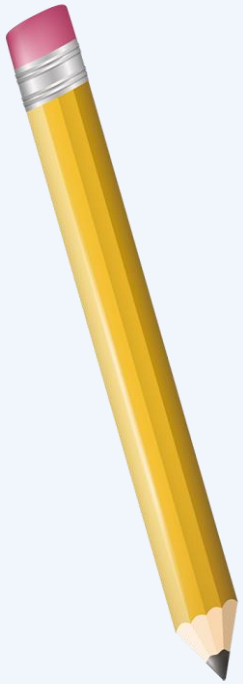
اللهم صل وسلم وبارك على سيدنا محمد وعلى
آله وصحبة وسلم تسليماً كثيراًطيباً مباركاً فيه

File Organization



Dr \ Mohammed Ahmed Mahfouz

Doctor of Information Systems
Thebes Higher Institute for Management and Information Technology

Indexing

Lecture No. 10

Contents

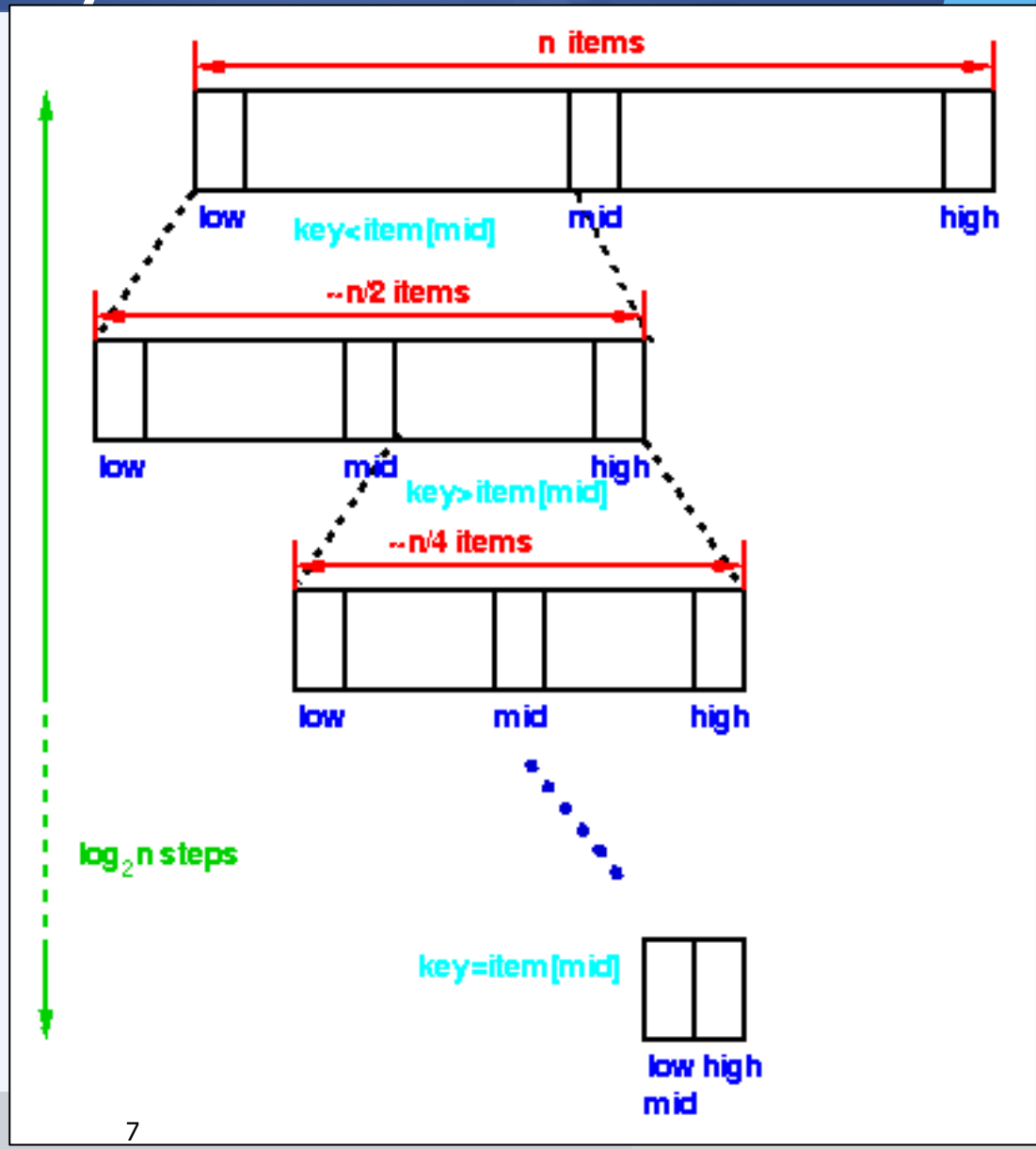
Binary Searching

Binary Search

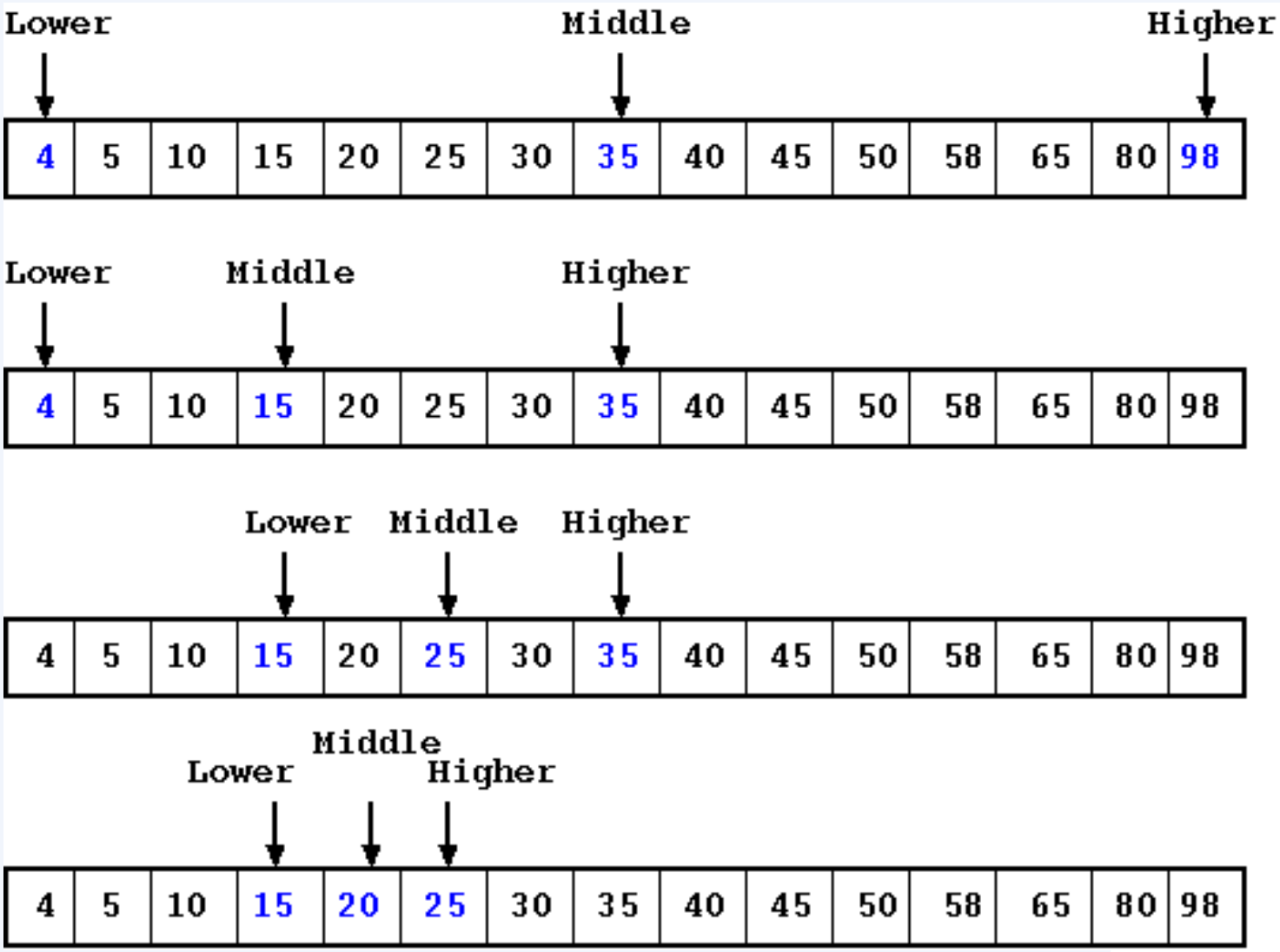
- Let us consider fixed-length records that must be searched by a key value
- If we knew the Relative Record Number RRN of the record identified by this key value, we could jump directly to the record (by using seekg function)
- In practice, we do not have this information and we must search for the record containing this key value
- If the file is not sorted by the key value we may have to look at every possible record before we find the desired record
- An alternative to this is to maintain the file sorted by key value and use binary searching

