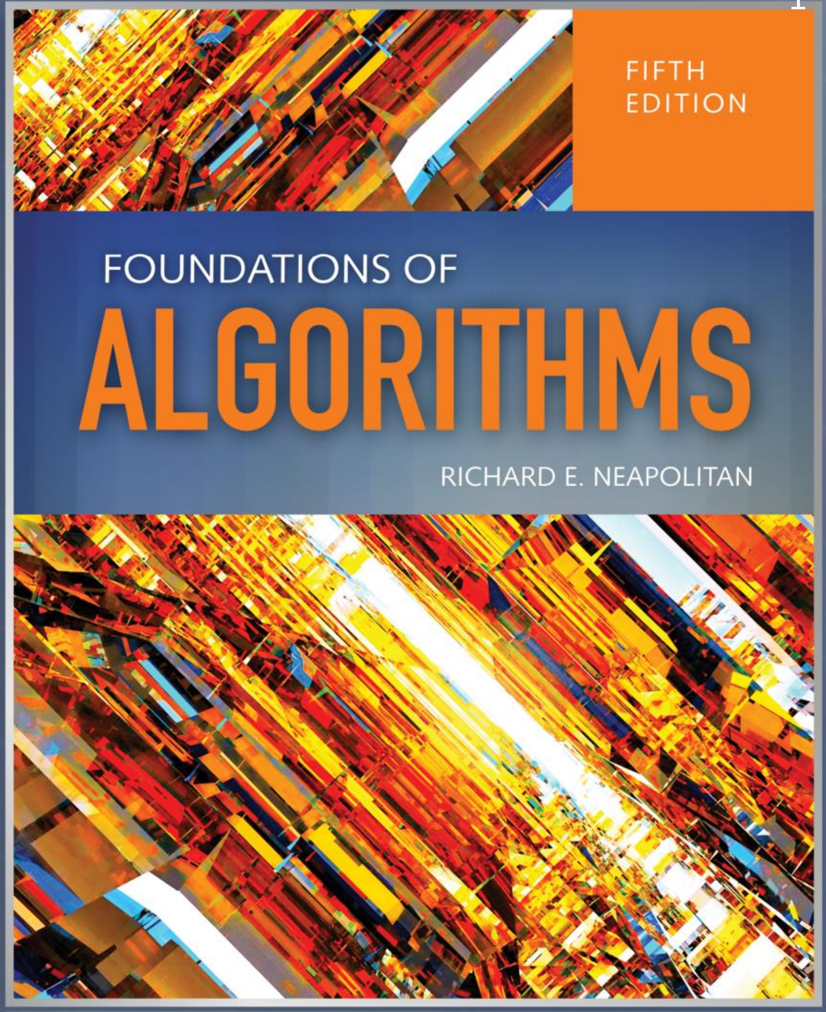
More Computational Complexity: The Searching Problem

Chapter 8



Objectives

- Establish a lower bound on searching by comparison of keys of n distinct keys for key x.
- Prove binary search is optimal
- Apply interpolation search to evenly distributed data
- Differentiate between static and dynamic searching
- Establish the average binary search tree search time
- Define 3-2 Tree

