Searching

- Find an element in a collection in the main memory or on the disk
  - collection: $(K_1,I_1),(K_2,I_2)...(K_N,I_N)$
  - given a query (I,K) locate (I_i,K_i): $K_i = K$
- Primary key K_i: identity of record
- Secondary key: can be repeated
- The search can be successful or unsuccessful

Searching Methods

- Sequential: data on lists or arrays
  - O(N) time, may be unacceptably slow
- Indexed search:
  - tree indexing: data in trees
  - hashing or direct access : data on tables
- Indexing requires preprocessing and extra space

Important Factors

- Ordered or unordered data
- Known or unknown data distribution
  - some elements are searched more frequently
- Data in main memory or disk
  - time depends on algorithmic steps or disk accesses
- Dynamic (or static) data collections
  - Insertions & deletions are allowed (or not allowed)
- Types of search operations allowed
  - random queries: search for records with key = k
  - range queries: search for records $key_{low} <= k <= key_{high}$

Unordered Sequences

- Lists or arrays of N elements

elements	10	9	2	15	4	8	1

- Number of comparisons: $\overline{X} = \sum_{i=1}^{N} p_i x_i$
  - p_i: prob. to search for the *i*-th element
  - x_i: number of comparisons when searching for the *i*-th element

Equally Probable Elements

- Cost of successful search

$$\bar{x} = \sum_{i=1}^{N} p_i x_i = \frac{1}{N} \sum_{i=1}^{N} x_i = \frac{1}{N}(1 + 2 + \ldots + N) = \frac{N+1}{2}$$

- Cost to search for an element which may or may not be in the array
  - if p_e: probability to search for the i-th element

$$\bar{x} = \frac{N+1}{2} p_e + (1 - p_e) N$$

Other Cases

- If $p_1 >= p_2 >= \ldots >= p_N$: move elements with higher probabilities to the front

element	10	9	2	15	4	8	1
p_i	0.2	0.1	0.25	0.15	0.05	0.23	0.02

- If the probabilities are not known it is likely that some elements are searched more frequently than others

I. Move to Front

- Move the element to the front
  - e.g., if the user searches for 10

1	4	9	15	10	8	2

  - becomes:

10	1	4	9	15	8	2

- Easy for lists, difficult for arrays: N-1 elements are moved 1 position to the left

II. Transpositions

- The element is shifted one position to the right
  - e.g., search(10)

