Introduction to Databases
CS 451 / 551

Lecture 4:

Searching and Indexing: Part 1

Suyash Gupta

Assistant Professor

Distopia Labs and ORNG

Dept. of Computer Science

(E) suyash@uoregon.edu

(W) gupta-suyash.github.io

UNIVERSITY OF OREGON

Assignment 1 is Out!
Deadline: Oct 28, 2025 at 11:59pm

Start collaborating with your groups!

Quiz 1: Oct 16, 2025 (in class)

Given a pile of numbers, how can we search or access any number or a range of numbers.

Say following is a set of numbers

23 12 76 45

98 34 8 19

67 78

83 6

How will you find 76?

23 12 76 45

34 8 98 19

67 78 83

6

How will you check that 44 does not exist?

23 12 76 45

34 8 98 19

67 78

6 83

Lets assume these numbers are in a list

23 34 12 67 8 6 76 98 78 45 83 19

Can we impose an order?

How about we sort all the numbers?

6 8 12 19 23 34 45 67 76 78 83 98

Sorting allow easy linear scan

6 8 12 19 23 34 45 67 76 78 83 98

We have a very simple algorithm, **linear scan**:

- Start from the first number and check each number.
- Once you reach a number greater than the number you are searching, stop the search.

Sorted Linear Scan

6 8 12 19 23 34 45 67 76 78 83 98

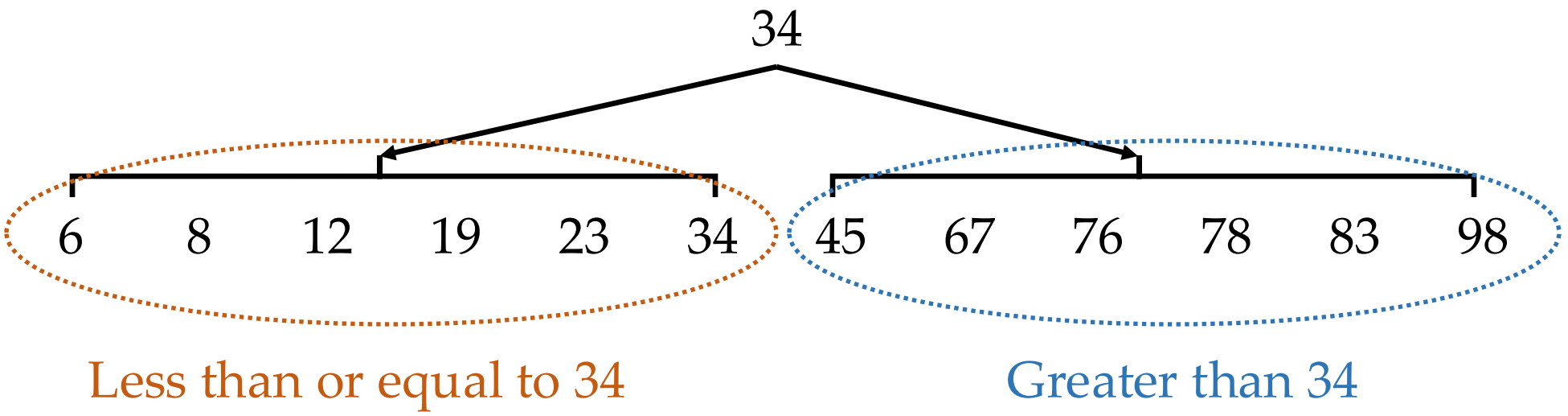
What is the average time complexity?

O(n)

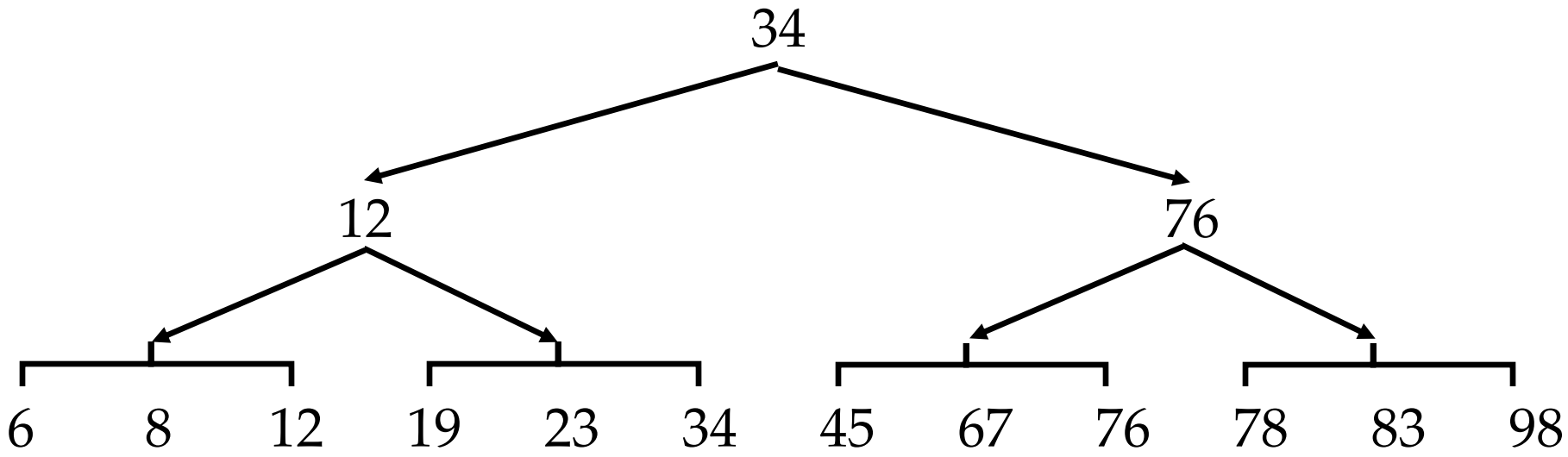
Can we do better than Linear Scan?