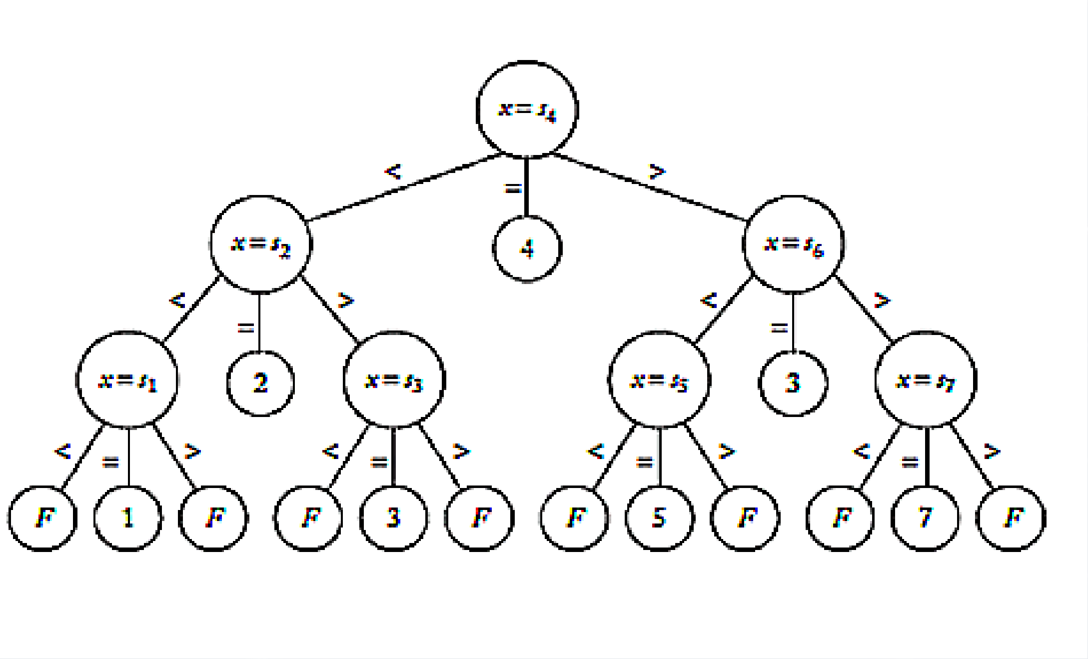
Objectives

- Determine lower bounds on finding largest and smallest keys
- Define a linear-time algorithm to find the kth smallest key in an array of n distinct keys

Binary Search

- Algorithm 2.1
- $W(n) = \lfloor \lg n \rfloor + 1$
- Establish binary search algorithm is optimal
- Establish a lower bounds for searching only by comparison of keys

Figure 8.1



Decision Tree Binary Search

- Every algorithm that searches for key x in an array of n keys has a corresponding pruned, valid decision tree
- Leaf represents a point where algorithm stops and reports index of x or failure
- Every internal node represents a comparison
- Valid if for each possible outcome, there is a path from the root to a leaf reporting that outcome
- Pruned if every leaf is reachable

Pruned, valid Decision Tree

- 3 different results: >, =, <
- Decision node can have at most three children
- Equality is a leaf returning index
- At most 2 children can be comparison nodes
- Therefore, # comparisons is a **binary tree**

Establish a lower bound on the number of comparisons

- Every node reachable: worst case comparison nodes on longest path
- Number of nodes on the longest path from the root to a leaf in the binary tree consisting of comparison of nodes
- Number of comparisons is the depth+1

Lemma 8.1

- Establish a lower bound on the depth of the binary tree consisting of comparison nodes
- If n is the number of nodes in a binary tree and d is the depth:
  - $d >= \lfloor \lg n \rfloor$

Proof Lemma 8.1

- At most one root, At most 2 nodes with depth 1, At most 2^2 nodes with depth 2, . . . , At most 2^d nodes with depth d
- $n <= 1 + 2 + 2^2 + . . . + 2^d$
- A3:
- $=> n <= 2^{d+1} - 1$
- $=> n< 2^{d+1}$
- Take lg of both sides
- $\lg n < \lg 2^{d+1}$
- $=> \lg n < (d+1)\lg 2 = d+1$
- $=> \lfloor \lg n \rfloor <= d$

A3

$$\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$$

Lemma 8.2

- To be a pruned, valid decision tree for searching n distinct keys for a key x, the binary tree consisting of the comparison nodes must contain at least n nodes

Steps of Proof by Contradiction

1. Show that every si must be in at least one comparison node
2. Show that since every si must be in a comparison node there must be n comparison nodes

Theorem 8.1

- Any deterministic algorithm that searches for a key x in an array of n distinct keys only by comparisons of keys must in the worst case do at least $\lfloor \lg n \rfloor + 1$ comparisons of keys