1	4	9	15	10	8	2

  - becomes

1	4	9	10	15	8	2

- Easy for arrays and lists

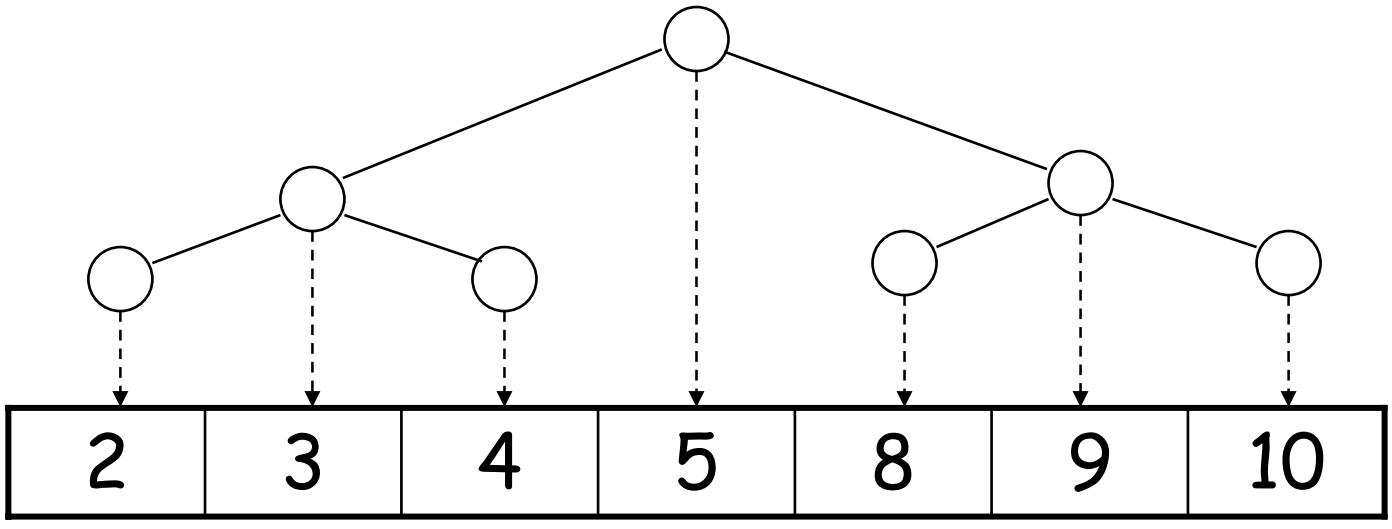
Critic

- Move to front adapts rapidly to the search conditions of the application
- Transposition adapts slowly but is more intuitively correct
- Combine the two techniques:
  - use initially move to front and
  - transposition later

Searching Ordered Sequences

- Sort the elements once
  - complexity: O(logN) instead of O(N)
- Search techniques:
  - binary search
  - interpolation search
  - indexed sequential search

I. Binary Search



$$N = 2^0 + 2^1 + 2^2 + \ldots \geq \sum_{i=0}^{d} 2^i = 2^{d+1} - 1 \Rightarrow \boxed{d \leq log(N+1)}$$