Binary Search

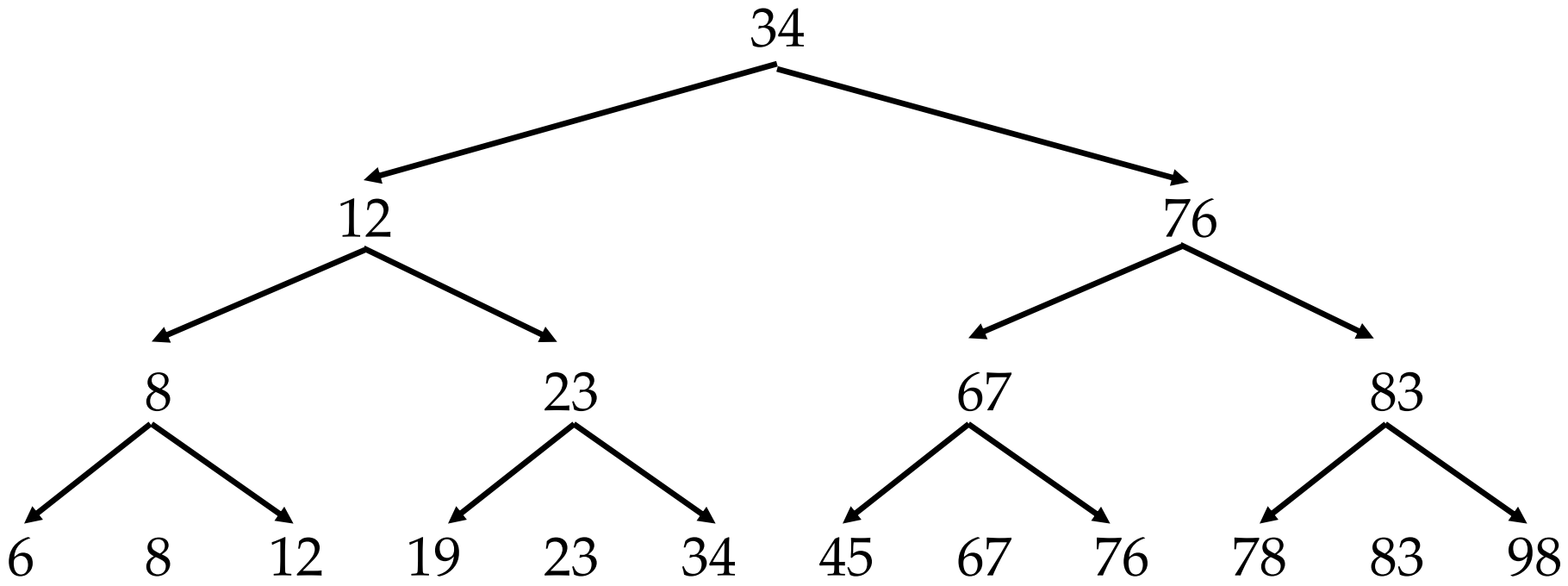
- Lets split the numbers in a manner that eases searching.
- Simply said, we are going to group numbers.



