Lecture 20 - April 1

Binary Search Trees, Balanced BSTs

BST: Searching, Insertion
Hight Balance Property
Priority Queue: Introduction

Announcements/Reminders

- Assignment 4 (on linked Trees) released
- Makeup Lecture (for ProgTest2) to be posted
- Bonus opportunity: Final Course Evaluation
- Office hours 3pm Tue/Wed/Thu this week
- Lecture notes template, Office Hours, TA Contact

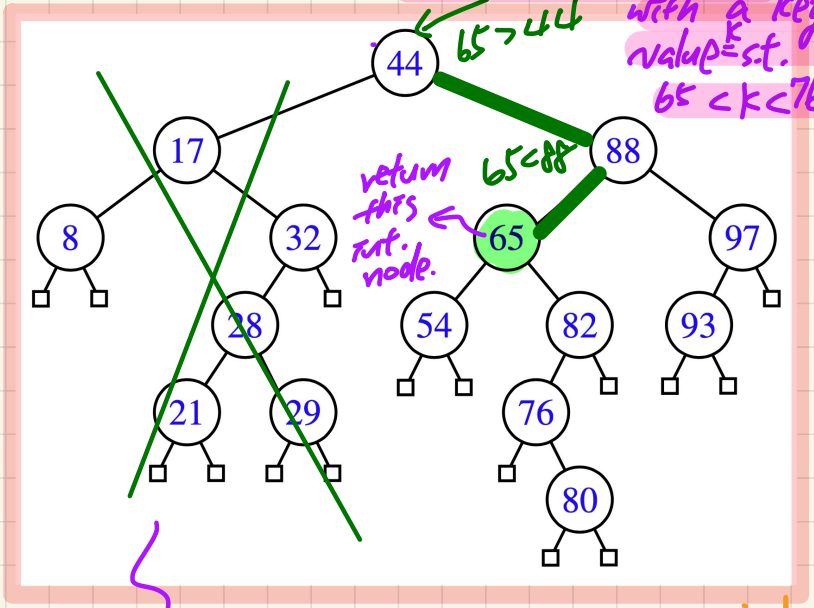
* i.o.t. : 54 65 ext 76 80 82 ...

BST Operation: Searching a Key

① successful
↳ r.n. int. node storing the matching key value
② unsuccessful
↳ rn. ext node

** for a subsequent insertion, this searching ext node can be set with a key value = k s.t. 65 < k < 76

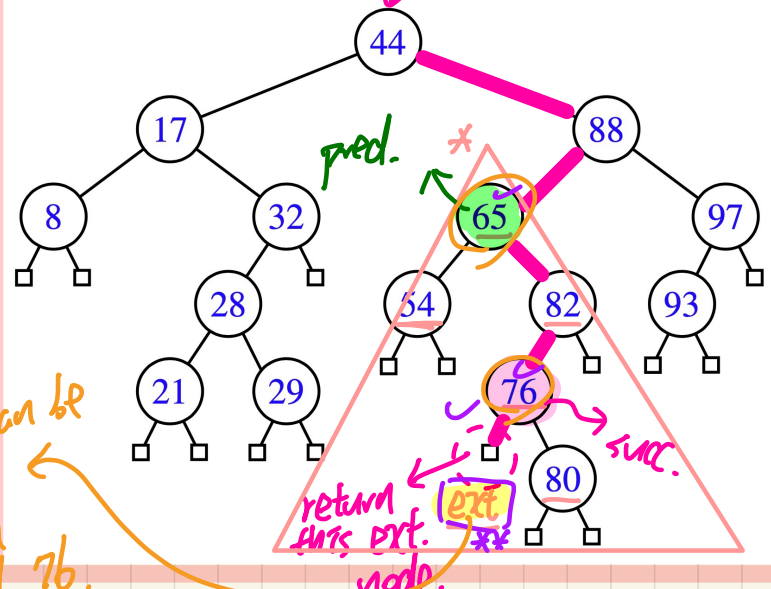
Search key 65 (key)



eliminate this subtree (∵ all keys < 44)
↳ (how large? 1/2 or O(1))