d: max number of comparisons

Complexity

- Maximum number or comparisons: a leaf is reached

$$\lceil log_2(N+1) \rceil$$

- Expected number of comparisons: tree searching stops before a leaf is reached

$$\frac{N+1}{N} log_2(N+1) - 1 \approx log_2 \frac{N}{2}$$

II. Interpolation

- Searching is guided by the values of the array
  - L: minimum value
  - U: maximum value
  - search position $h = \frac{x - L + 1}{U - L} N$
- Binary search always goes to the middle position

Example

- if x[h] = key element found; else search array on the left or on the right of h
  - e.g.

0							100

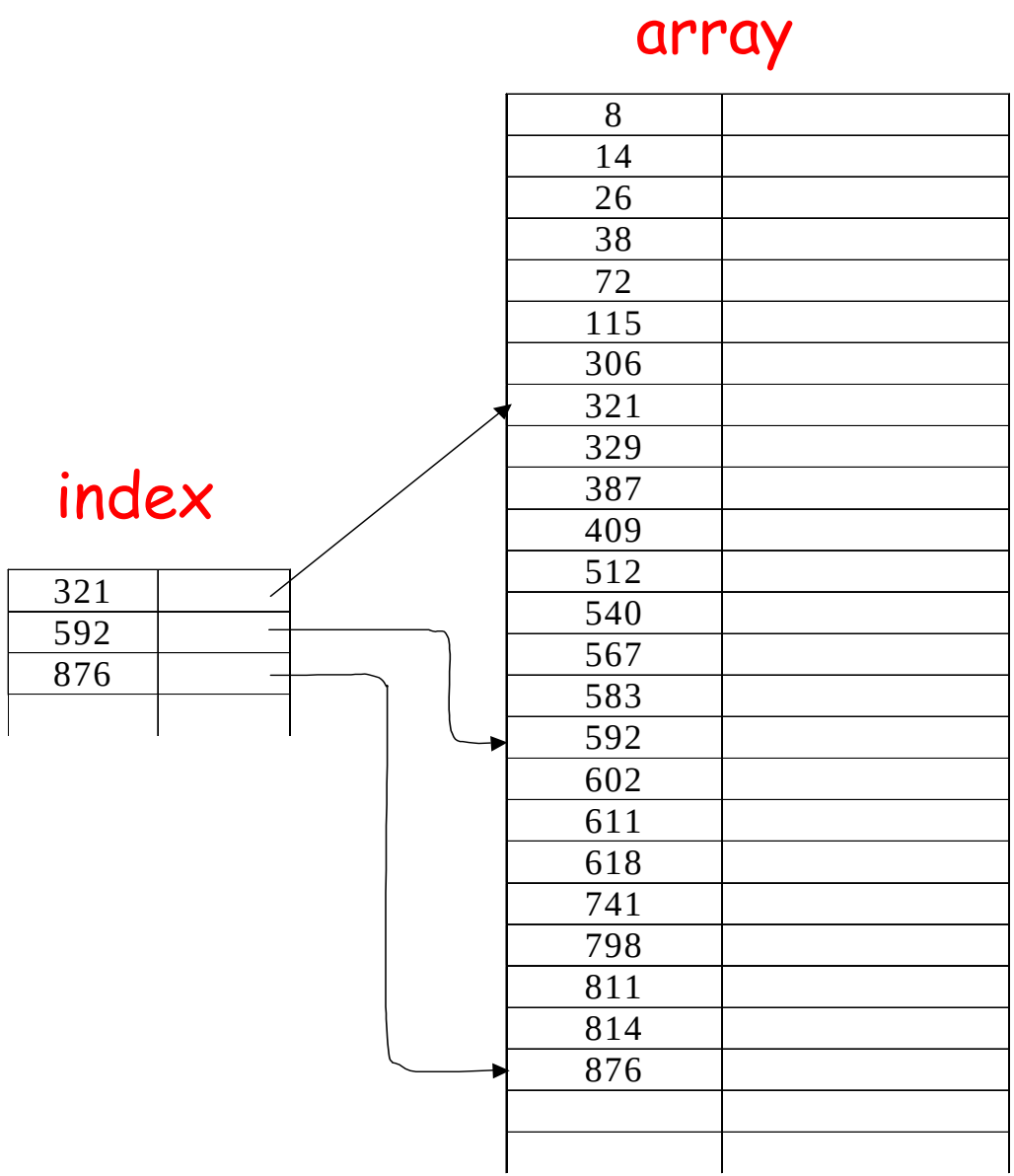
- search(80): focuses on the 20% rightmost part of the array