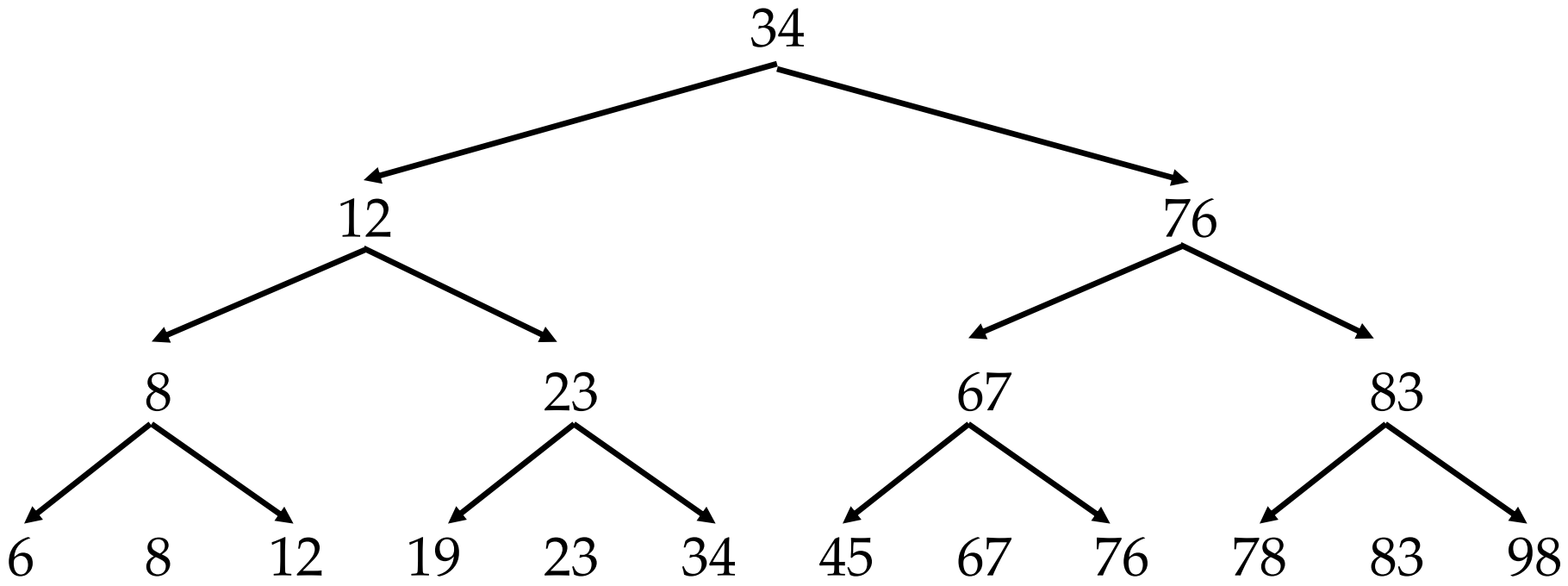
Binary Search Split Further



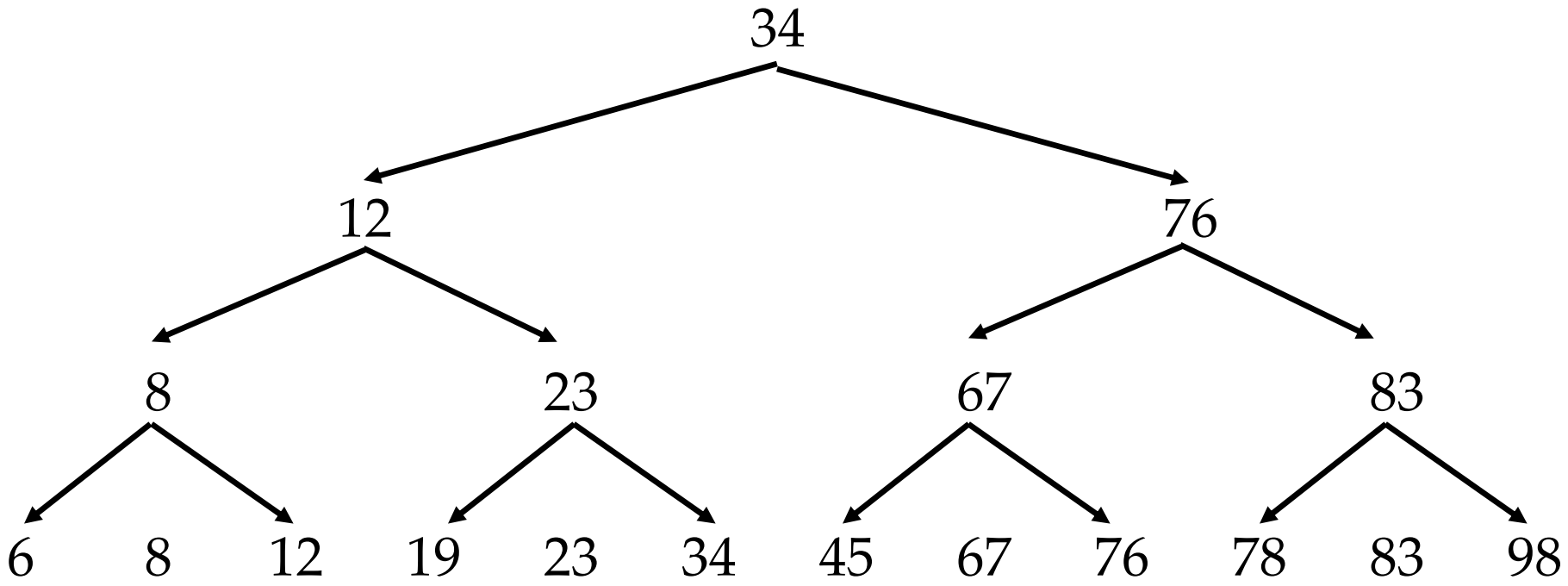
Binary Search Split Further



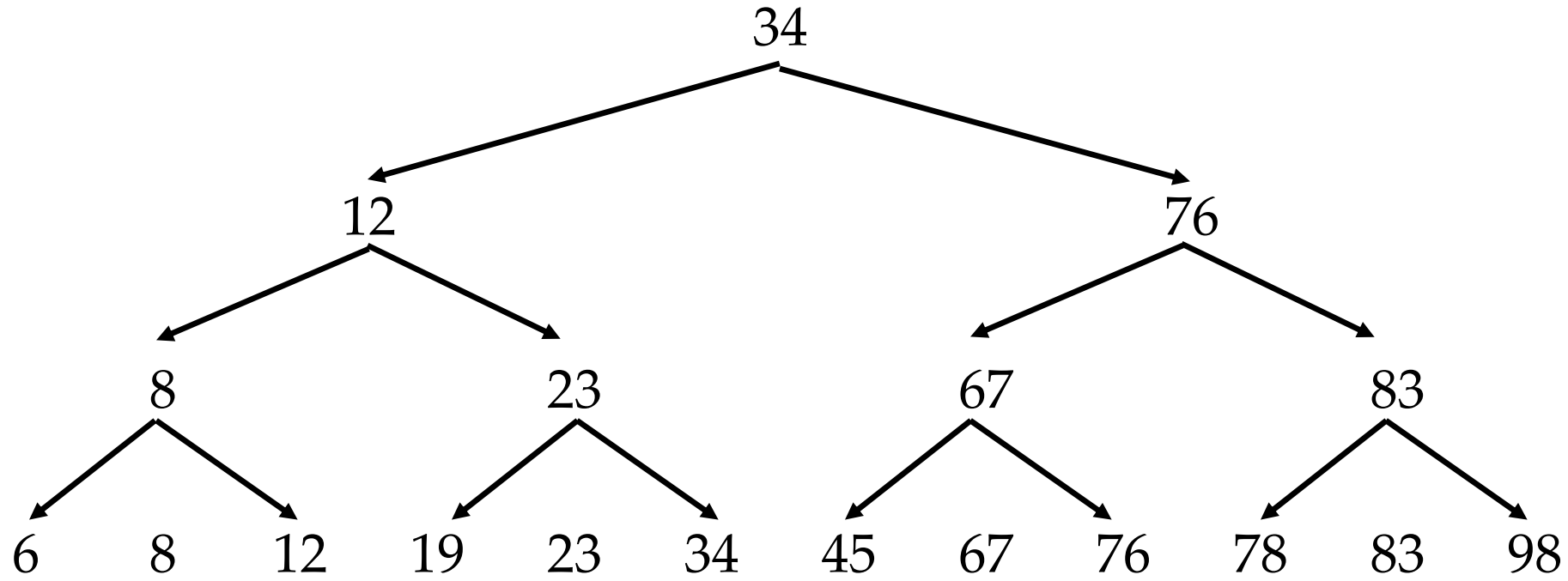
This structure is similar to Binary Tree



Challenges for Binary Tree?

