Transform and Conquer

Searching Problem

- Given a (multi)set S of values and a search key K, find an occurrence of K in S, if any
- Searching must be considered in the context of:
  - file size (internal vs. external)
  - dynamics of data (static vs. dynamic)
- Dictionary operations (dynamic data):
  - find (search)
  - insert
  - delete

Taxonomy of Searching Algorithms

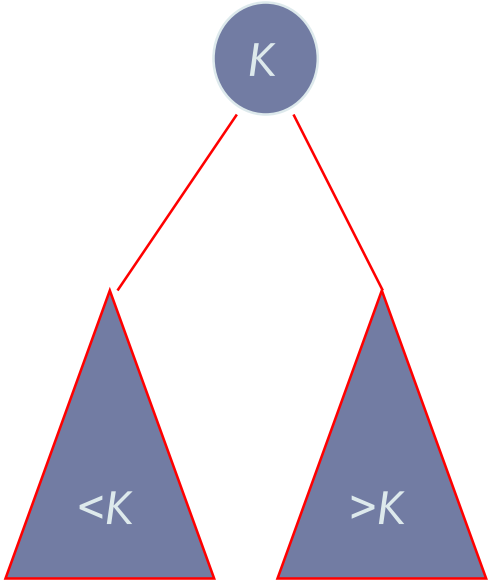
- List searching (array, vector, linked list)
  - sequential search
  - binary search
  - interpolation search

- Tree searching
  - binary search tree
  - binary balanced trees:
    - AVL trees, red-black trees
  - multi-way balanced trees:
    - 2-3 trees, 2-3-4 trees, B trees

- Hashing
  - open hashing (separate chaining)
  - closed hashing (open addressing)

Binary Search Tree

Arrange keys in a binary tree with the *binary search tree property*:

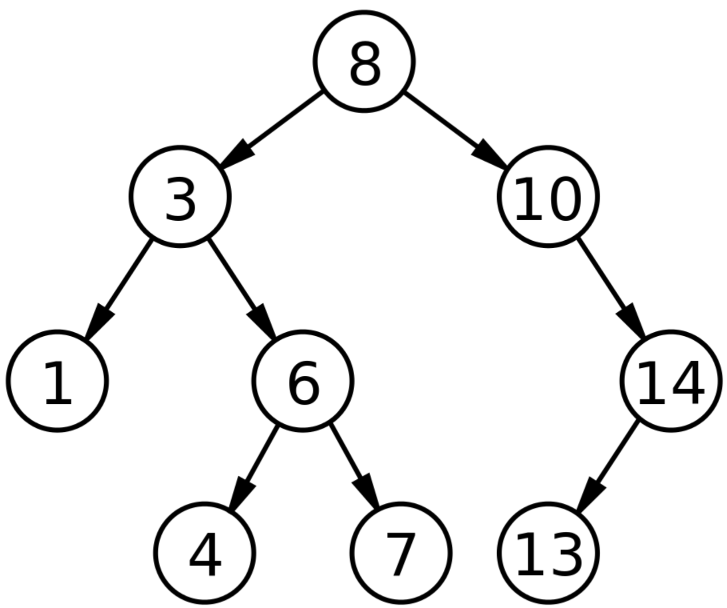


Dictionary Operations on Binary Search Trees

Searching

Start from root:

1. $\emptyset \rightarrow$ not found
2. $=$ key $\rightarrow$ found, return
3. $< key \rightarrow$ continue search in left sub-tree
4. $> key \rightarrow$ continue search in right sub-tree



Dictionary Operations on Binary Search Trees

Searching

Start from root:

1. $\emptyset \rightarrow$ not found
2. $=$ key $\rightarrow$ found, return
3. $< key \rightarrow$ continue search in left sub-tree
4. $> key \rightarrow$ continue search in right sub-tree

Search for 4

Searching

Start from root:

1. $\emptyset \rightarrow$ not found
2. $=$ key $\rightarrow$ found, return
3. $< key \rightarrow$ continue search in left sub-tree
4. $> key \rightarrow$ continue search in right sub-tree