Search key 68

65 < k(ext) < 76

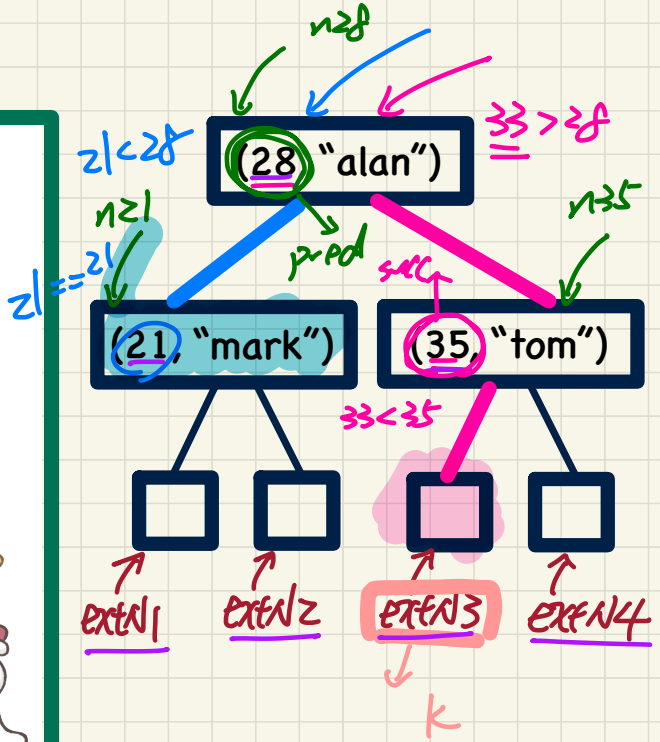


this key can be set between 65 and 76.

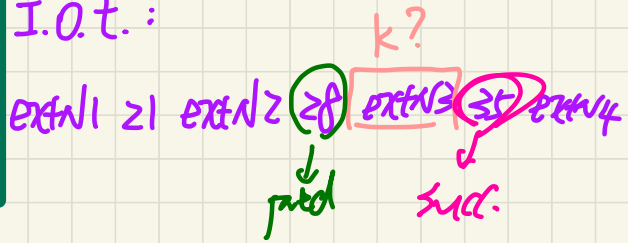
Tracing: Searching through a BST

```java
@Test
public void test_binary_search_trees_search() {
  BSTNode<String> n28 = new BSTNode<>(28, "alan");
  BSTNode<String> n21 = new BSTNode<>(21, "mark");
  BSTNode<String> n35 = new BSTNode<>(35, "tom");
  BSTNode<String> extN1 = new BSTNode<>();
  BSTNode<String> extN2 = new BSTNode<>();
  BSTNode<String> extN3 = new BSTNode<>();
  BSTNode<String> extN4 = new BSTNode<>();
  n28.setLeft(n21); n21.setParent(n28);
  n28.setRight(n35); n35.setParent(n28);
  n21.setLeft(extN1); extN1.setParent(n21);
  n21.setRight(extN2); extN2.setParent(n21);
  n35.setLeft(extN3); extN3.setParent(n35);
  n35.setRight(extN4); extN4.setParent(n35);

  BSTUtilities<String> u = new BSTUtilities<>();
  /* search existing keys */
  assertTrue(n28 == u.search(n28, 28));
  assertTrue(n21 == u.search(n28, 21));
  assertTrue(n35 == u.search(n28, 35));
  /* search non-existing keys */
  assertTrue(extN1 == u.search(n28, 17)); /* *17* < 21 */
  assertTrue(extN2 == u.search(n28, 23)); /* 21 < *23* < 28 */
  assertTrue(extN3 == u.search(n28, 33)); /* 28 < *33* < 35 */
  assertTrue(extN4 == u.search(n28, 38)); /* 35 < *38* */
}
```