Proof Theorem 8.1

- Given a pruned, valid decision tree for the algorithm that searches n distinct keys for a key x
- Worst-case number of comparisons is the umber of nodes on the longest path from the root to a leaf in the binary tree of comparison nodes: $d + 1$
- By Lemma 8.2, the binary tree has at least n nodes
- By Lemma 8.1, $d >= \lfloor \lg n \rfloor$

Interpolation Search

- Given an application, data is evenly distributed in the search array
- Instead of starting search at mid, make decision based on value of x
- Use linear interpolation to determine where x should be located
- mid =low + $\lfloor$(x – S[low])/(S[high]-S[low]) x (high – low)$\rfloor$

Volatility of data

- Static Searching: records are added to the file one at a time. Once file established, records never added or deleted.
  - Array appropriate storage/search structure
- Dynamic Searching: records frequently added/deleted (e.g. airline reservation system)
  - Binary Search Tree

Binary Search Tree

- Tree of items from an ordered set such that
  - Each node contains one key
  - Keys in the left subtree of a given node are less than or equal or the key in that node
  - Keys in the right subtree of a given node are greater than or equal to the key in that node

Figure 8.5

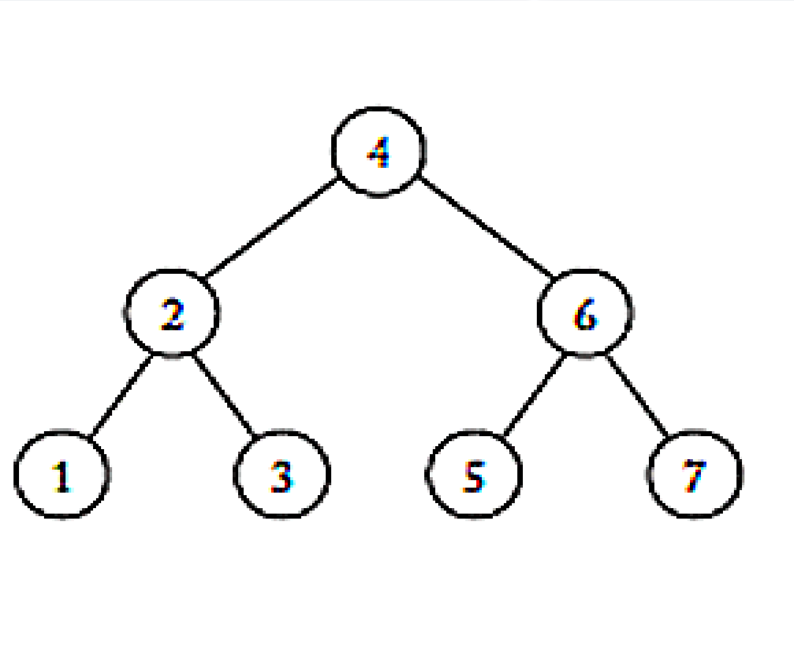
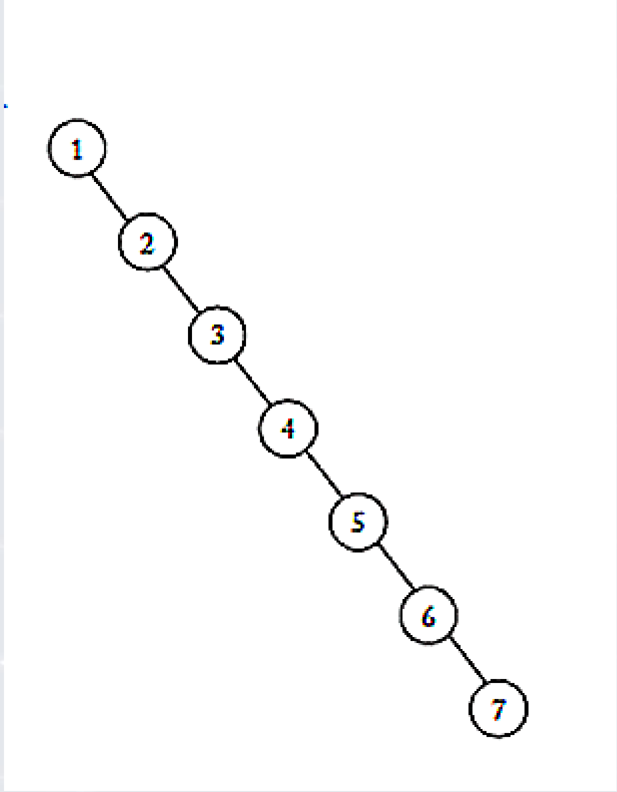


