THE UNIVERSITY OF MELBOURNE

Binary Search Trees: Introduction

Semester 2, 2020

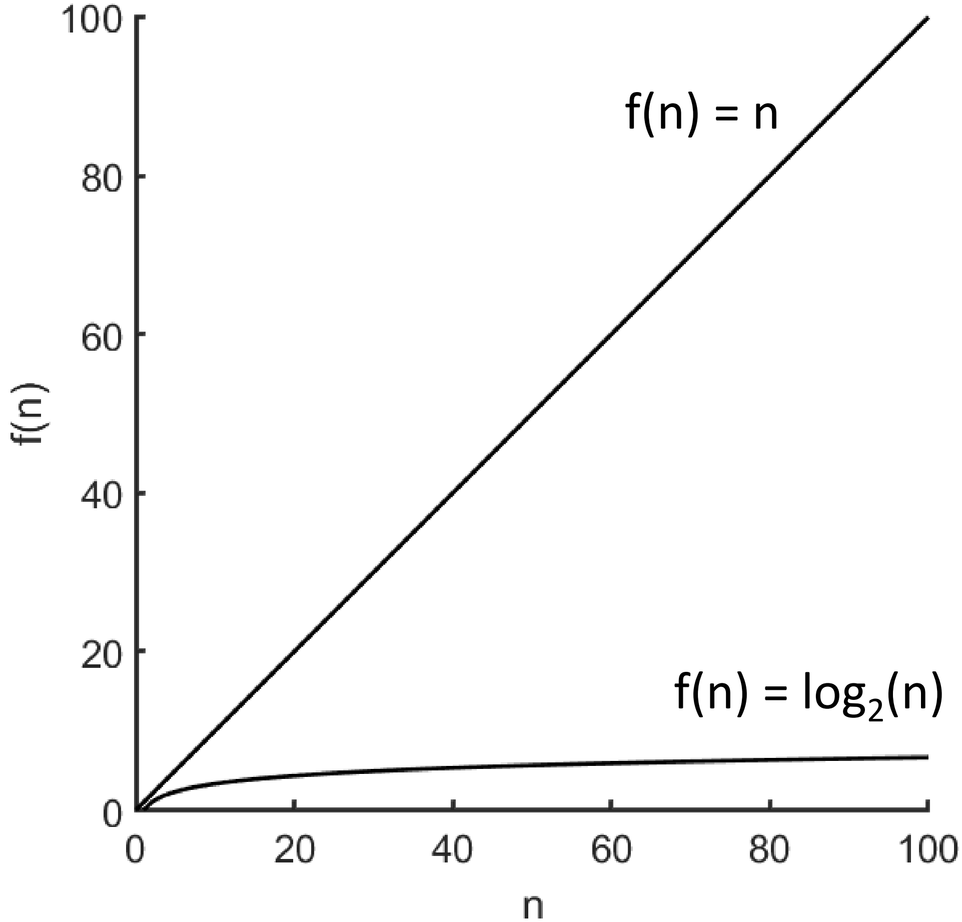
Kris Ehinger

Dictionary operations

- Search for an arbitrary key
- Insert item
- Delete item
- Return the maximum (or minimum) key
- Return the (previous / next) key
  - AKA the **in-order predecessor** or **in-order successor**

- How can we make all of these O(log n)?

O(n) versus O(log n)