I.O.t.:



* Recall: given N nodes : $\_\_\_ \le h \le \_\_\_$

Running Time: Search on a BST

ext. or int.

```
public BSTNode<E> search(BSTNode<E> p, int k) {
  BSTNode<E> result = null;
  if(p.isExternal()) {
    result = p; /* unsuccessful search */
  }
  else if(p.getKey() == k) {
    result = p; /* successful search */
  }
  else if(k < p.getKey()) {
    result = search(p.getLeft(), k);
  }
  else if(k > p.getKey()) {
    result = search(p.getRight(), k);
  }
  return result;
}
```

p: root

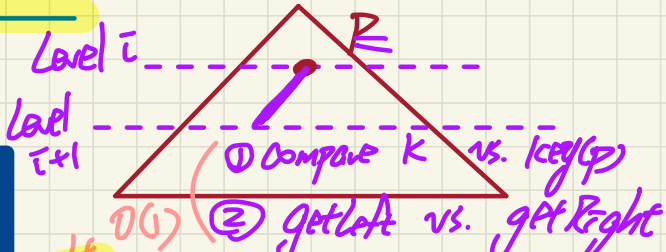
k: key to search.

ext. node

int. node

R1

R2

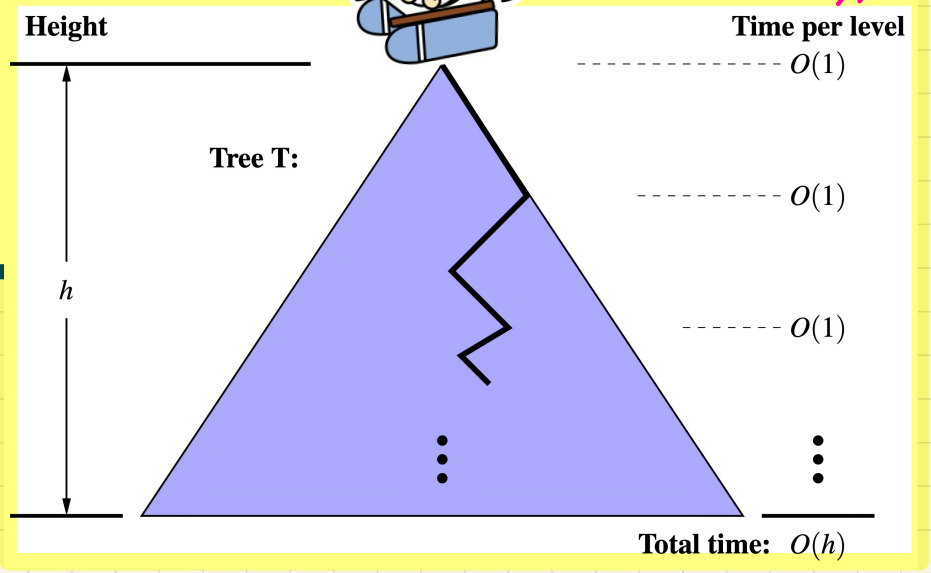


tight, accurate

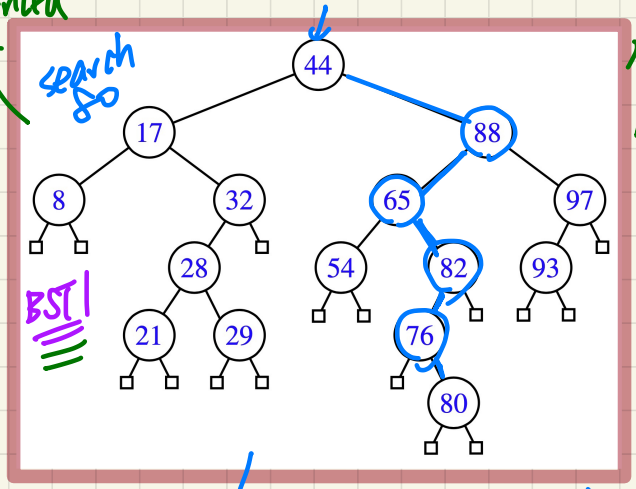
(A1) *O(h)