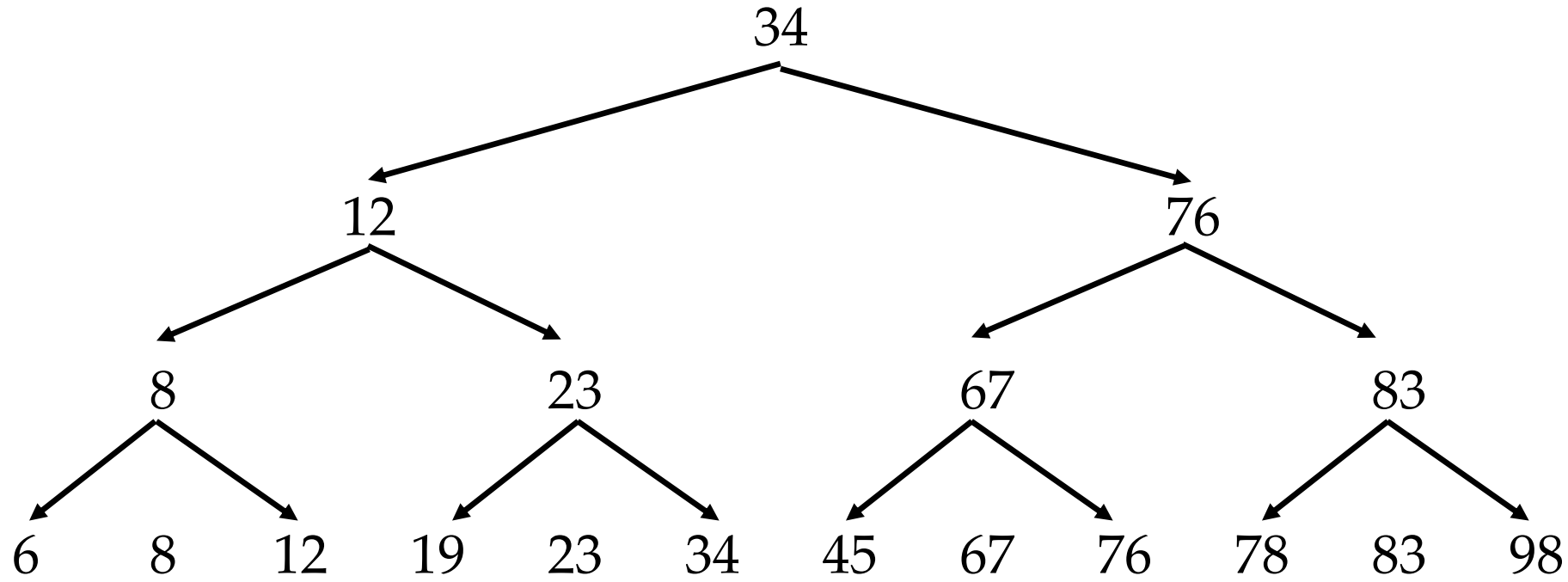


Binary Tree Challenges: Insertion and Deletion



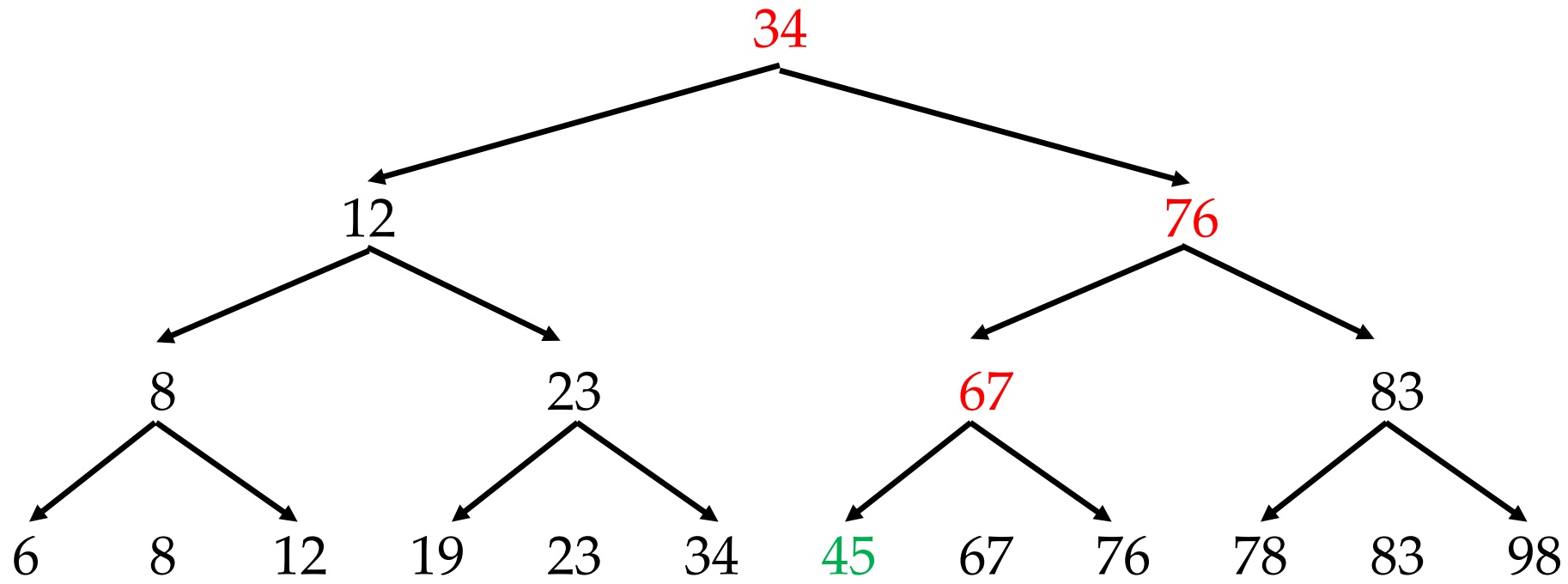
- So, we have a binary tree.
- Inserting a new key or deleting an existing key is going to require updating the tree structure.

Binary Tree Challenges: Concurrency



- Something that we will talk later is *concurrency*.
- Searching/Updating multiple numbers at the same time would be hard as we need to lock a large part of the tree.

Binary Tree Challenges: Point or Range Query



- To read 45, we need 4 Disk read operations. Remember, all of these are stored on disk!
- But, say we want to read all numbers greater than 4?
- Is sorted list better now? Supports sequential access.

Indexes

Indexes

- Databases can be large → Extremely large with thousands of records.

- How can you search a record quickly in such large databases?
  - Remember, we saw that in a file, each record is identified through a search key.
  - To search a record, File system manager needs to bring a record from the disk, and then check its search key.
  - Imagine a query that wants to access multiple such records!

- Can we do better? → Yes, with the help of Indexes!
  - Like books have *table of contents* that tell about the chapters in the book.
  - Indexes inform about records.

How to determine a Good Index?

- A good index should help to search a record fast!
- Characteristics of a good index:
  - **Access Types:** Supports accessing a particular record (point query) and/or records within a specified range (range query).
  - **Access Time:** Time to find a particular record.
  - **Insertion Time:** Time to insert a new record in the index (includes time to find the right place to insert).
  - **Deletion Time:** Time to delete a new record in the index (includes time to find the item to be deleted).
  - **Space Overhead:** The space consumed by the index.

Types of Indexes

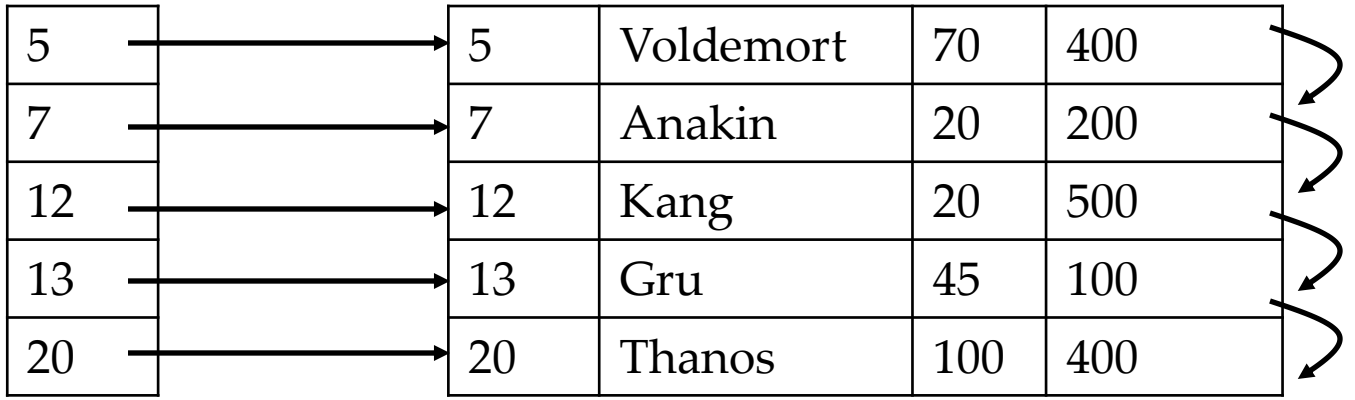
- Broadly, we can divide indexes into two groups:
  - **Ordered Indexes**
  - Hashed Indexes