Figure 8.6



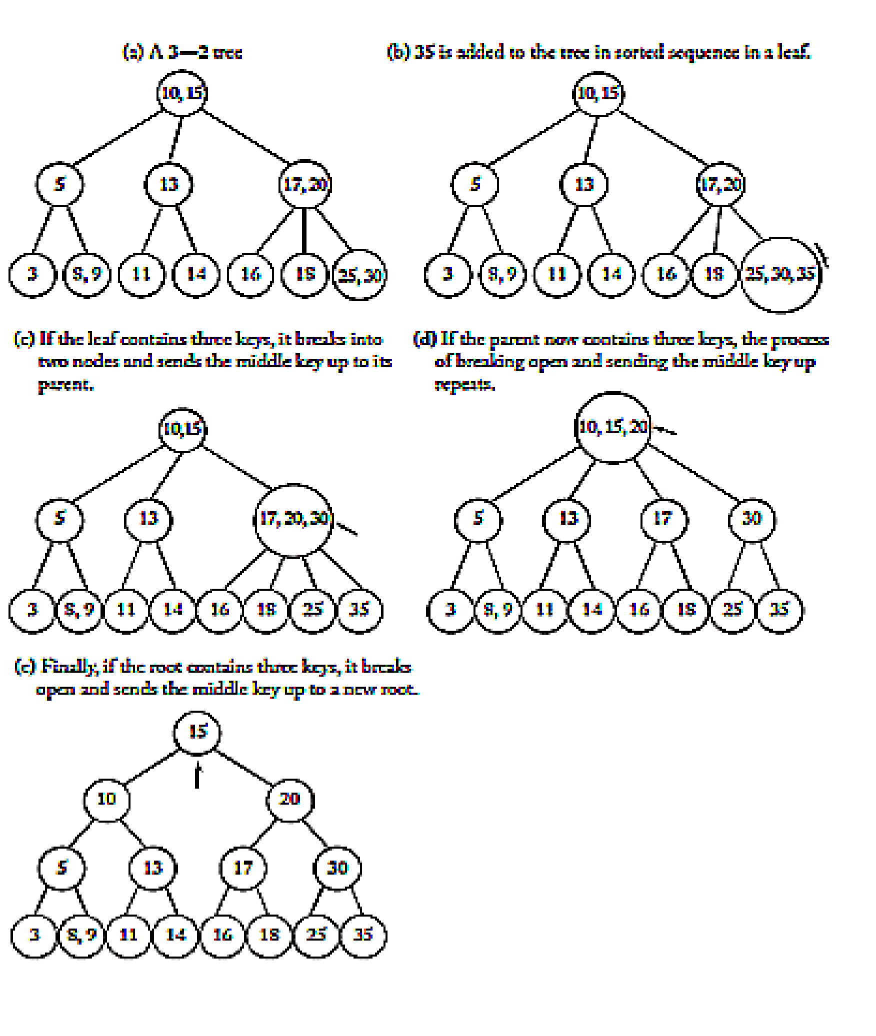
In-order traversal

- Visit all nodes in the left subtree using in-order traversal
- Visit root
- Visit all nodes in the right subtree using in-order traversal
- Produces sorted order of keys