(A2) O(logN) → best case, lower bound

(A3) O(N) → worst case, upper bound



Binary Search: Non-Linear vs. Linear Structures



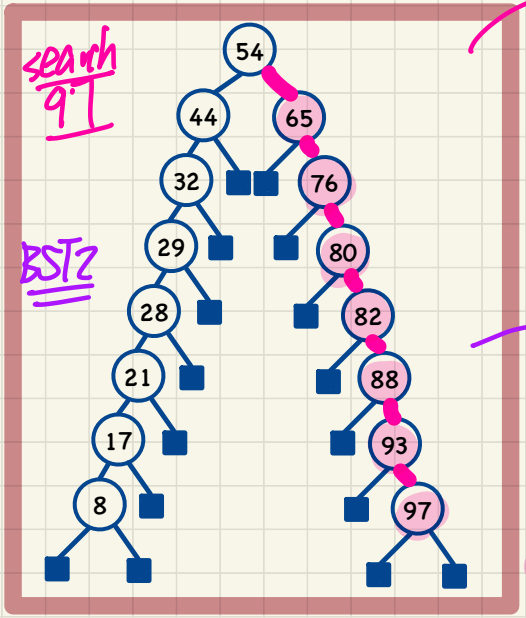
$N = 15$

$h = 5$

$\approx O(\log N)$

searching 80 on BST1 $\approx$ performing a binary search on a

T.o.t.(BST1) = T.o.t(BST2) = a



$N = 15$

$h = 7$

$\frac{N}{2}$ $O(N)$

searching 97 on BST2 $\approx$ performing a linear search on a

a

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
8	17	21	28	29	32	44	54	65	76	80	82	88	93	97

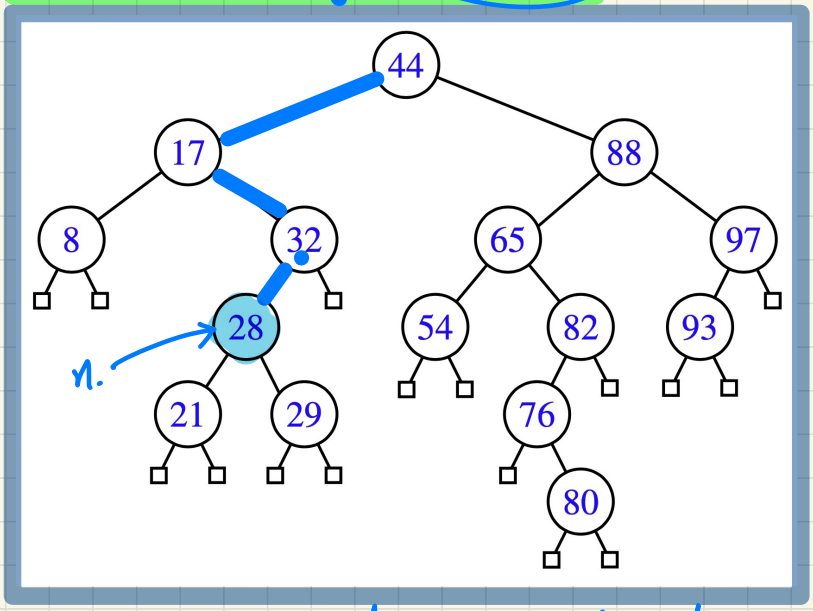
REVIEW ! a

Visualizing BST Operation: Insertion

RT: dominated by O(h) searching →

Insert Entry (28, "suyeon")

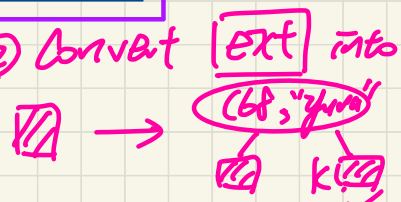
k. v.