Clustering Indexes

- Index built on some search key.
- Search key could be the primary key or any other field of the table.
- Index entries are stored in a sorted manner.
  - Hence, also called as sequential indexes.
- Types: Dense and Spare Indexes.

Dense Indexes

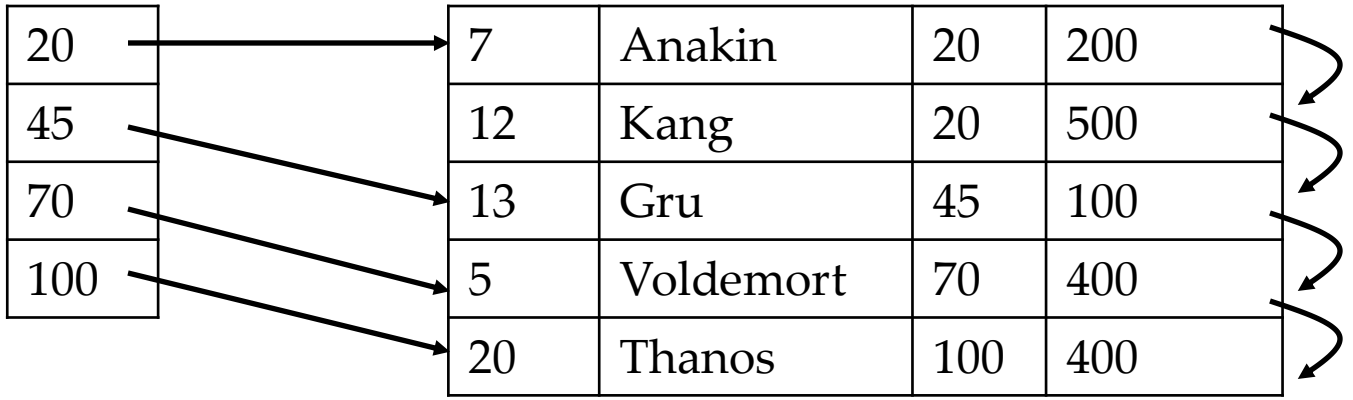
- Dense index includes an entry for every search-key.



Dense index with search-key “ID”

Dense Indexes

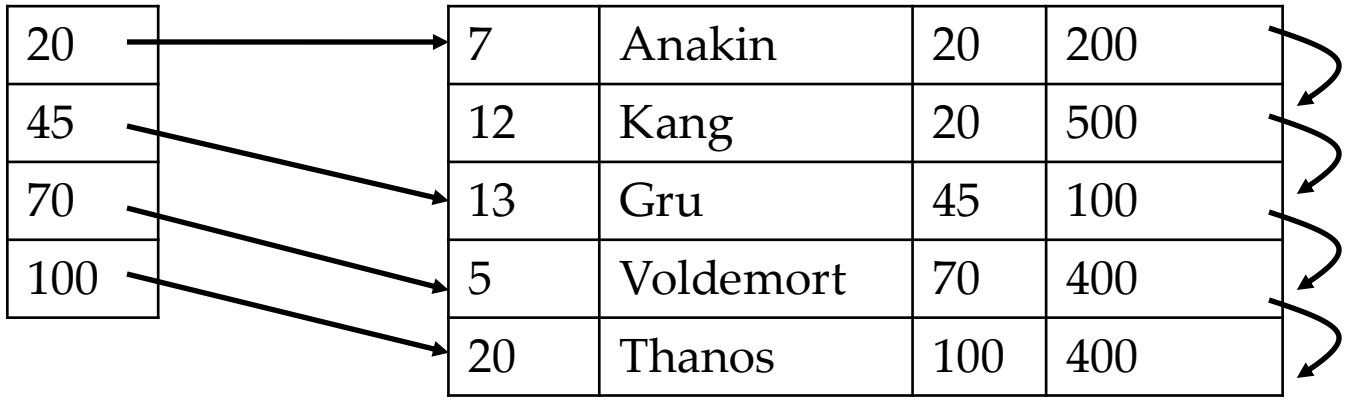
- Any attribute can act as the Search-key.



Dense index with search-key “Age”

Inserting a new key/record in Dense Indexes

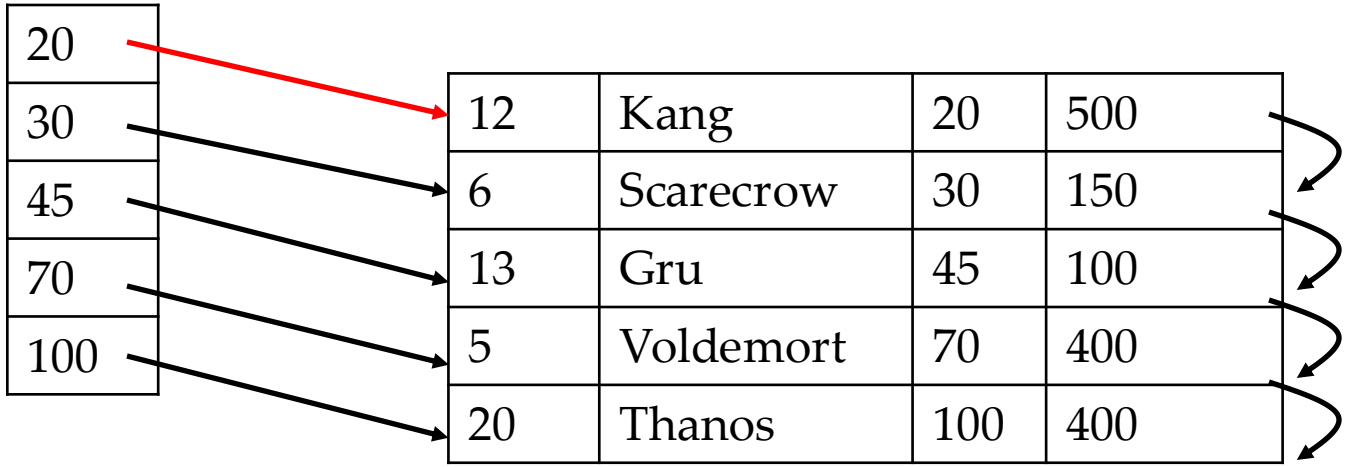
- Say, I want to insert a record with **age 30** (search-key is Age).