Theorem 8.3

- Assume all input are equally probable and that the search key x is equally probable to be any of the n keys, the average search time over all inputs containing n distinct keys using binary search trees is given approximately by $A(n) \approx 1.38 \lg n$
- Theorem 8.3 does not mean average for a given input is $1.38 \lg n$
- Theorem 8.3 is the average search time over all inputs containing n keys

Figure 8.7


(a) A 3—2 tree
(b) 35 is added to the tree in sorted sequence in a leaf.
(c) If the leaf contains three keys, it breaks into two nodes and sends the middle key up to its parent.
(d) If the parent now contains three keys, the process of breaking open and sending the middle key up repeats.
(e) Finally, if the root contains three keys, it breaks open and sends the middle key up to a new root.
10, 15
5
13
17, 20
3
8, 9
11
14
16
18
25, 30
25, 30, 35
17, 20, 30
25
35
10, 15, 20
17
30
15
10
20

Selection Problem

- Find kth largest or kth smallest key in a list of n keys
- Algorithm 8.2 Find Largest Key
  - T(n) = n-1
- Theorem 8.7: Any deterministic algorithm that can find the largest of n keys in every possible input only by comparisons of keys must in every case do at least n-1 comparisons of keys
- Proof by contradiction

Find the smallest and largest key

- Algorithm 8.3
- Worst case is when S[1] is the smallest key, second comparison is done for all i
- $W(n) = 2(n-1)$
- Improve?

Find smallest and largest keys by pairing keys

- Assume n is even
- Pair i_1 and i_2 , i_3 and i_4 , . . . , i_j and i_{j+1} ,. . . , i_{n-1} and i_n
- Find the largest of the pair: i_l and compare it with the largest so far
- Compare the other key of the pair with the smallest so far
- Iterate loop by 2 instead of 1
- n is even: $T(n) = 3n/2 – 2$
- n is odd: $T(n) = 3n/2- 3/2$

Theorem 8.8

- Any deterministic algorithm that can find both the smallest and the largest of keys in every possible input only by comparisons of keys must in the worst case do at least:
  - 3n/2 – 2 if n is even
  - 3n/2 – 3/2 if n is odd
- Proof involves use of an adversary argument

Algorithm 8.5

- Find the kth smallest key in an array S of n distinct keys
- Could sort the array in n lg n time and take the kth smallest
- Use Algorithm 2.7, partition from quicksort
- Recursively partition the left sub-array if k is less than pivotpoint
- Recursively partition the right sub-array if k is greater than pivotpoint

Algorithm 8.5

- $W(n) = n(n-1)/2$
- $A(n) \in \theta(n)$
- Can quadratic time worst case be prevented?

