

Set 09: Ordered Dictionary Abstract Data Types: Sorted Arrays and BST

CS240: Data Structures and Data Management

Jérémy Barbay

Outline

Introduction to Ordered Dictionary ADTs

Sorted Arrays

Motivations

Binary Search

One Sided Binary Search

Binary Search Trees

Find

Insert

Remove

Introducing Order in Dictionaries

Motivations

- ▶ What if we access keys uniformly?
- ▶ Use ordering on keys to speed up the Find operator.
- ▶ How expensive is it for the operator Insert?

Dictionary ADT

- ▶ Container of key-element pairs,
where the keys are totally ordered.
- ▶ Required operations (as for general Dictionaries):
 - ▶ `insert(k,e)`,
 - ▶ `remove(k)`,
 - ▶ `find(k)`,
 - ▶ `isEmpty()`
- ▶ Now also supports:
 - ▶ `closestKeyBefore(k)`,
 - ▶ `closestElemAfter(k)`

This corresponds to dictionaries in the Comparison Model.

Ordered Dictionary ADTs and their DS

- ▶ Array
- ▶ Binary Search Tree (BST)
- ▶ Sequence (Skip Lists)
- ▶ AVL
- ▶ (2, 4) Trees
- ▶ B-Trees

Diferent solutions to different problems.

References:

- ▶ Goodrich and Tamassia: pp. 140-151
- ▶ Cormen, Leisersen, Rivest, Stein: 253-264

Outline

Introduction to Ordered Dictionary ADTs

Sorted Arrays

- Motivations
- Binary Search
- One Sided Binary Search

Binary Search Trees

- Find
- Insert
- Remove

Sorted Arrays as Dictionaries

Motivations

- ▶ Maintain an array **sorted** by keys:
- ▶ Use ordering on keys to speed up the Find operation:
- ▶ How expensive is it for the operators Insert and Remove?

Binary Search

Naive Implementation

```
BINARY SEARCH( $x, A, l, r$ )
  if  $l > r$  then
    return false
  else
     $m \leftarrow \lfloor \frac{l+r}{2} \rfloor$ 
    if  $A[m] = x$  then
      return true
    else if  $A[m] < x$  then
      return BINARY SEARCH( $x, A, m, r$ )
    else
      return BINARY SEARCH( $x, A, l, m$ )
    end if
  end if
```

In the worst case,

- ▶ each recursive call performs comparisons.
- ▶ the entire search performs comparisons.

Binary Search

Better Implementation

```

BINARY SEARCH( $x, A, l, r$ )
  if  $l = r$  then
    return ( $A[l] = x$ )
  end if
   $m \leftarrow \lfloor \frac{l+r}{2} \rfloor$ 
  if  $A[m] > x$  then
    return BINARY SEARCH( $x, A, m, r$ )
  else
    return BINARY SEARCH( $x, A, l, m$ )
  end if
    
```

- Note that $l \leq m < r$.
- In the worst case, each recursive call performs 2 comparisons, and the entire search performs only 2 comparisons.

Binary Search

Non Recursive Implementation

```

BINARY SEARCH( $x, A, l, r$ )
  while  $l < r$  do
     $m \leftarrow \lfloor \frac{l+r}{2} \rfloor$ 
    if  $A[m] > x$  then
       $l \leftarrow m$ 
    else
       $r \leftarrow m$ 
    end if
  end while
  return ( $A[l] = x$ )
    
```

- Note that $l \leq m < r$.
- Removing recursion makes it faster in practice.
- The Worst Case complexity is still $\log_2 n$.