6	Scarecrow	30	150

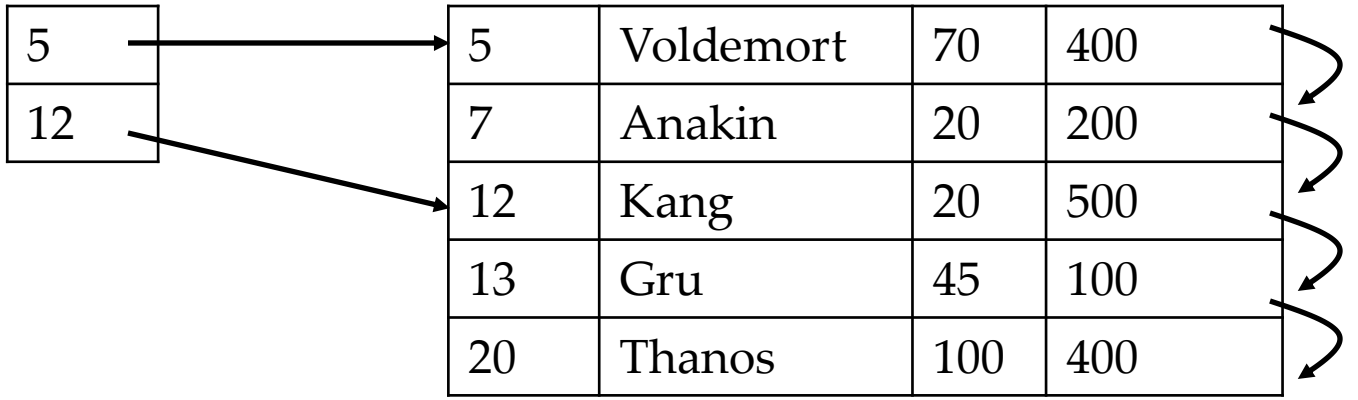
Deleting a key/record in Dense Indexes

- Say, I want to delete a record with **name Anakin and age 20** (search-key is Age).



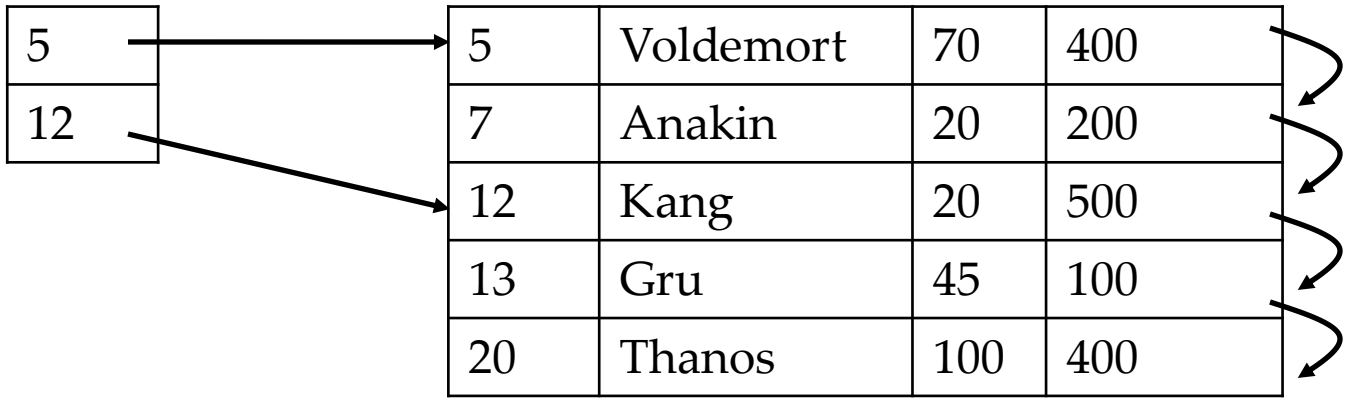
Sparse Indexes

- Sparse index includes an entry for only some search-keys.
  - Records are divided in groups and only one representative key per group.



Sparse Indexes

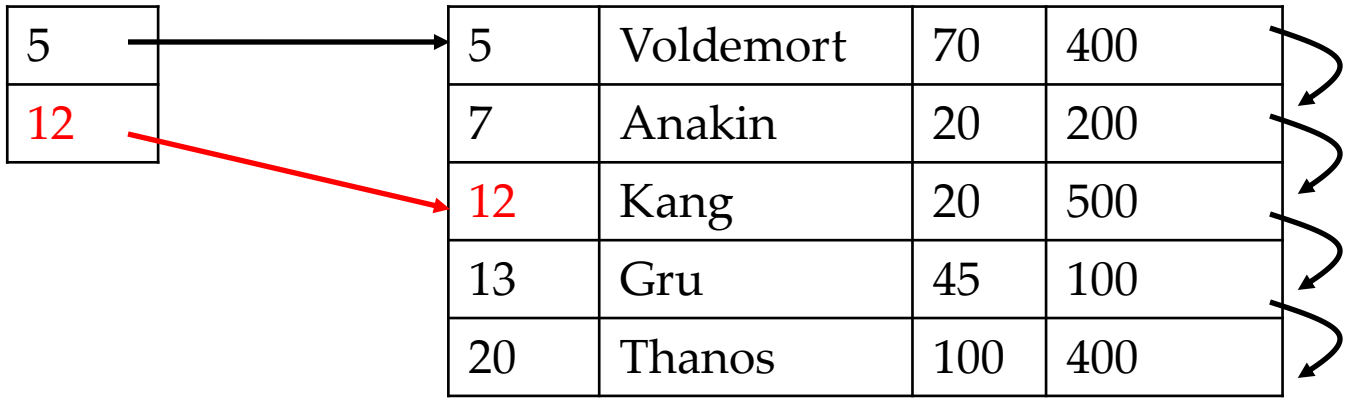
- To find an entry in a sparse index.
  - Start at the largest value smaller than the required entry.
  - Then follow the pointers.



Process to finding 20.