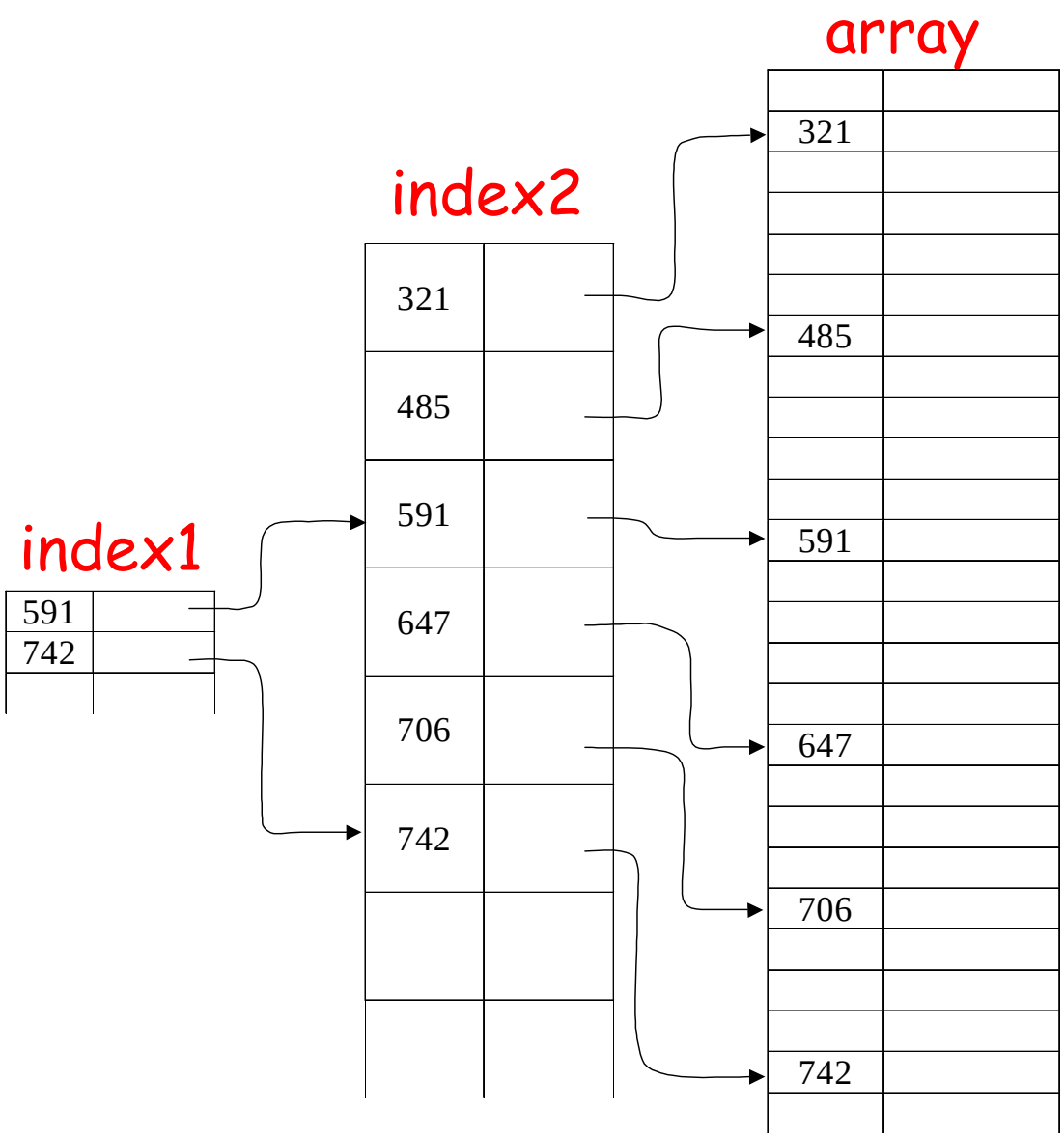
Complexity

- Average case: O(loglogN) uniform distribution of keys in the array
- Worst case: O(N) on non uniform distribution
- Binary search is O(logN) always!

III. Indexed Sequential Search

- A sorted index is set aside in addition to the array
- Each element in the index points to a block of elements in the array
  - e.g., block of 10 or 20 elements
- The index is searched before the array and guides the search in the array


array
index
8
14
26
38
72
115
306
321
329
387
409
512
540
567
583
592
602
611
618
741
798
811
814
876
321
592
876

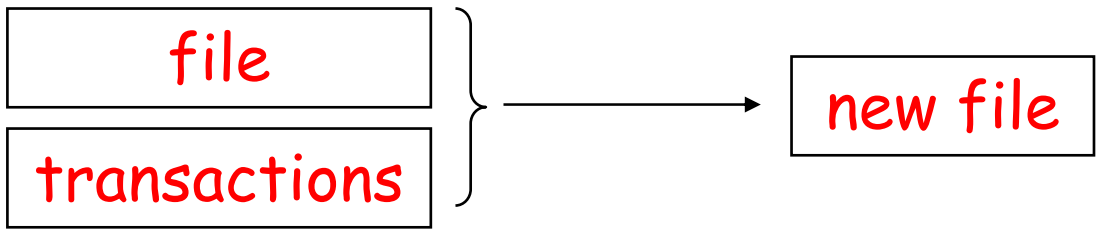

array
index2
index1
321
485
591
647
706
742
591
742
321
485
591
647
706
742

File Searching

- Access a data page, load it in the main memory and search for the key
  - unordered files: O(#blocks) disk accesses
  - ordered files: O(log#blocks) disk accesses
  - disk head moves back and forth
  - difficult to control the disk head moves especially in multi-user environments
  - leave 20% extra space for insertions

Ordered Files

- Optimize the performance using an auxiliary batch file
  - batch operations in ascending key order
  - process the operations one after the other



