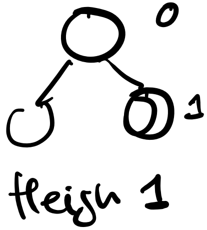
Complete Binary Tree: Every level, except possibly the last, is completely filled, and all nodes are as far left as possible

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

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

Level	# of nodes
0	2^0
1	2^1
2	2^2
H	2^H





Total nodes

$= 2^0 + 2^1 + 2^2 + \dots 2^{H}$

$n = 2^{H+1} - 1$

$(n+1) = 2^{H+1}$

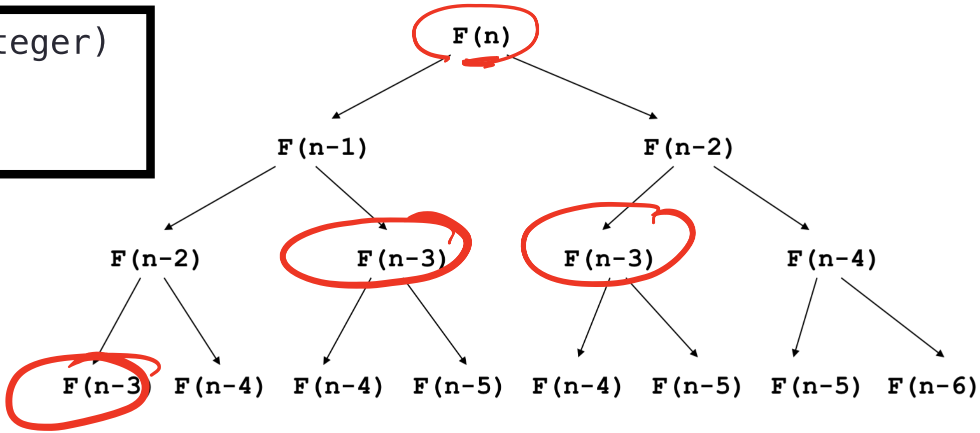
$\log_2(n+1) - 1 = H(n)$

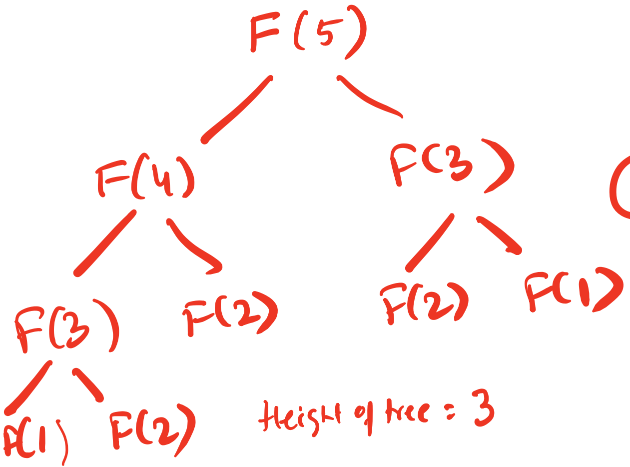
$H(n) = O(\log_2 n)$

Big-O analysis

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

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





Height of tree = 3

The same subproblems get solved over and over again!

The number of function calls is bounded by the number of nodes in a binary tree of height (n-2), which is $2^{n-2+1}-1 = 2^{n-1}-1$ (derived in the next lecture)

Running time = $O(2^n)$ ← exponential!!

Another approach to deriving the Big-O of running time

$T(n)$: Running time of $F(n)$

We have the following recurrence relation

$T(2) = T(1) = 1$

$$T(n) = T(n-1) + T(n-2) + c$$

c is some constant

$$\leq 2\,T(n-1) + c$$

$(T(n-1) > T(n-2)$, and we can make this approximation in Big-O analysis$)$

$$\leq 2\,(2\,T(n-2) + c) + c$$

(Substitute for $T(n-1)$)

$$= 2^2\,T(n-2) + 3c$$

Repeating this process we get

$$T(n) \leq 2^k\,T(n-k) + (2^k - 1)c$$

Base case $n-k = 1$

$\Rightarrow k = n-1$

Substitute for k to get

$$T(n) \leq 2^{n-1}\,T(1) + (2^{n-1} - 1)c$$

$$= 2^{n-1} \cdot 1 + 2^{n-1} \cdot c - c$$

$$= 2^{n-1}(1+c) - c$$

$$= O(2^n)$$

(same result as before!)

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