Binary Search

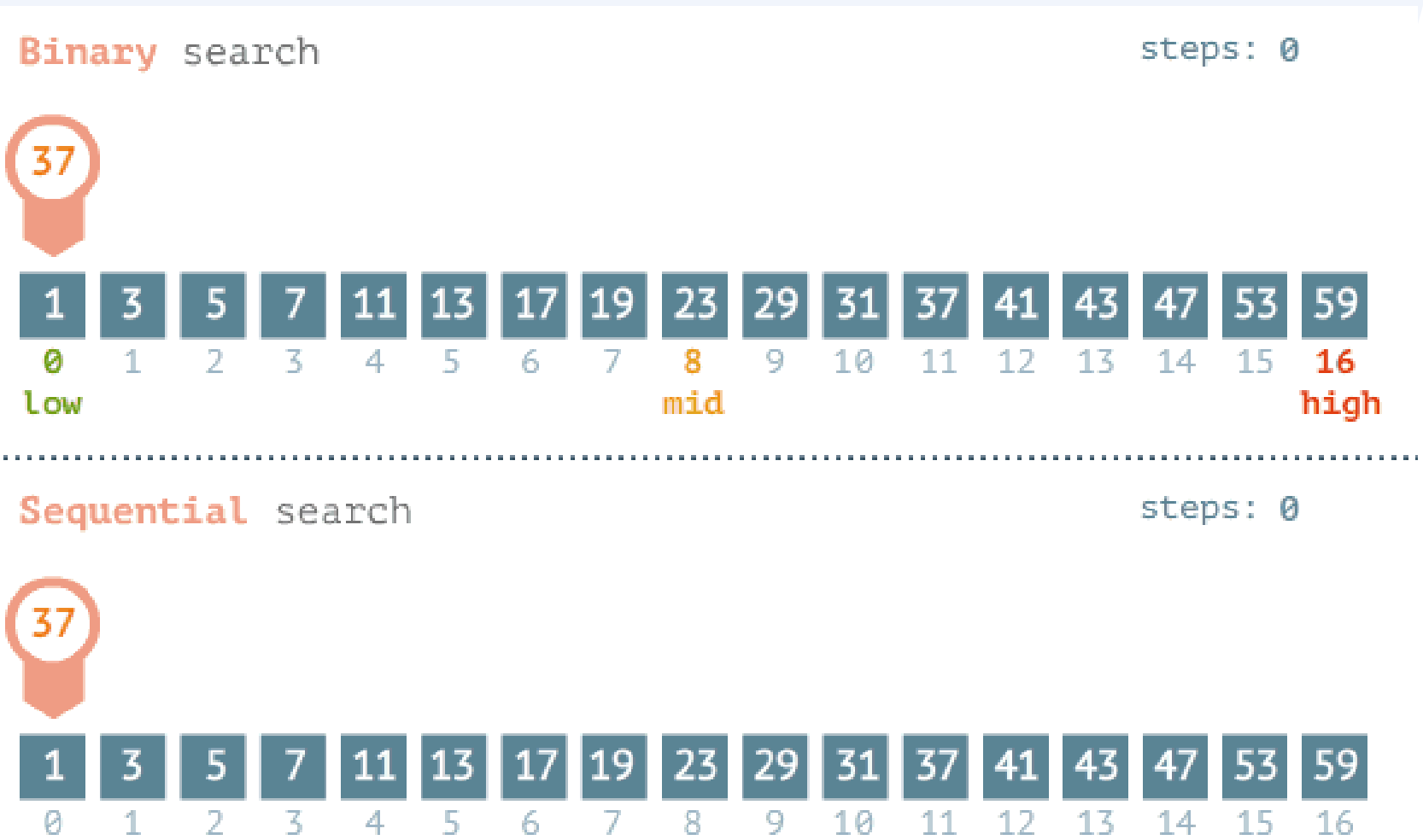
- A technique for searching an ordered list in which we first check the middle item and - based on that comparison - "discard" half the data. The same procedure is then applied to the remaining half until a match is found or there are no more items left.