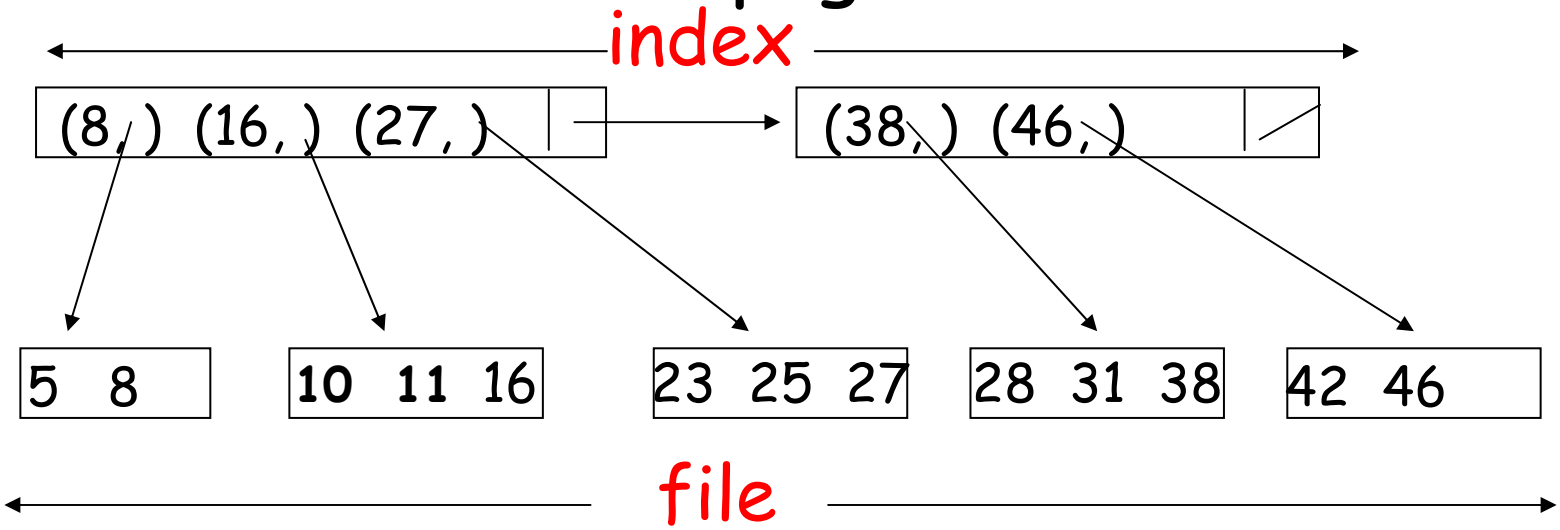
  - batch $a_1 <= a_2 <= \ldots <= a_N$

ISAM

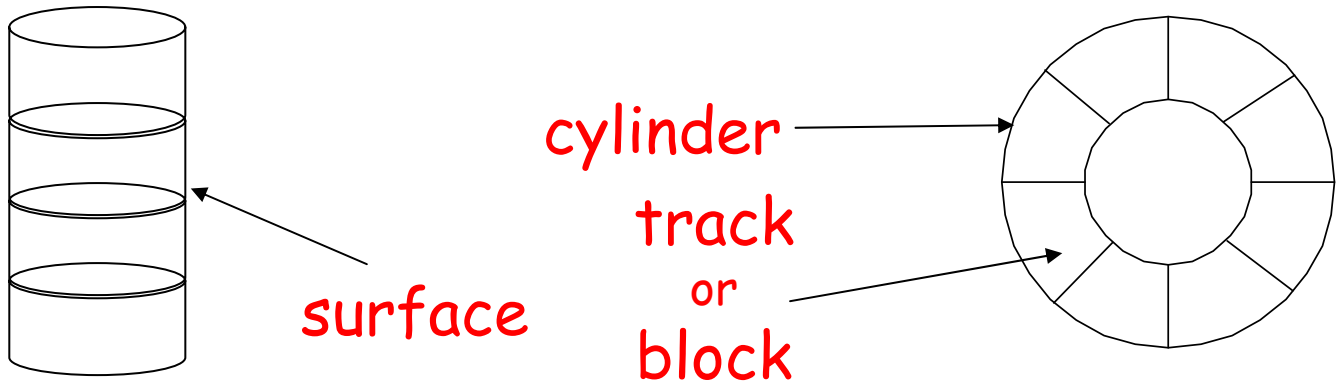
- Data pages on the disk
- Indices for faster retrievals
- Pseudo Dynamic Scheme
- Dynamic Schemes
  - B-trees
  - B+-trees, …

Index Sequential Files (ISAM)

- Random access based on primary key
- Fast disk access through an index
- Indices to data pages on the disk