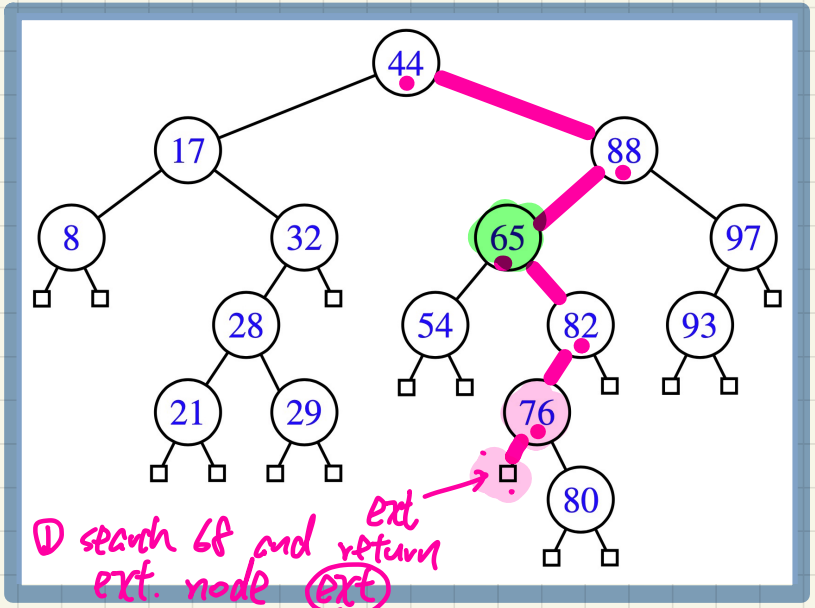
① search 28 and return int. node n

② n.setElement("suyeon");

② convert ext into an int. node.



Insert Entry (68, "yuna")

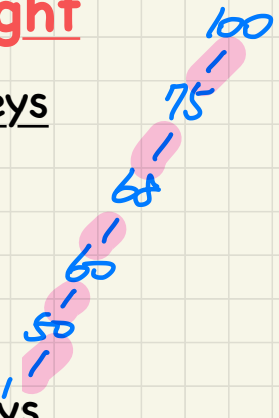


① search 68 and return ext. node ext

Worst-Case RT: BST with Linear Height

Example 1: Inserted Entries with Decreasing Keys

<100, 75, 68, 60, 50, 1>



Example 2: Inserted Entries with Increasing Keys

<1, 50, 60, 68, 75, 100>

BST with O(N) height
⇒ search/insert/delete can be O(N)

exercises (heights? N? log N?)