Sparse Indexes

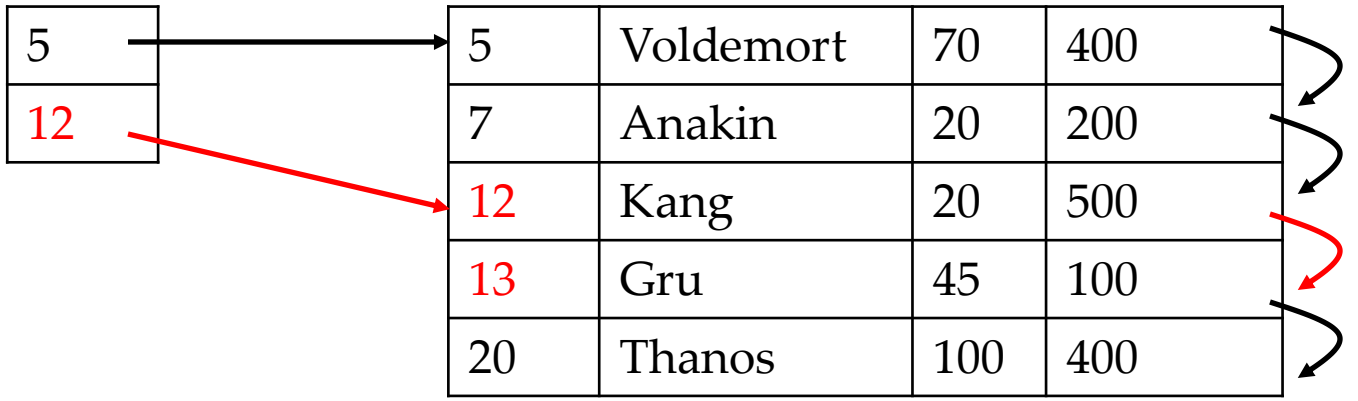
- To find an entry in an sparse index.
  - Start at the largest value smaller than the required entry.
  - Then follow the pointers.



Process to finding 20.

Sparse Indexes

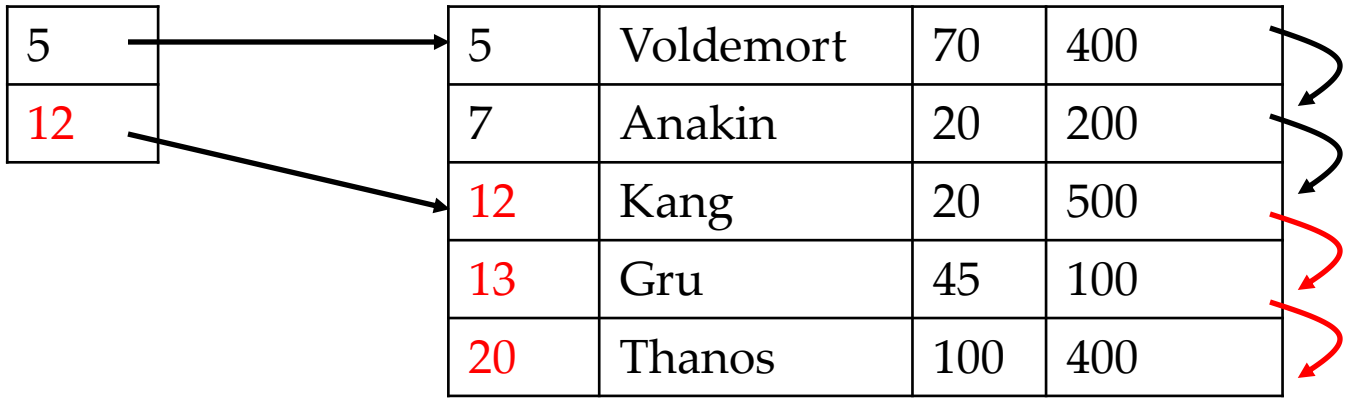
- To find an entry in an sparse index.
  - Start at the largest value smaller than the required entry.
  - Then follow the pointers.



Process to finding 20.

Sparse Indexes

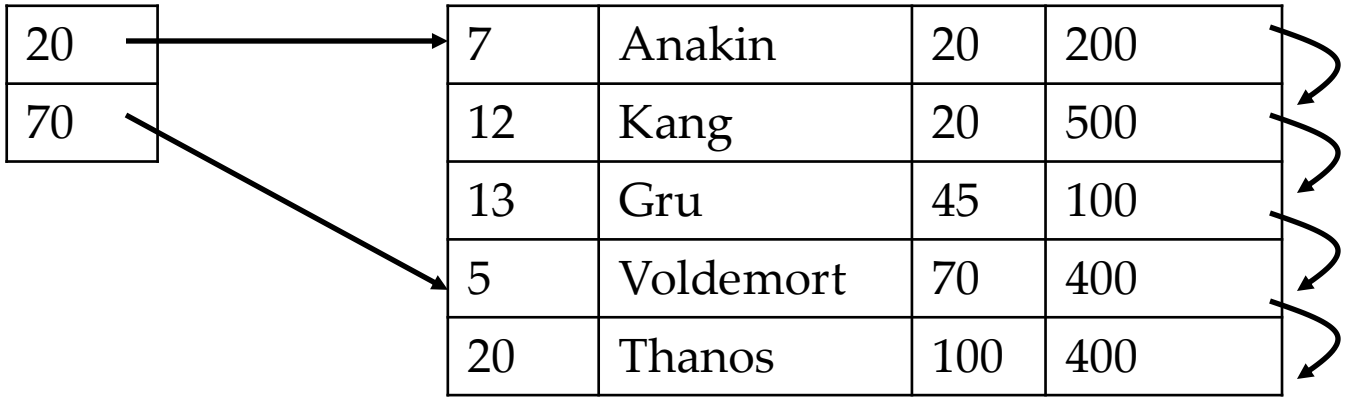
- To find an entry in an sparse index.
  - Start at the largest value smaller than the required entry.
  - Then follow the pointers.



Process to finding 20.