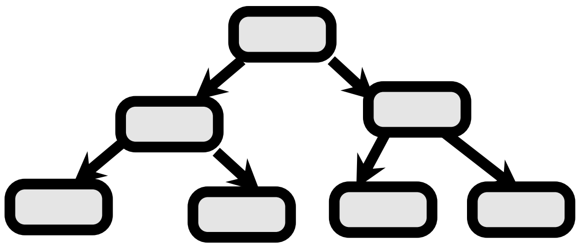


n	$\log_2(n)$
1,000	~10
1,000,000	~20
1,000,000,000	~30

Breaking out of linearity

- Compare:
- Linked list

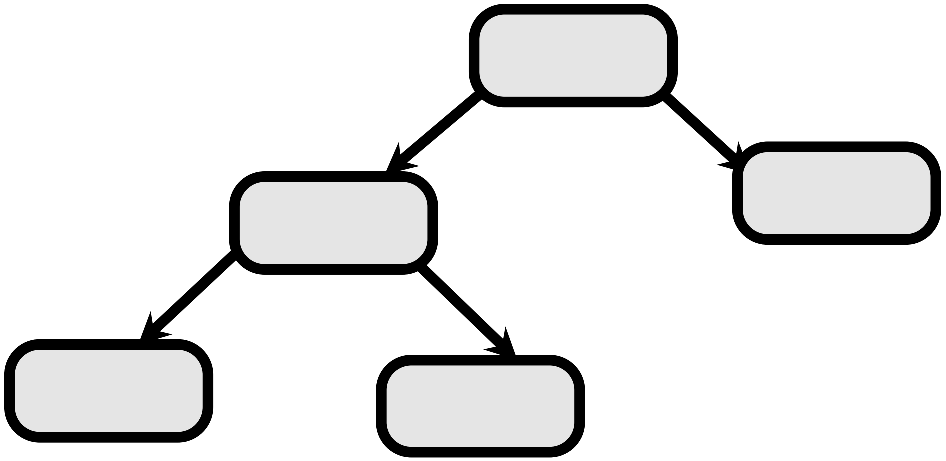
- Binary tree



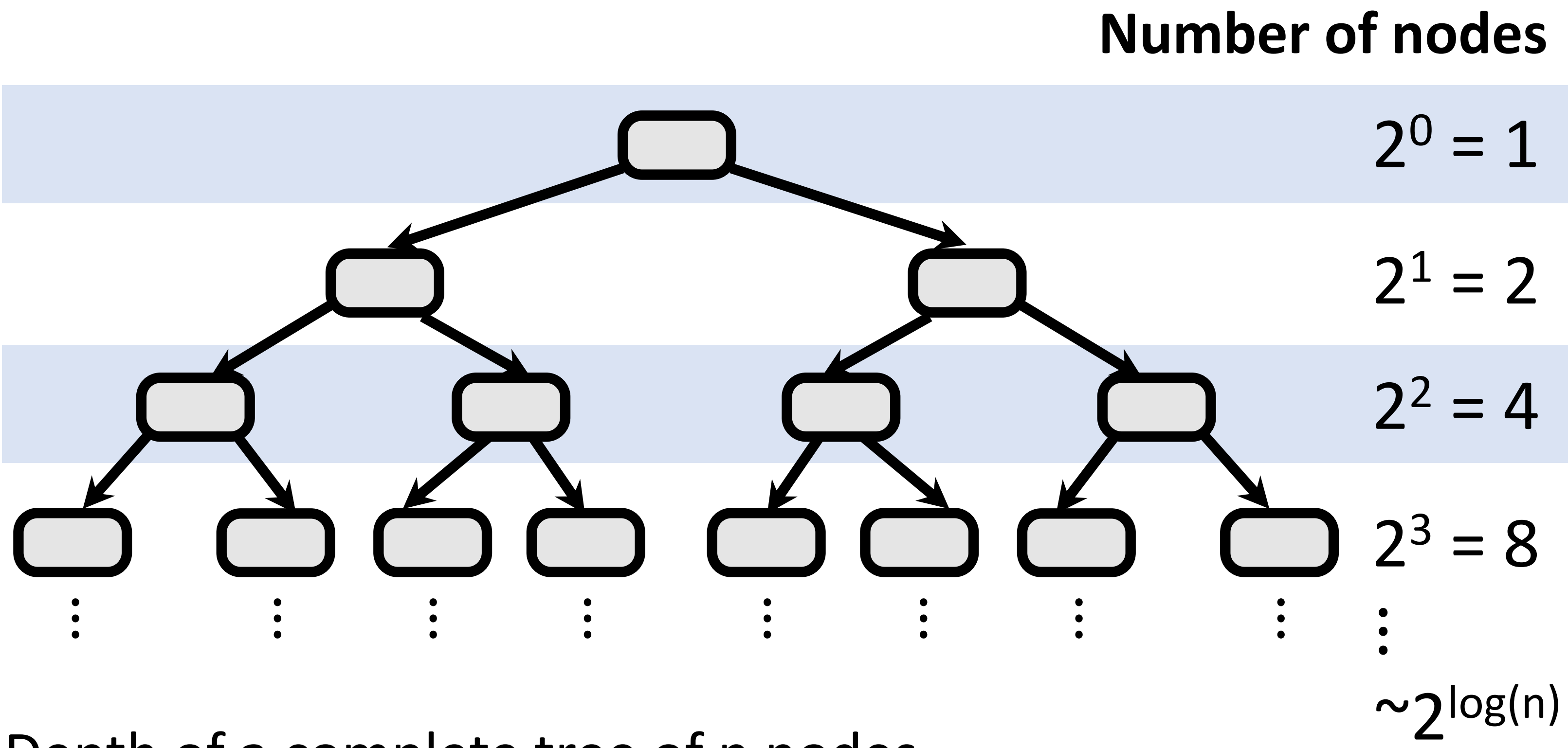
- If we knew whether the desired item was in the left subtree or the right subtree, we could find it more quickly!