Search for 4

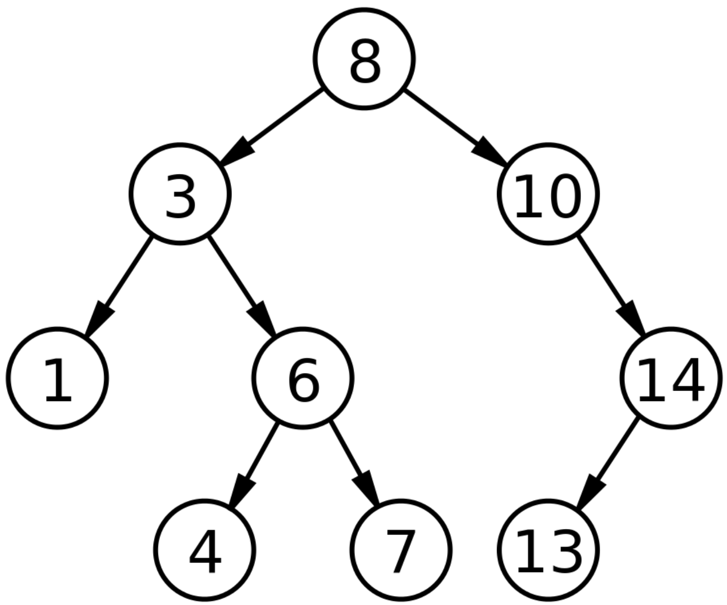
Dictionary Operations on Binary Search Trees

Searching

Start from root:

1. $\emptyset \rightarrow$ not found
2. $=$ key $\rightarrow$ found, return
3. $< key \rightarrow$ continue search in left sub-tree
4. $> key \rightarrow$ continue search in right sub-tree

Search for 11



Dictionary Operations on Binary Search Trees

Searching:

Start from root:

1. $\emptyset \rightarrow$ not found
2. $=$ key $\rightarrow$ found, return
3. $< key \rightarrow$ continue search in left sub-tree
4. $> key \rightarrow$ continue search in right sub-tree

Efficiency:

Dictionary Operations on Binary Search Trees

Searching:

Start from root:

1. $\emptyset \rightarrow$ not found
2. $=$ key $\rightarrow$ found, return
3. $< key \rightarrow$ continue search in left sub-tree
4. $> key \rightarrow$ continue search in right sub-tree

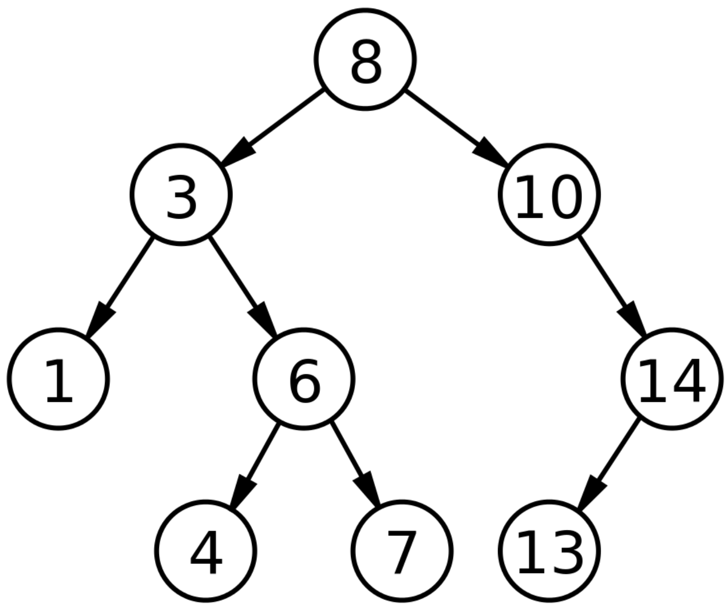
Efficiency:

- worst case efficiency: $\theta(n)$
- average case efficiency: $\theta(\log n)$

Dictionary Operations on Binary Search Trees

Insertion:

1. search for the key
2. insert at leaf where search terminated

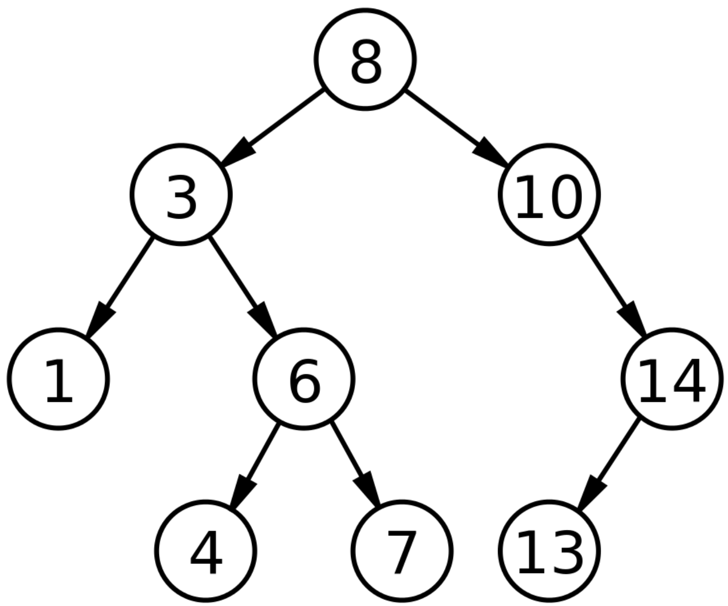


Dictionary Operations on Binary Search Trees

Insertion:

1. search for the key
2. insert at leaf where search terminated

Insert 9

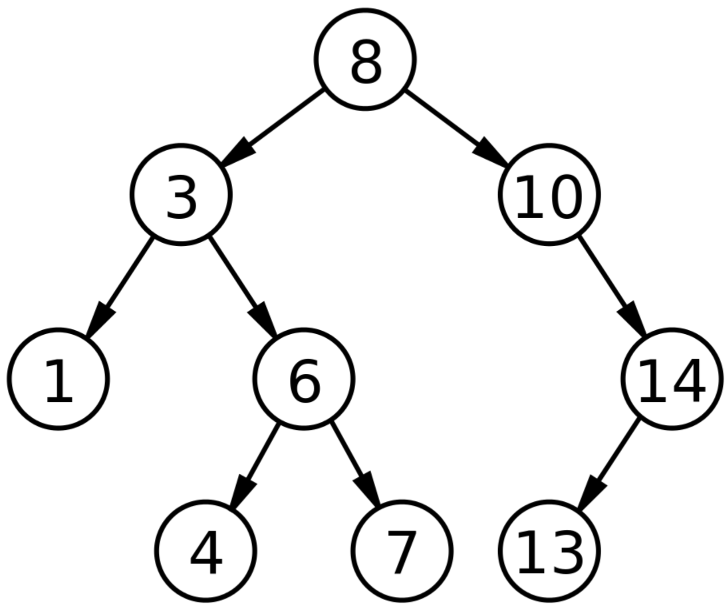


Dictionary Operations on Binary Search Trees

Insertion:

1. search for the key
2. insert at leaf where search terminated

Insert 5



Dictionary Operations on Binary Search Trees

Insertion:

1. search for the key
2. insert at leaf where search terminated

Efficiency:

Dictionary Operations on Binary Search Trees

Insertion:

1. search for the key
2. insert at leaf where search terminated

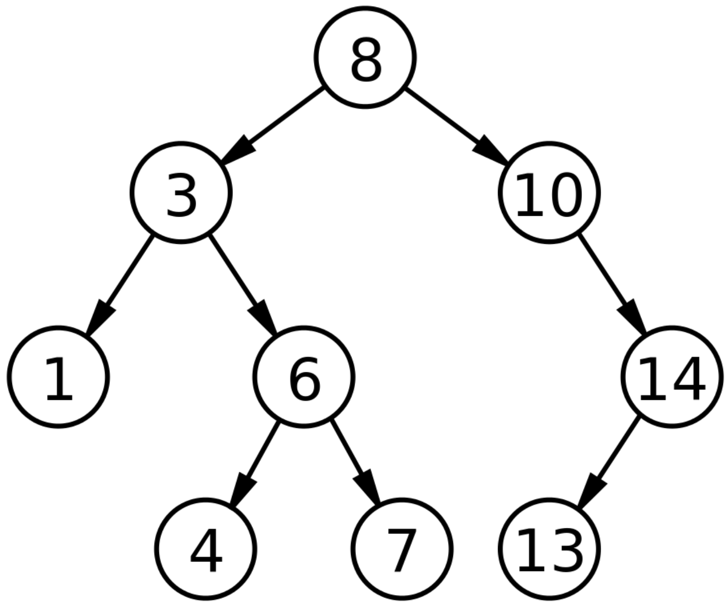
Efficiency:

- worst case efficiency: $\theta(n)$
- average case efficiency: $\theta(\log n)$

Dictionary Operations on Binary Search Trees

Deletion:

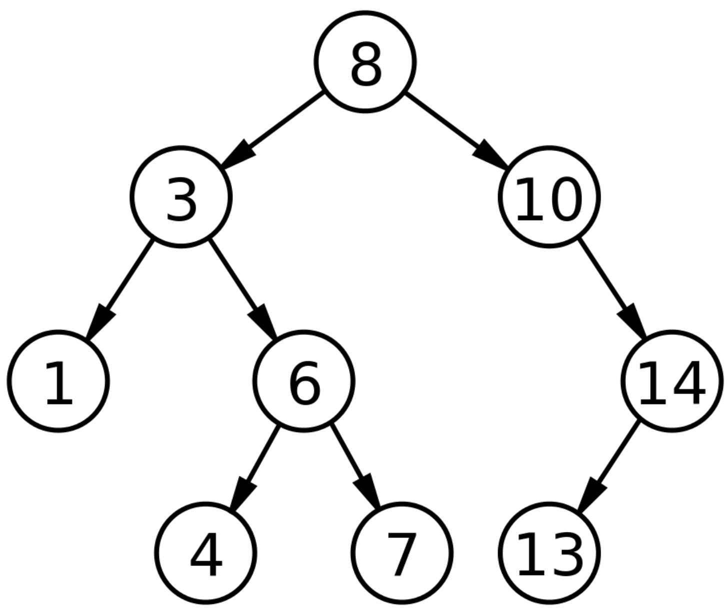
- deleting key at a leaf
- deleting key at node with single child
- deleting key at node with two children



Dictionary Operations on Binary Search Trees

Deletion: deleting key at a leaf

1. Search for the key
2. Remove it

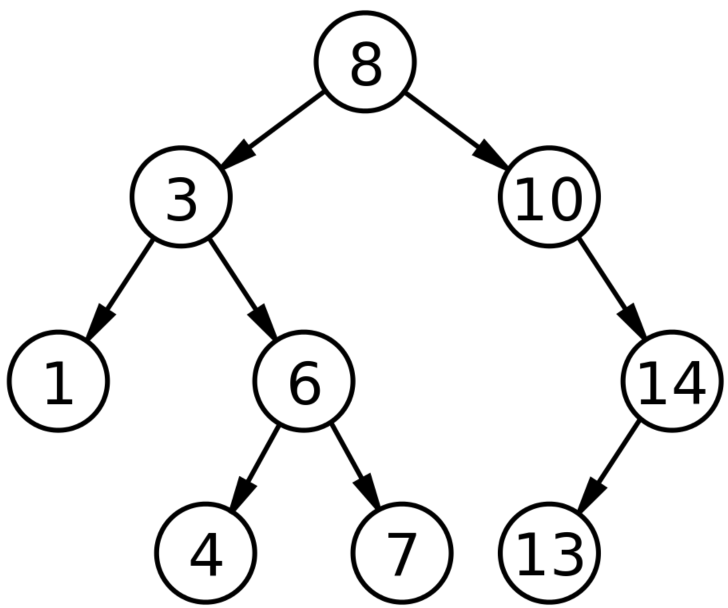


Dictionary Operations on Binary Search Trees

Deletion: deleting key at a leaf

1. Search for the key
2. Remove it

Remove 7



Dictionary Operations on Binary Search Trees

Deletion: deleting key at a leaf

1. Search for the key
2. Remove it

Efficiency:

Dictionary Operations on Binary Search Trees

Deletion: deleting key at a leaf

1. Search for the key
2. Remove it

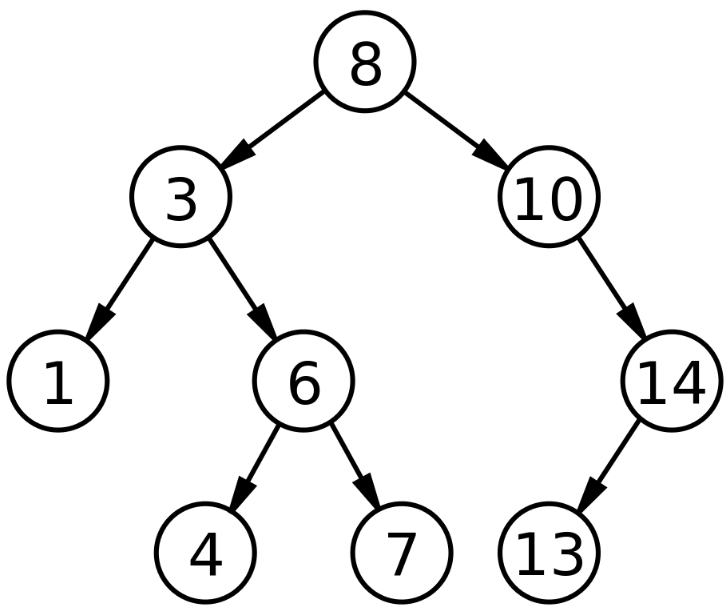
Efficiency:

- worst case efficiency: $\theta(n)$
- average case efficiency: $\theta(\log n)$

Dictionary Operations on Binary Search Trees

Deletion: deleting key at node with single child

1. Search for the key
2. Replace node with it's child

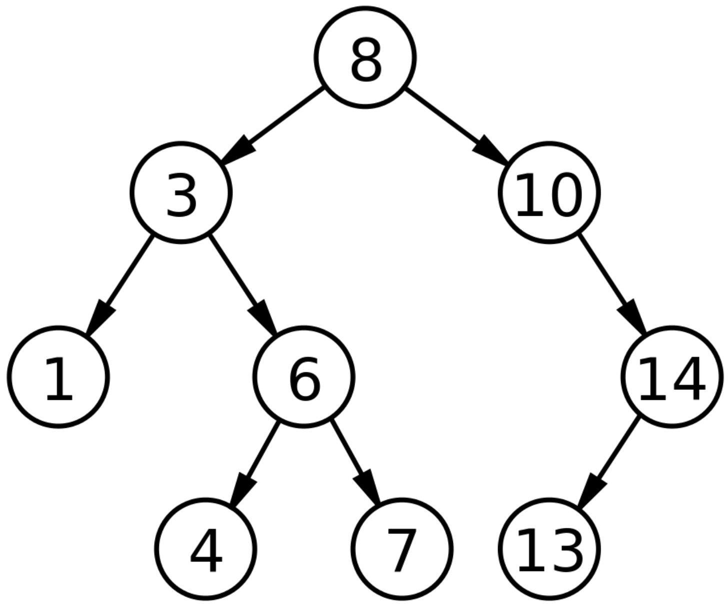


Dictionary Operations on Binary Search Trees

Deletion: deleting key at node with single child

1. Search for the key
2. Replace node with it's child

Remove 10



Dictionary Operations on Binary Search Trees

Deletion: deleting key at node with single child

1. Search for the key
2. Replace node with it's child

Efficiency:

Dictionary Operations on Binary Search Trees

Deletion: deleting key at node with single child

1. Search for the key
2. Replace node with it's child

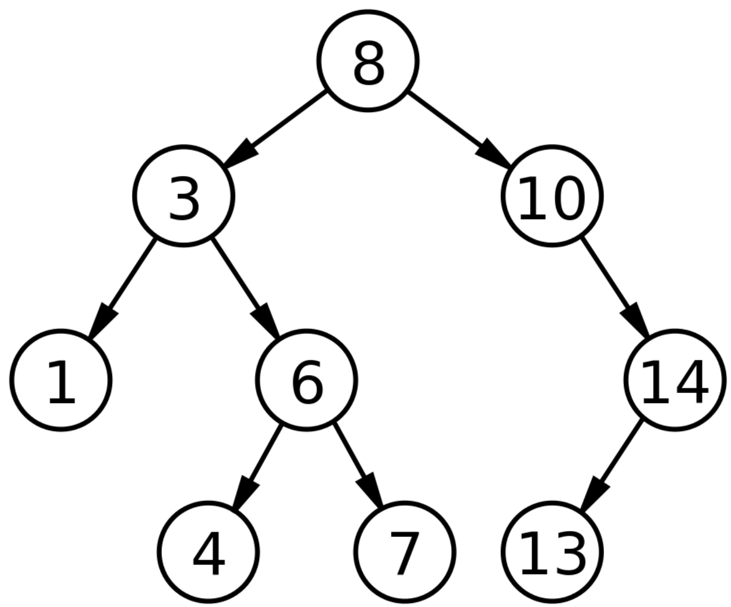
Efficiency:

- worst case efficiency: $\theta(n)$
- average case efficiency: $\theta(\log n)$

Dictionary Operations on Binary Search Trees

Deletion: deleting key at node with two children

1. Search for the Key
2. Traverse tree in-order
3. Find successor of the node
4. Replace key with its successor
5. Delete the successor