Example 3: Inserted Entries with In-Between Keys

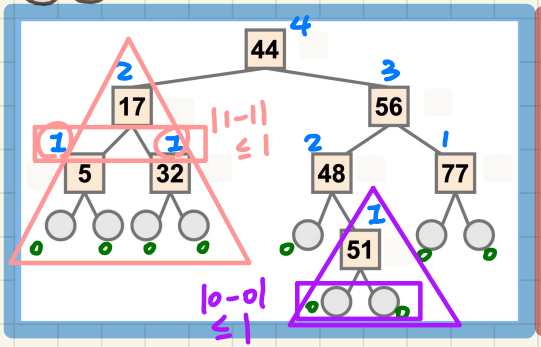
<1, 100, 50, 75, 60, 68>

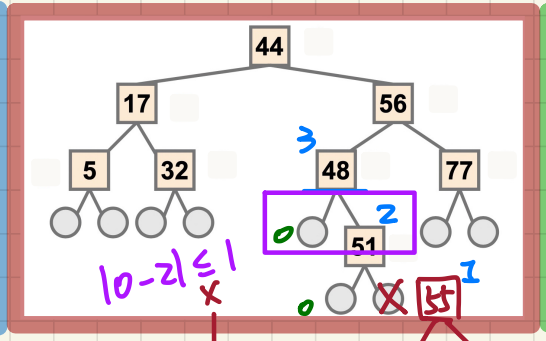
Balanced BST: Definition

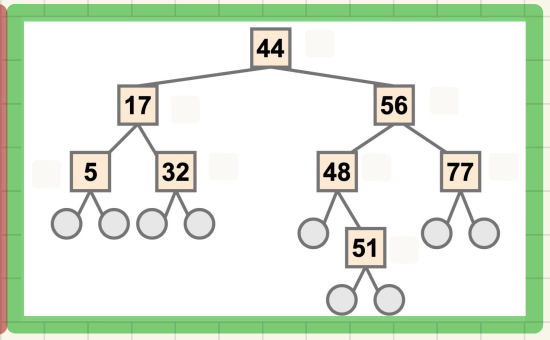
- internal node
- height
- height balance

Given a node p, the ***height*** of the subtree rooted at p is:

$$\textit{height}(p) = \begin{cases} 0 & \text{if } p \text{ is } \textit{external} \\ 1 + \textbf{MAX}\ (\{\ \textit{height}(c) \mid \textit{parent}\ (c) = p\ \}) & \text{if } p \text{ is } \textit{internal} \end{cases}$$





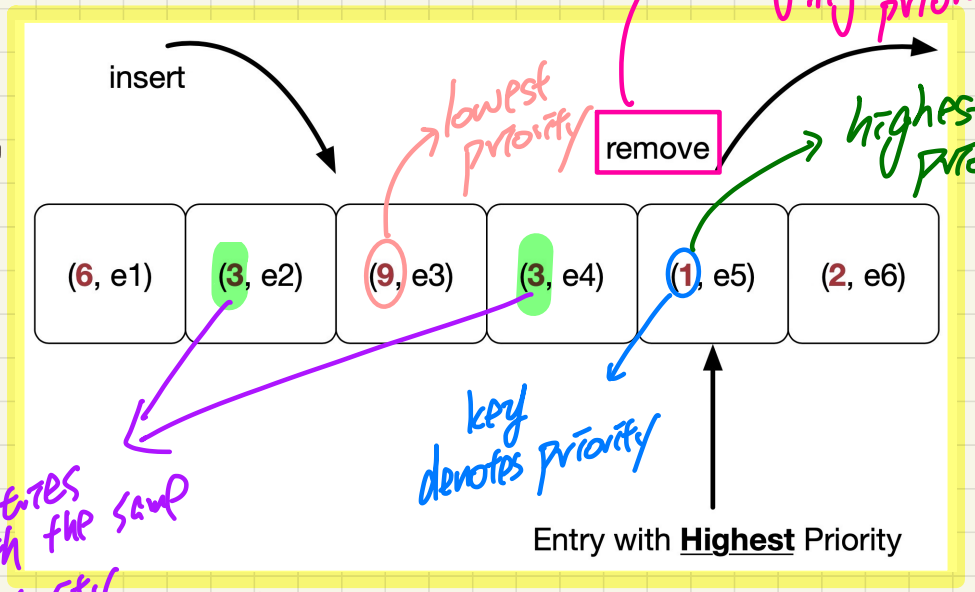


Q. Is the above tree a balanced BST?

Q. Still a balanced BST after inserting 55?

Q. Still a balanced BST after inserting 63?

What is a Priority Queue (PQ)



remove entry with highest priority

k ↓ priority ↑

lowest priority

highest priority

key denotes priority

Entries with the same priority (does not matter which entry is retrieved)

Compare PQ with FIFO queue

1. entries removed from PQ according to priorities.
2. entries removed from FIFO acc. to chronological order