Average Performance of Binary Search

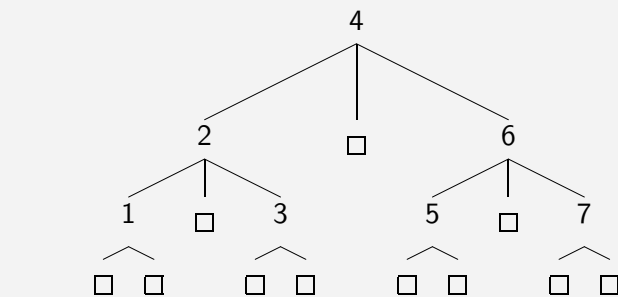
Exercise:

Consider searching in the array

1	2	3	4	5	6	7
---	---	---	---	---	---	---

. Suppose that x takes a value randomly and uniformly chosen among the elements of A . What is the average number of comparison performed by the first implementation?

Example:



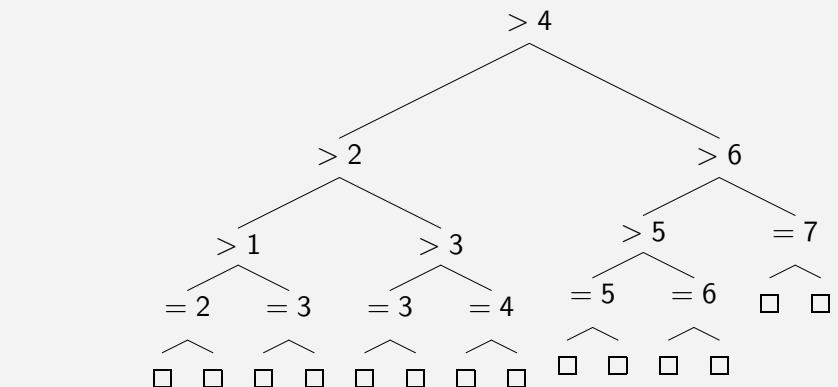
The algorithm performs 2 comparisons per node.

Average Performance of Binary Search (end)

Exercise:

What is the average number of comparison performed by the second implementation (exact, not asymptotic)?

Example:



The algorithm performs 1 comparison per node.

One Sided Binary Search

What if d elements have to be searched in the same sorted array?

- ▶ Naive algorithm performs d binary searches and comparisons.
- ▶ Another possibility is to sort the d elements in comparisons and search for them using **one sided doubling search** (also called **gallop**, or **doubling search**).

Example:

Consider searching for 18 and 203 in the array

1	4	5	9	15	16	17	...	369210
---	---	---	---	----	----	----	-----	--------

Complexity of One Sided Binary Search

Theorem

Given an element x and a sorted array A , One sided binary search finds its insertion rank p such that $A[p] \leq x < A[p+1]$ after $2\lceil \lg p \rceil + 1$ comparisons.

Proof.

- ▶ After the i th galloping comparison,
 - ▶ $1 + 2 + 4 + \dots + 2^{i-1} = 2^i$ elements have been “eliminated”.
 - ▶ the interval considered is of size 2^i , and a binary search on it would perform $1 + i$ comparisons.
- ▶ The algorithm finds p after the $i = \lceil \lg p \rceil$ th galloping comparison. □

Use of One Sided Binary Search

Theorem

Given an increasing sequence of elements x_1, \dots, x_d and a sorted array A , there is an algorithm which checks if those elements are in A in only $O(d \lg(n/d))$ comparisons.

Proof.

Let $p_0 = 0$, call p_i the insertion rank of x_i in A , and $q_i = p_i - p_{i-1}$ the distance to the last one. Using one sided binary search, each p_i is found using $2\lceil \lg q_i \rceil + 1$ comparisons, hence a total of

$$2 \sum_i (\lceil \lg q_i \rceil + 1) \leq 4d + \sum_i \lg q_i.$$

We can simplify more using the concativity of the function \lg :

$$\sum_i \lg q_i \leq d \lg((\sum_i q_i)/d) \leq d \lg(n/d).$$

Hence the result, as $4d + d \lg(n/d) \in O(d \lg(n/d))$. □

Short Summary

- ▶ An order on the elements helps.
- ▶ Arrays are not practical for insertion.
- ▶ Many variants of Binary Search, and many implementations.