Binary Search

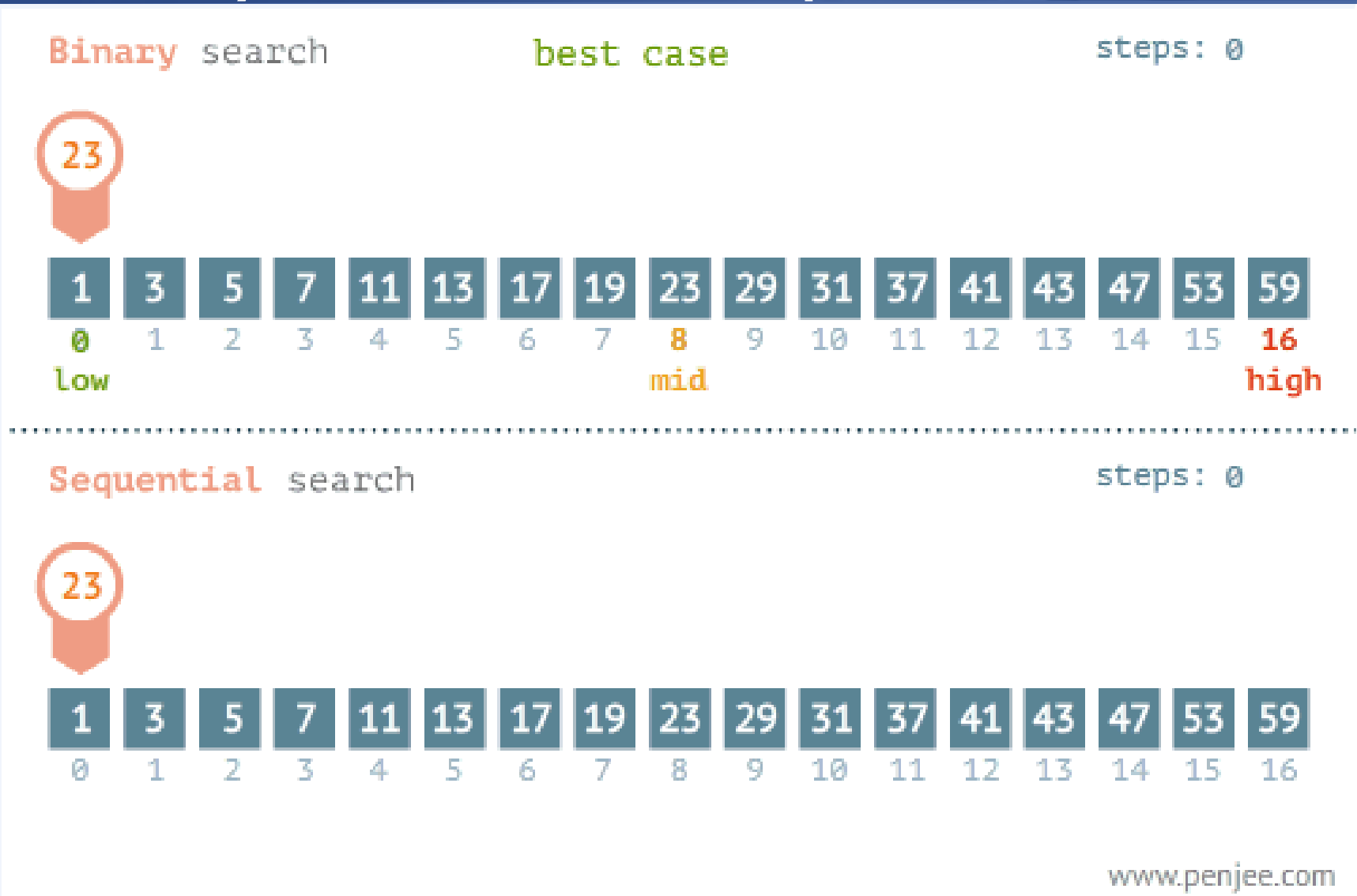


Binary Search vs Sequential Search



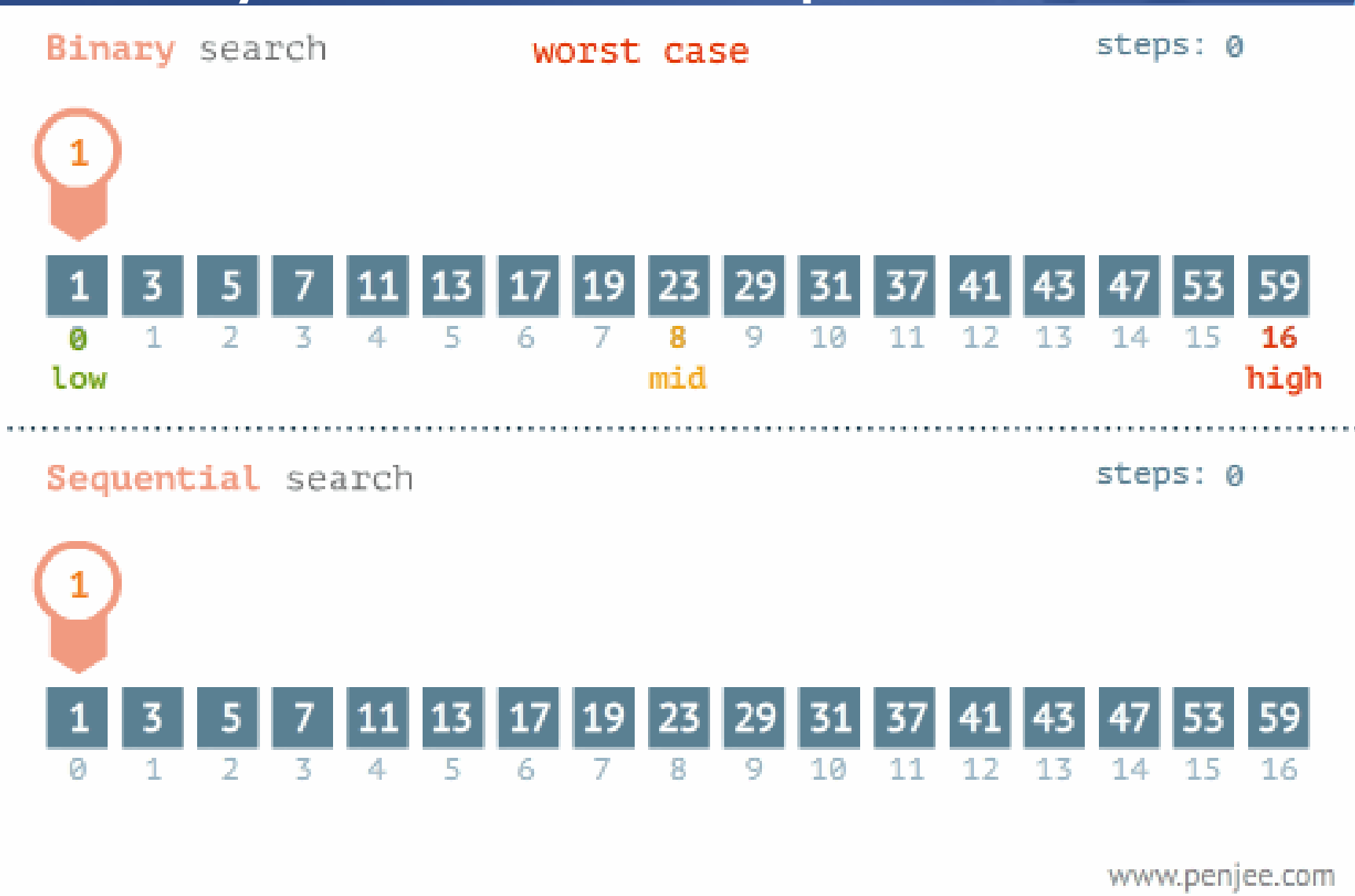
www.penjee.com

