Lecture 22 - Wednesday, March 29

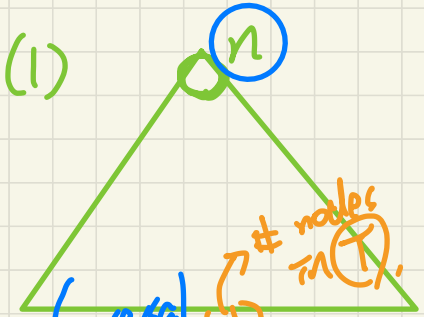
Announcements

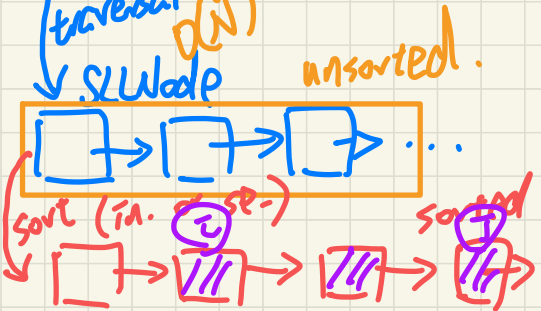
- Bonus Opportunity - Course Evaluation
- ProgTest1: Jackie (Office Hour)
- Assignment3 solution released, ProgTest2

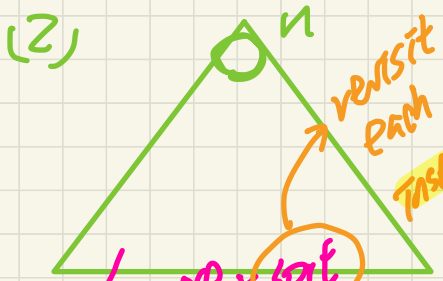
Assignment 3

Task 1 : Rank

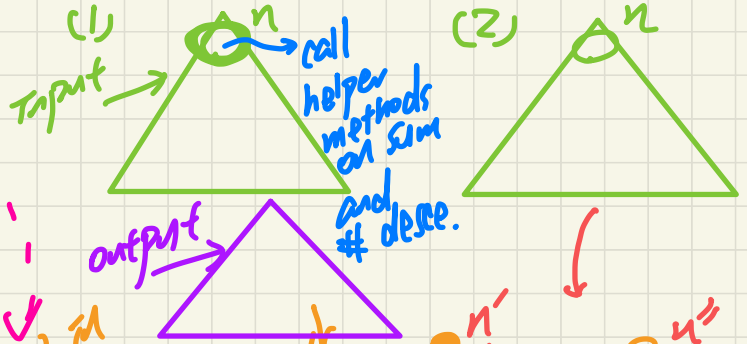
task1 (TN n, int i, int j)

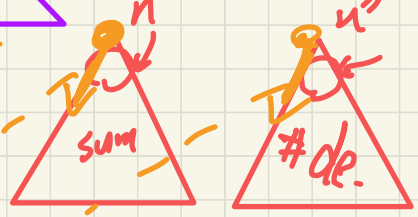


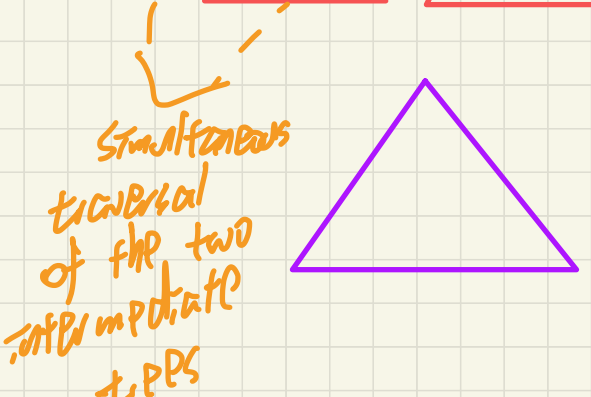




Task 2 : Stats







Lecture

Binary Search Tree (BST)

Implementing a Generic BST in Java
Searching

BST operations:

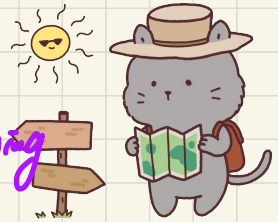
1. Input: a BST
   - ↳ search property

2.1 Searching
2.2 Insertion
2.3 Deletion

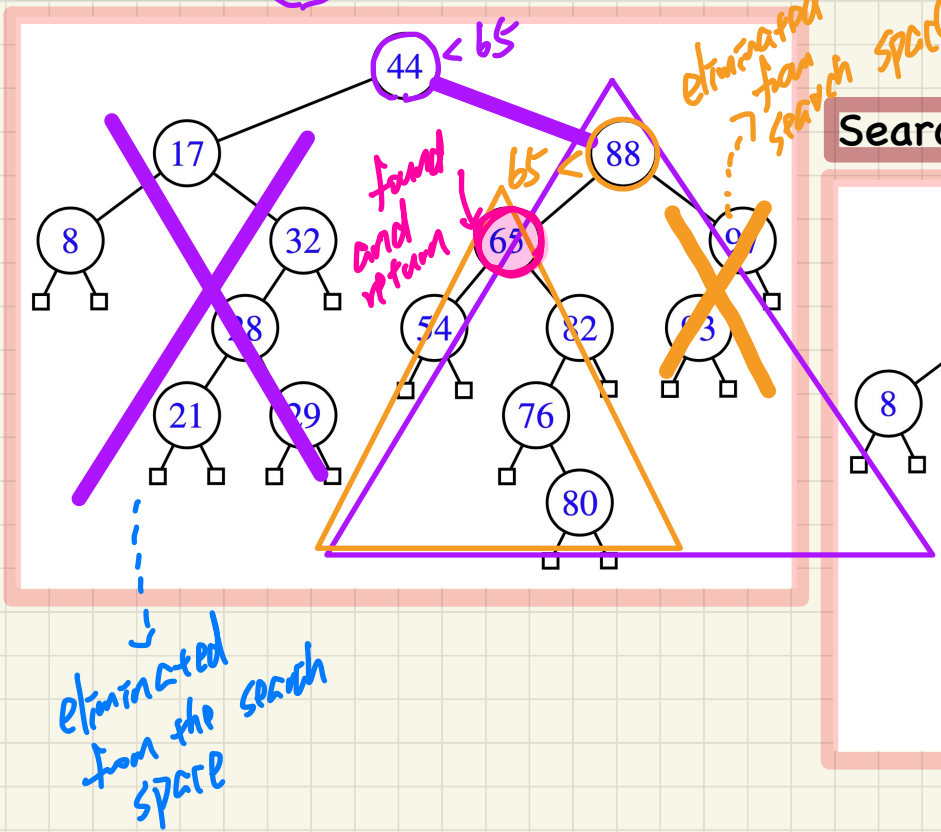
} Critical
- ↳ Search property maintained in the output: it remains a BST.

BST Operation: Searching a Key

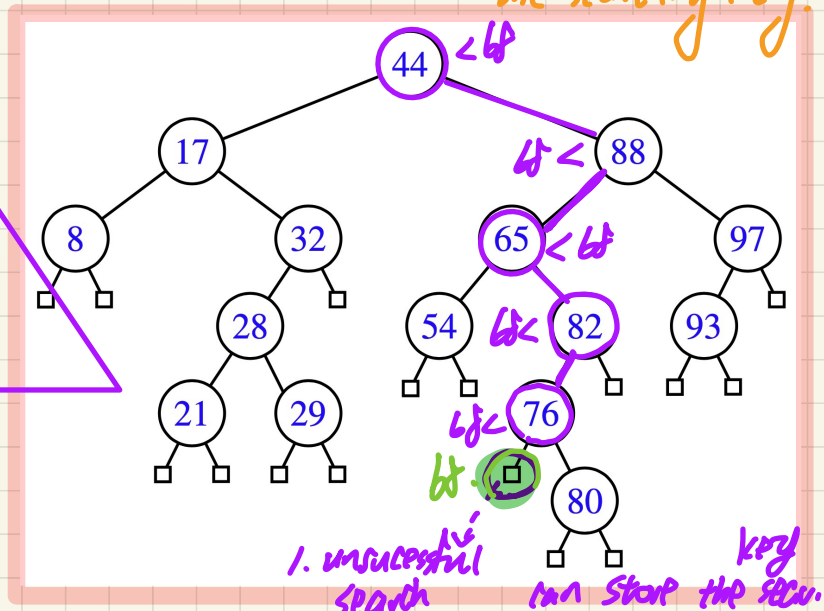
searching { successful → vn. int. node with matching key
unsuccessful → vn. ext. node that can store the searching key.



Search key 65



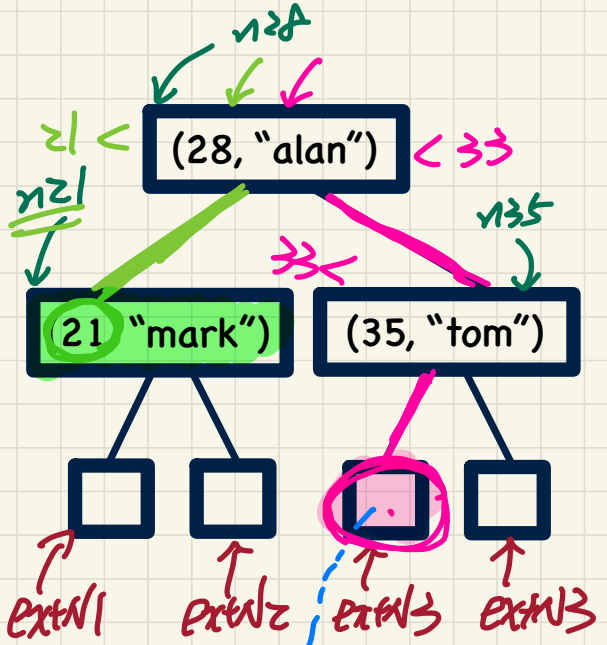
Search key 68



Tracing: Searching through a BST

```
@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* */
}
```