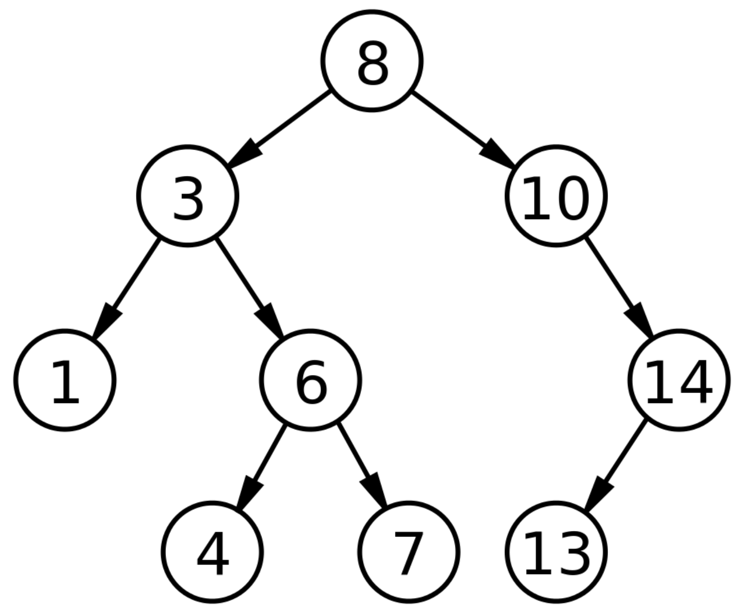


Dictionary Operations on Binary Search Trees

Deletion: deleting key at node with two children

1. Search for the Key
2. Traverse tree in-order
3. Find successor of the node
4. Replace key with its successor
5. Delete the successor

Remove 3

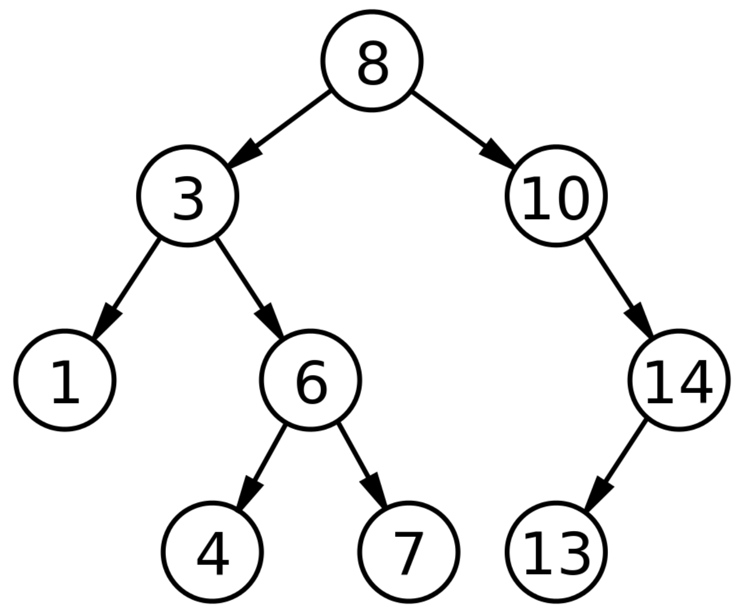


Dictionary Operations on Binary Search Trees

Deletion: deleting key at node with two children

1. Search for the Key
2. Traverse tree in-order
3. Find successor of the node
4. Replace key with its successor
5. Delete the successor

Remove 8



Dictionary Operations on Binary Search Trees

Deletion: deleting key at node with two children

1. Search for the Key
2. Traverse tree in-order
3. Find successor of the node
4. Replace key with its successor
5. Delete the successor

Efficiency:

Dictionary Operations on Binary Search Trees

Deletion: deleting key at node with two children

1. Search for the Key
2. Traverse tree in-order
3. Find successor of the node
4. Replace key with its successor
5. Delete the successor

Efficiency:

- worst case efficiency: $\theta(n)$
- average case efficiency: $\theta(\log n)$

Dictionary Operations on Binary Search Trees

- Searching – straightforward
- Insertion – search for key, insert at leaf where search terminated
- Deletion – 3 cases:
  - deleting key at a leaf
  - deleting key at node with single child
  - deleting key at node with two children

- Efficiency:
  - depends on the tree's height: $\lfloor \log_2 n \rfloor \leq h \leq n - 1$,
  - with height average (random files) be about $3\log_2 n$

- Thus all three operations have
  - worst case efficiency: $\theta(n)$
  - average case efficiency: $\theta(\log n)$

- Bonus: in-order traversal produces sorted list

Dictionary Operations on Binary Search Trees

- Bonus:

- in-order traversal produces sorted list

- With different order of inserts, you get different BSTs