Complete binary tree

- A binary tree is **complete** if every level, except (optionally) the last, is:
  - Completely filled, with
  - All nodes are as far left as possible



Complete binary tree



Depth of a complete tree of n nodes = approximately $\log_2(n)$

Binary search tree node

- Linked list node:

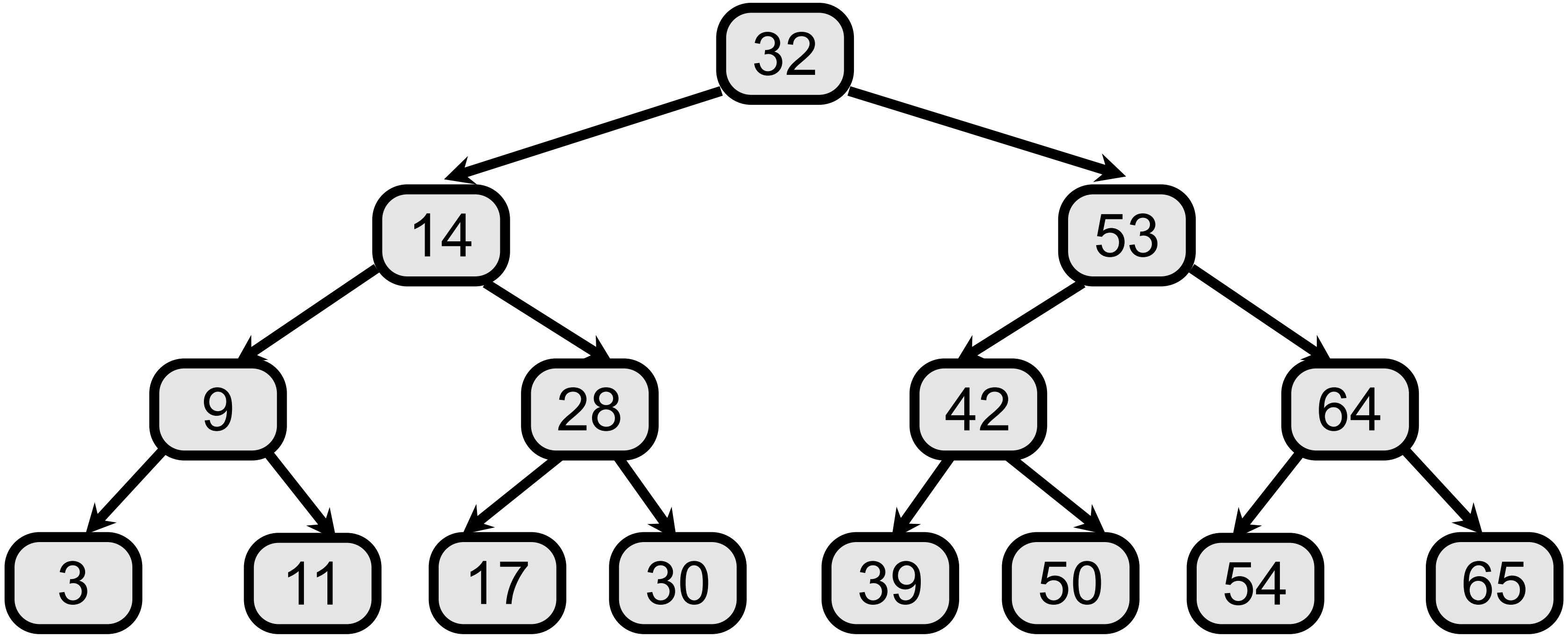
```
struct node {
    record r;
    struct node *next;
};
```