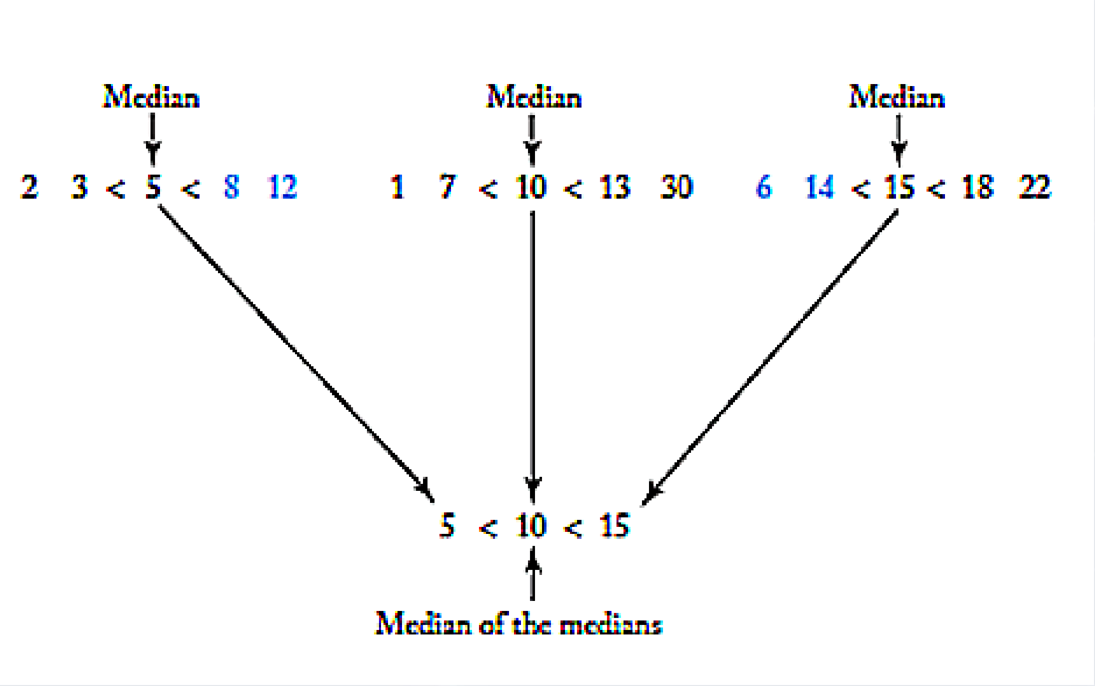
Pivotpoint

- Split array in the middle
- Input cut in half each recursive call
- Median of n distinct keys (precise if n is odd)
- Half of the keys smaller
- Half of the keys larger

Assume n is an odd multiple of 5

- Divide n keys into n/5 groups of keys each containing 5 keys
- Find the median of each of the groups directly
- The median of 5 items can be found with 6 comparisons (see exercises)
- Call selection to find the median of the n/5 medians
- Median of n/5 medians not necessarily the median of the n elements but will be close
- pivotitem is the median of the medians
- Partition around pivotitem – optimal pivotpoint

Figure 8.12


Median
Median
Median
2 3 < 5 < 8 12
1 7 < 10 < 13 30
6 14 < 15 < 18 22
5 < 10 < 15
Median of the medians

Keys n is an odd multiple of 5

- Number of keys known to be on one side of the median of the medians: $\frac{1}{2}[n - 1 – 2(n/5 – 1)]$
- Number of keys that could be on either side of the median of the medians: $2(n/5 – 1)$
- At most $\frac{1}{2}[n - 1 – 2(n/5 – 1)] + 2(n/5 – 1) = 7n/10 – 3/2$ keys on one side of the median of the medians

Worst-Case Time Complexity (Selection Using the Median)

- Basic operation: comparison of S[i] with pivotitem in partition2
- Input size: n
- Recurrence assumes n is an odd multiple of 5 (holds for n in general)
- Time in function selection2 when called from selection 2
  - At most 7n/10 – 3/2 keys on one side of the pivotpoint (worst case number of keys in call)

Worst-Case Time Complexity (Selection Using the Median)

- Time in selection2 when called from partition2
  - Number of keys is n/5 (n/5 medians)

Worst-Case Analysis Continued

- Number of comparisons required to find medians:
  - 6 comparisons (see exercises)
  - $n/5$ groups of 5 => total number of comparisons $6n/5$
- Number of comparisons required to partition the array => n
- $W(n) = W(7n/10 – 3/2) + W(n/5) + 6n/5 + n$
- $W(n) \approx W(7n/10) + W(n/5) + 11n/5$
- Recurrence does not indicate any obvious solution

Constructive Induction used to obtain Candidate Solution

- Suspect W(n) is linear
- Assume W(m) <= cm for all m < n and for some constant c
- Recurrence implies
  - $W(n) \approx W(7n/10) + W(n/5) + 11n/5$
  - $<= c(7n/5) + c(n/5) + 11n/5$
- To conclude W(n) <= cn solve
  - $<= c(7n/5) + c(n/5) + 11n/5$
  - $22 <= c$
- $W(n) \in \theta(n)$