* any key between 28 (in-order traversal) and 35 can be stored here.
↳ sorting prop.

Running Time: Search on a BST

* Recall: given N nodes, $\log N \le h \le N$

internal or external

```
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;
}
```

root

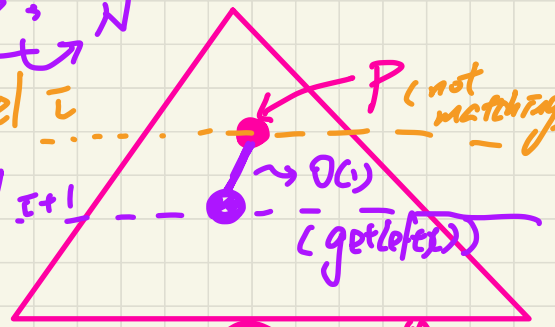
key to search

base case.

R1

R2

recursive cases



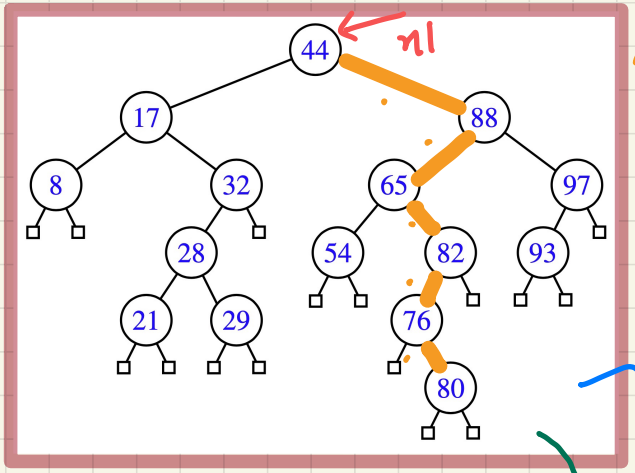
(A1) h elaborate.

(A2) log N (best case)

(A3) N (worst case)

worst case: unsuccessful search ⇒ have to get down to the bottom level.

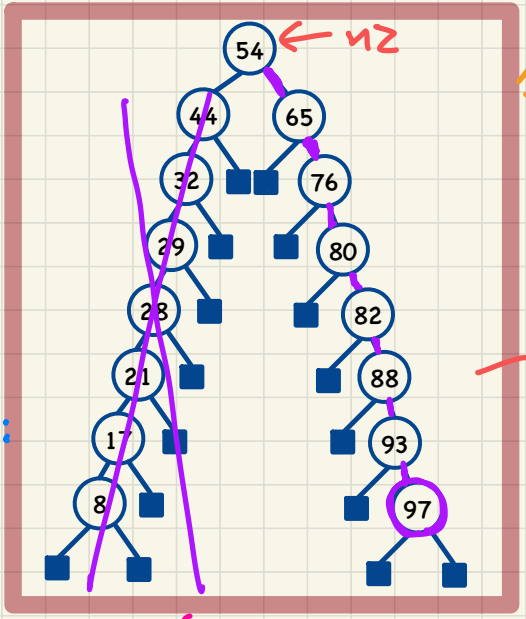
Binary Search: Non-Linear vs. Linear Structures



$N = 15$

$h = 5$

$\lesssim \log N$.



$N = 15$

$h = 7 \simeq \frac{N}{2}$

worst case: $O(N)$

in-order(n1) = in-order(n2) = a

search (97)