- Binary search tree node:

```
struct node {
    record r;
    struct node *left;
    struct node *right;
    struct node *parent;
};
```

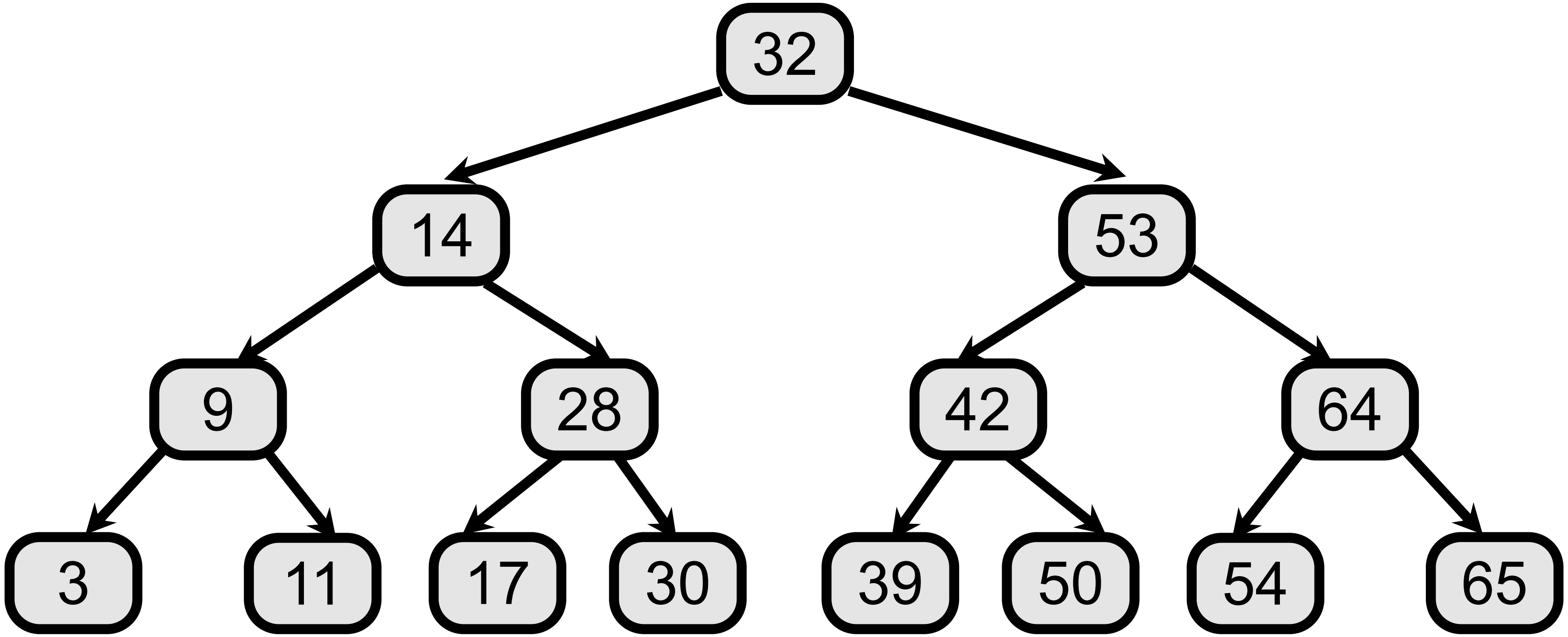
Binary search tree: search