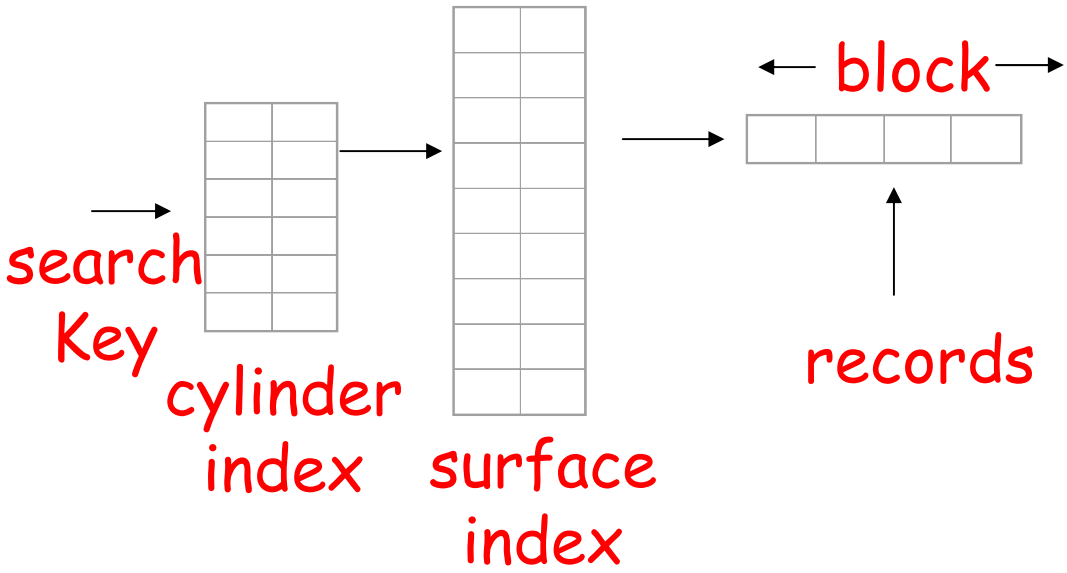


ISAM Index



- Master index: to disks - surfaces
- Cylinder index: one per disk unit
- Track index: one per cylinder

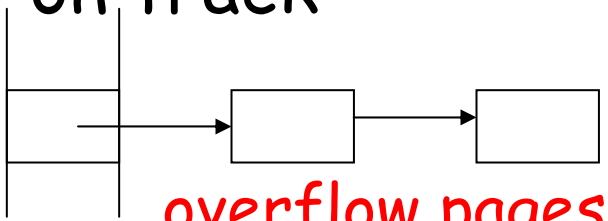
Retrieval



- Locate cylinder: 1st disk access
- Locate surface: 2nd disk access
- Locate track: 3rd disk access
- Overflows will cause more disk accesses!!

Overflows

- No space left on track
- Solutions
  1. chaining:

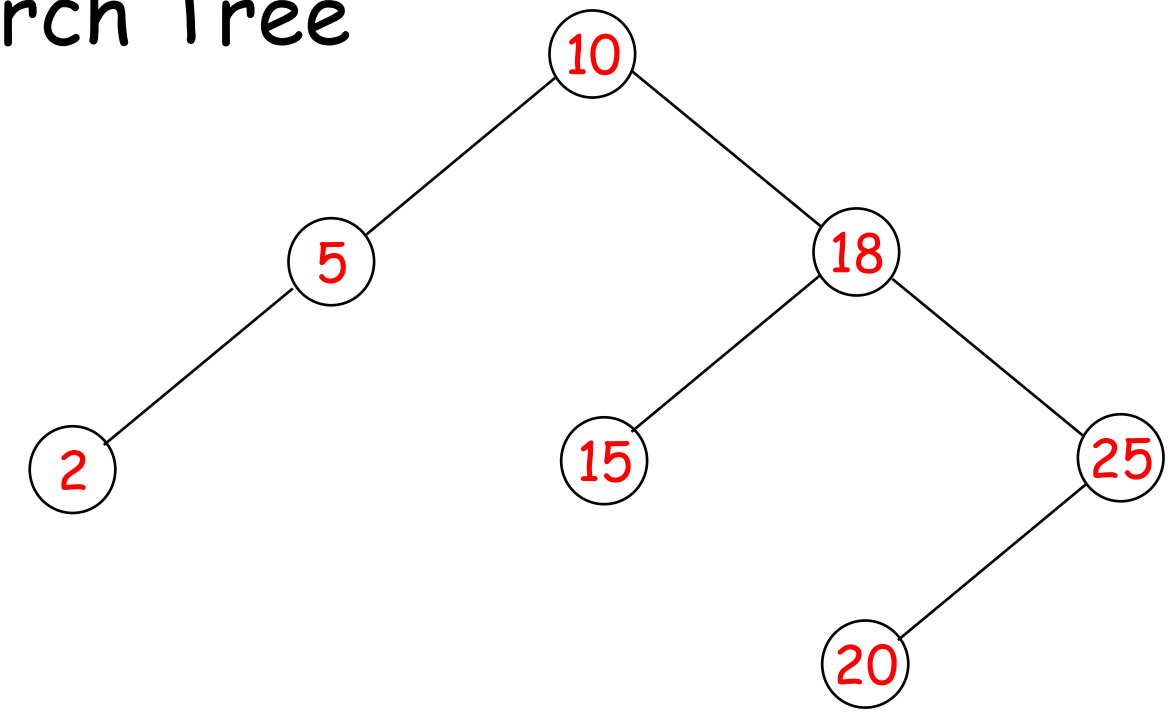


overflow pages

  2. distribution of overflow space between neighboring primary pages
     - file reorganization necessary soon or later!!
- Dependence on hardware!
- Pseudo dynamic behavior!

Tree Search

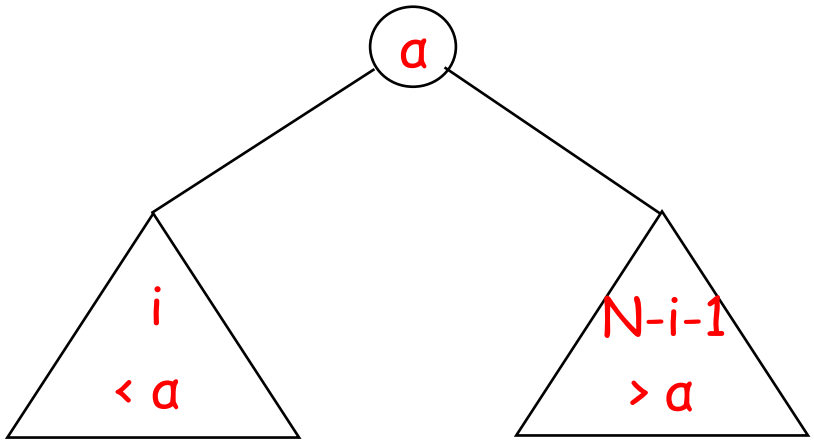
- The elements are stored in a Binary Search Tree



Complexity

- Average number of key comparisons or length of path traversed
  - average case: O(logN) comparisons
  - worst case: BST is reduced to list and search is O(N) !!
- The form of a BST depends on the insertion sequence
  - the keys are ordered: BST becomes list

Theorem



- Testing for membership in a random BST takes O(logN) time (expected cost)
- P(n): average number of nodes from root to a node
- P(0)=0, P(1)=1
- P(i): average height of left sub-tree
- P(n-i-1): average height of right sub-tree

Proof

- Average number of comparisons

$$p'(N) = \frac{i}{N}\underbrace{[p(i) + 1]}_{\text{left sub-tree}} + \underbrace{\frac{N-i-1}{N}[p(N-i-1)+1]}_{\text{right sub-tree}} + \frac{1}{N}_{\text{root}}$$

- Average over all insertion sequences

$$p(N) = \frac{1}{N}\sum_{i=0}^{N-1} P'(i) = \frac{1}{N}\sum_{i=0}^{N-1}\left[\frac{1}{N} + \frac{P(i)+1}{N} i + \frac{P(N-i-1)+1}{N}(N-i-1)\right] =$$

$$= 1 + \frac{1}{N^2}\sum_{i=0}^{N-1}[iP(i) + (N-i-1)P(N-i-1)]$$

Proof (cont.)

- … because a can be inserted first, second, n-th element => n cases
- N – i - 1 → i => $$P(N) = 1 + \frac{2}{N^2} \sum_{i=1}^{N-1} iP(i)$$
- Prove by induction: P(N) <= 1 + 4logN
  - a more careful analysis shows that the constant is about 1.4 => P(N) <= 1.4logN

	Trees	Arrays/Lists	Hashing
Main memory (Static)	▪Optimal Trees	▪Unsorted (move-to-front, transposition) ▪Sorted (binary search)	▪Rehashing ▪Coalesced ▪chaining
Main memory (dynamic mem. allocation)	▪BST ▪AVL ▪SPLAY	▪Unsorted (move-to-front, transposition)	▪Separate chaining
Disk (static)		▪Files with overflows ▪Indexed sequential Files (ISAM)	▪Table ▪Separate chaining
Disk (dynamic mem. allocation)	▪M-trees ▪B-trees, B+-trees (VSAM)		▪Dynamic ▪Extendible ▪Linear