Best case of searching: $O(\log N)$

Corresponds to

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

Lecture

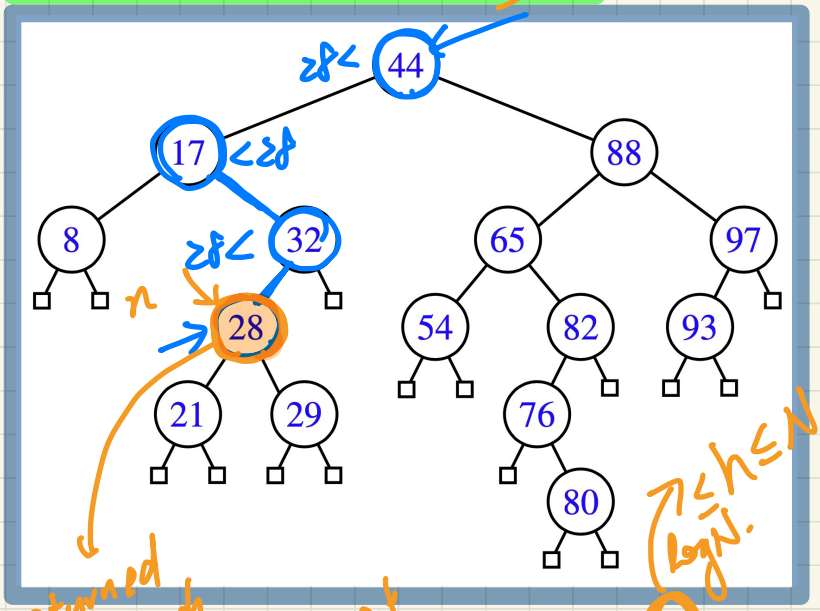
Binary Search Tree (BST)

Implementing a Generic BST in Java
Insertion

Visualizing BST Operation: Insertion

should be non-existing.
↳ otherwise, do some replacement

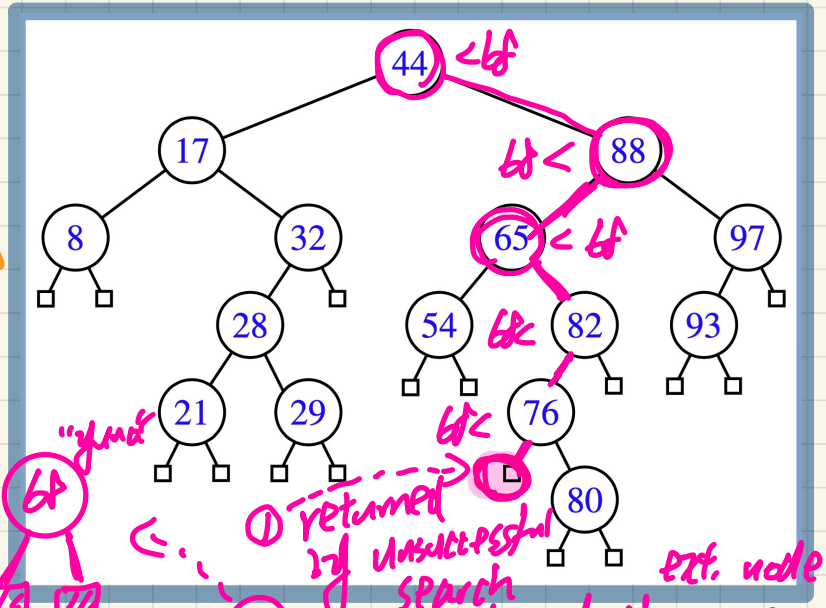
Insert Entry (28, "suyeon")



① returned by the search method
② set n's element to "suyeon"

RT: O(h) ↳ searching.
$\log N \le h \le N$

Insert Entry (68, "yuna")



① returned by unsuccessful search
② store key and ele. to this ext. node

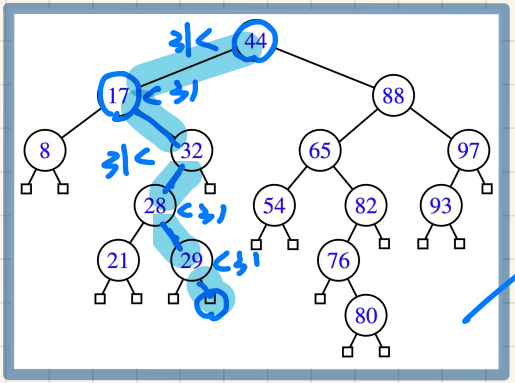
Lecture

Binary Search Tree (BST)

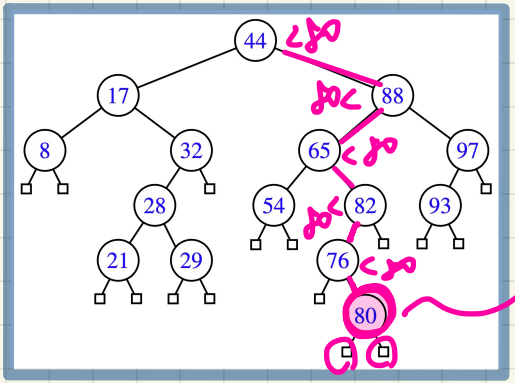
Implementing a Generic BST in Java
Deletion

Visualizing BST Operation: Deletion

Case 1: Delete Entry with Key 31



Case 2: Delete Entry with Key 80



Case 3: Delete Entry with Key 32