- Find record with key = 42



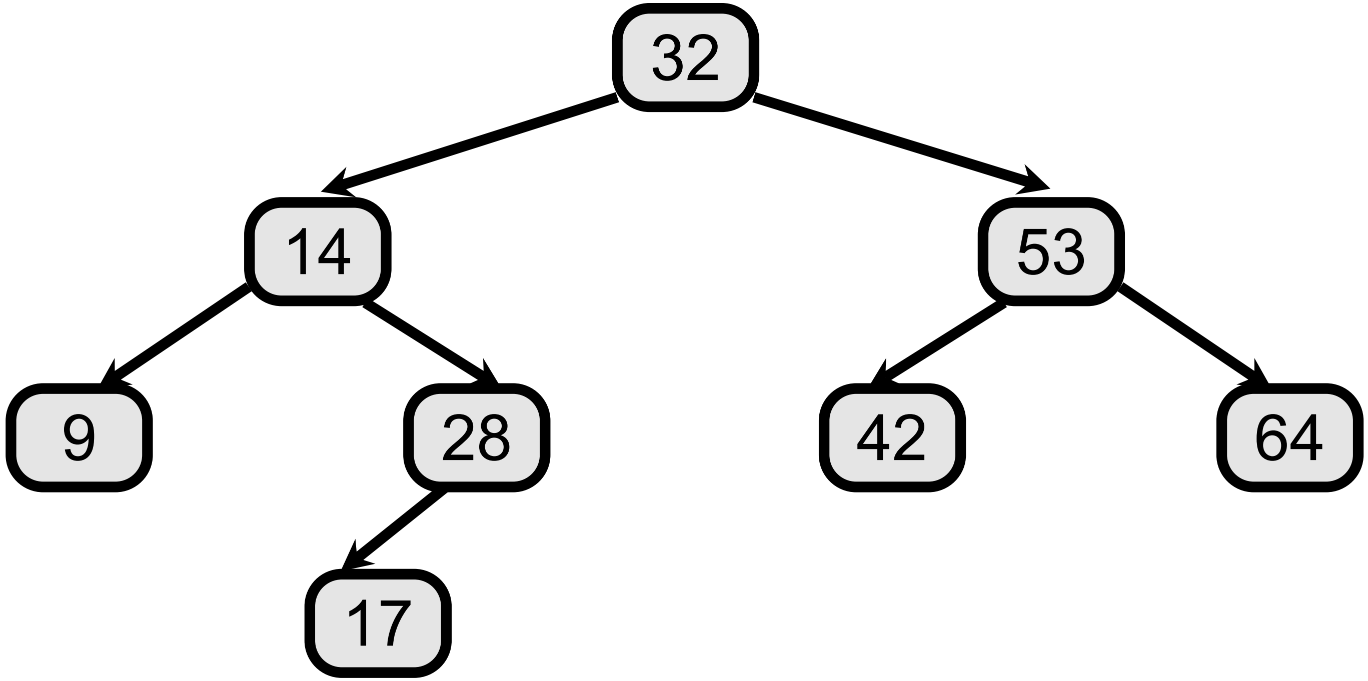
Binary search tree: search

- Find record with key = 10



Binary search tree: insert

- Insert node 17 into this tree:

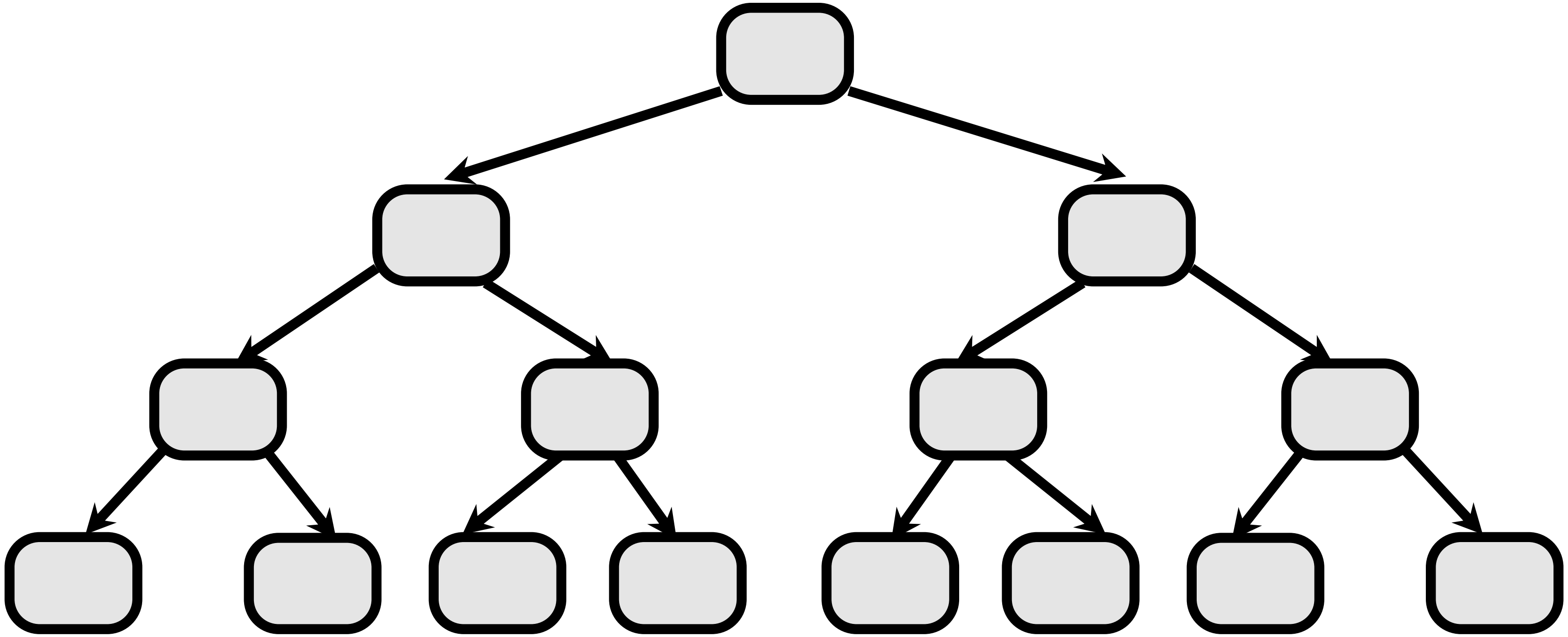


Building a binary search tree

- Try it: https://www.cs.usfca.edu/~galles/visualization/BST.html

Binary search tree: min/max

- How to find the smallest item in a tree? Largest?



Time complexity of BST

- Best case:
  - Height of tree = $\log_2(n)$
  - Path from root to any node
    - Longest: $\log_2(n)$
    - Average: $\sim\log_2(n)$
- Worst case: “stick” or linked list
  - Height of tree = n
  - Path from root to any node
    - Longest: n
    - Average: (n/2)