Inserting a new key/record in Sparse Indexes

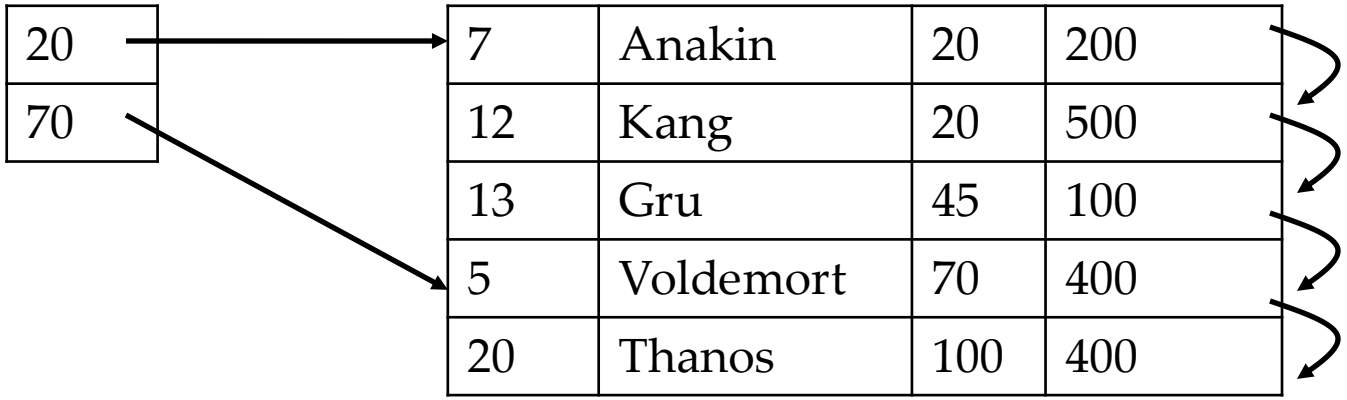
- Say, I want to insert a record with **age 30** (search-key is Age).



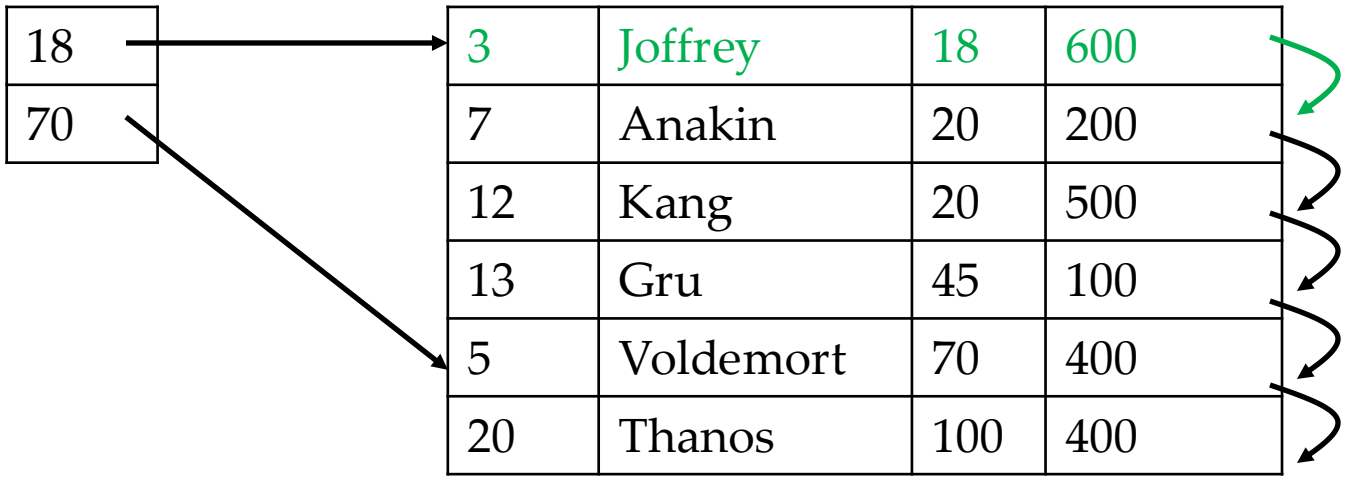
6	Scarecrow	30	150

Inserting a new key/record in Sparse Indexes

- Say, I want to insert a record with **age 18** (search-key is Age).



3	Joffrey	18	600

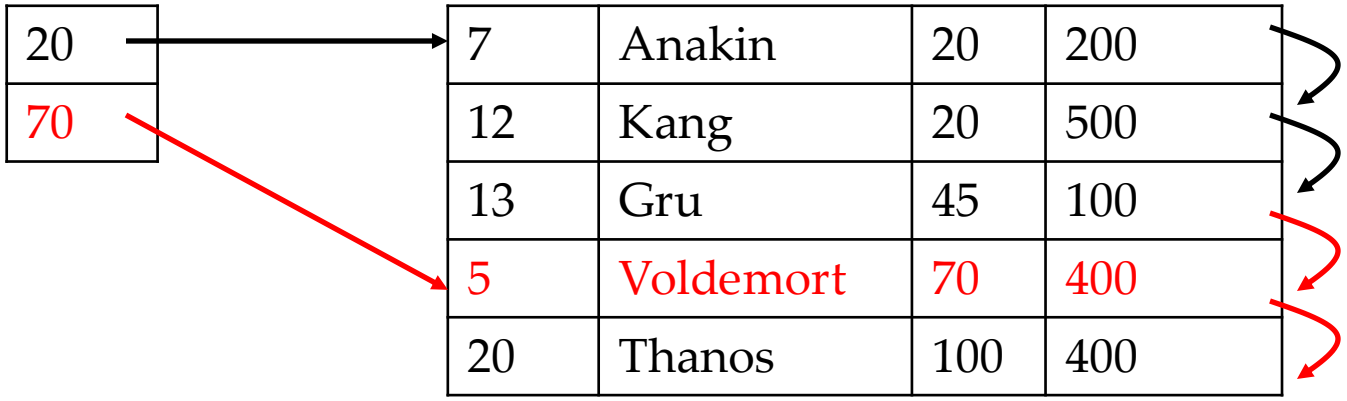


Deleting a key/record in Sparse Indexes

- Say, I want to delete the record with **name Kang and age 20** (search-key is Age).

Deleting a key/record in Sparse Indexes

- Say, I want to delete the record **Voldemort with age 70** (search-key is Age).



Multi-Level Indexes

- Sparse index can still be large.
- In a database with 100 million entries a spare index of 100k entries will still span multiple pages of the disk.
- How about we design multiple levels of indices? → an index for an index

Secondary Indexes

- Databases can have more than one index.
- Say in your database, you have an index on age of each employee, but soon you observe that another set of frequent queries that you get is for employee salaries.
- Your current index is not effective in such a situation.
- You can design a “secondary index” on salary.
- However, secondary index must be dense! Should have an entry for each record.
  - Why? Because your records are stored in the file according to the primary index.

Automatic Index Creation

- Modern databases automatically create an index on the primary key.

- Whenever a new tuple is inserted, they verify if the primary key property (unique and not null) are not violated, and if not, the tuple is added to the database and an entry is set in the index.

Automatic Index Creation

- Modern databases automatically create an index on the primary key.

- Whenever a new tuple is inserted, they verify if the primary key property (unique and not null) are not violated, and if not, the tuple is added to the database and an entry is set in the index.