Outline

Introduction to Ordered Dictionary ADTs

Sorted Arrays

Motivations

Binary Search

One Sided Binary Search

Binary Search Trees

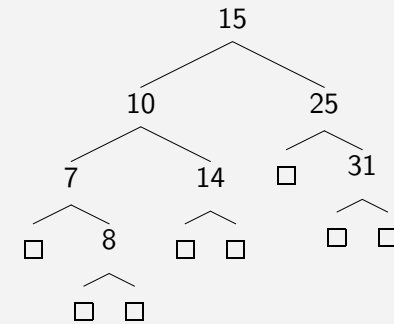
Find

Insert

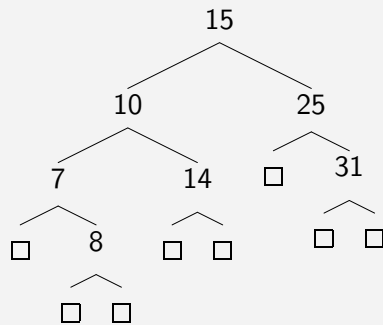
Remove

Binary Search Tree

- ▶ A binary tree storing (k, e) pairs at the internal nodes such that
 - ▶ All keys in nodes of left subtree are $< k$
 - ▶ All keys in nodes of right subtree are $> k$
- ▶ A set merely stores the keys (example below)
- ▶ External nodes are only placeholders and often not shown
- ▶ $\Theta(n)$ additional space usage

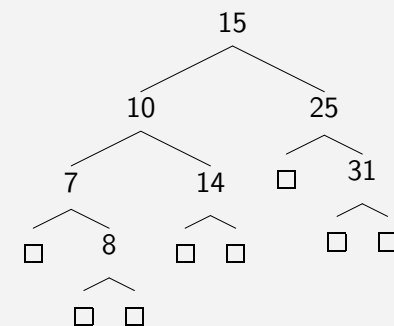


Find



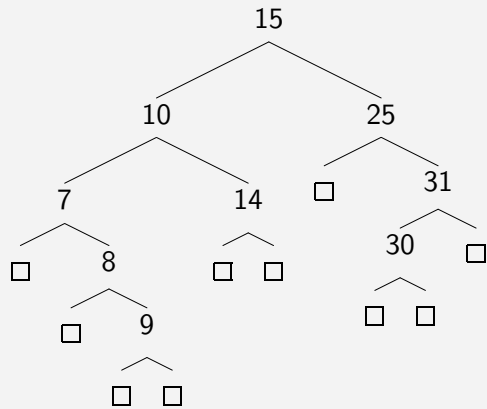
- ▶ Compare root's key to look-up key, K , and possibly traverse subtree
- ▶ If found return the node (or data associated with it)
- ▶ If not found return the external node where it **would** have been found
- ▶ Worst-case running time?

Insert



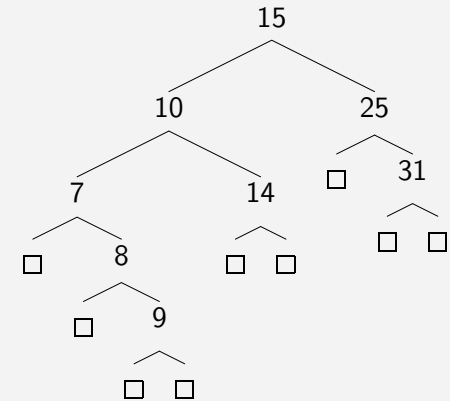
- ▶ Perform a search first
- ▶ Insert at the external node if one is returned
- ▶ **Example:** Insert(9) and Insert(30)
- ▶ Worst-case running time?

Remove



- ▶ Perform a search first
- ▶ Remove the internal node if one is returned
- ▶ Three cases:
 - ▶ **2 external children:** Remove(30)
 - ▶ **1 external child:** Remove(7)

Remove



- ▶ If node has two internal children replace contents with in-order predecessor, or in-order successor
- ▶ Remove the emptied node of the in-order predecessor, or in-order successor
- ▶ **Example:** Remove(15)
- ▶ Worst-case running time?

Two Answers