Binary Search vs Sequential Search



www.penjee.com

Binary Search vs Sequential Search



Binary Search vs Sequential Search

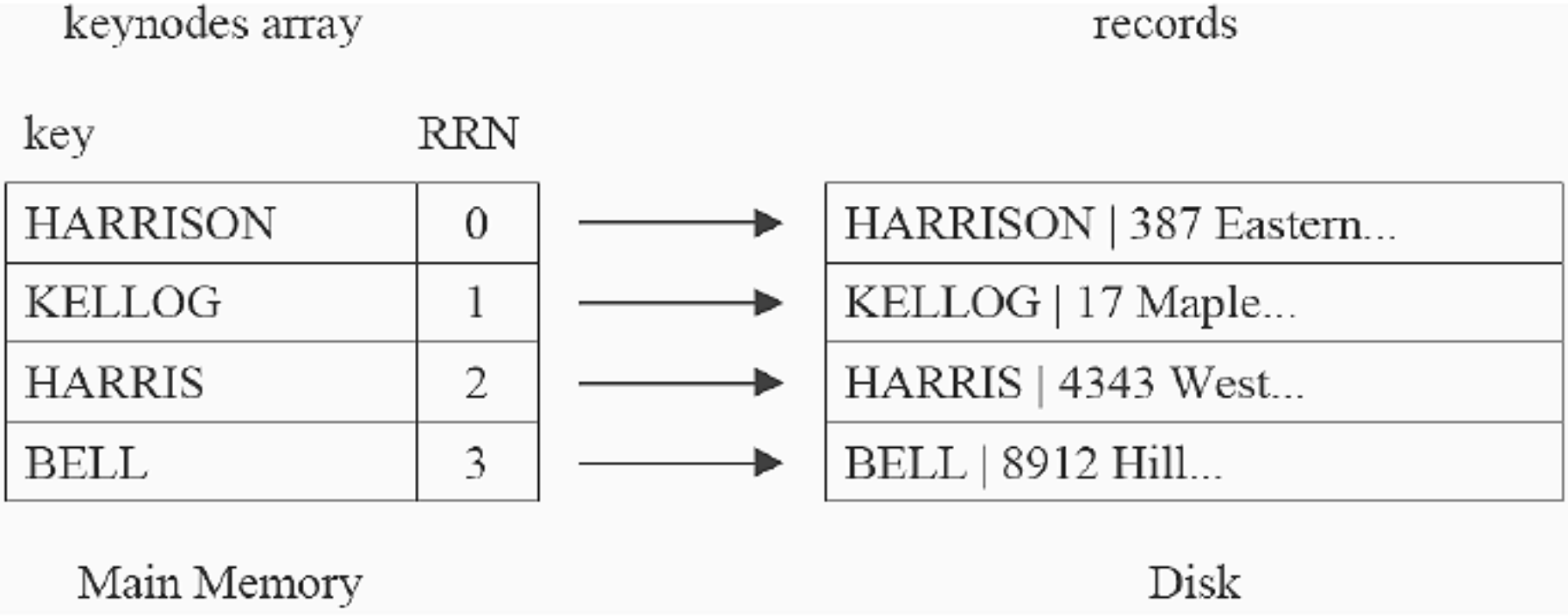
- **Sequential Search:** O(n)
- **Binary Search:** $O(\log_2 n)$
- If file size is doubled, sequential search time is doubled, while binary search time increases by 1

Key Sorting

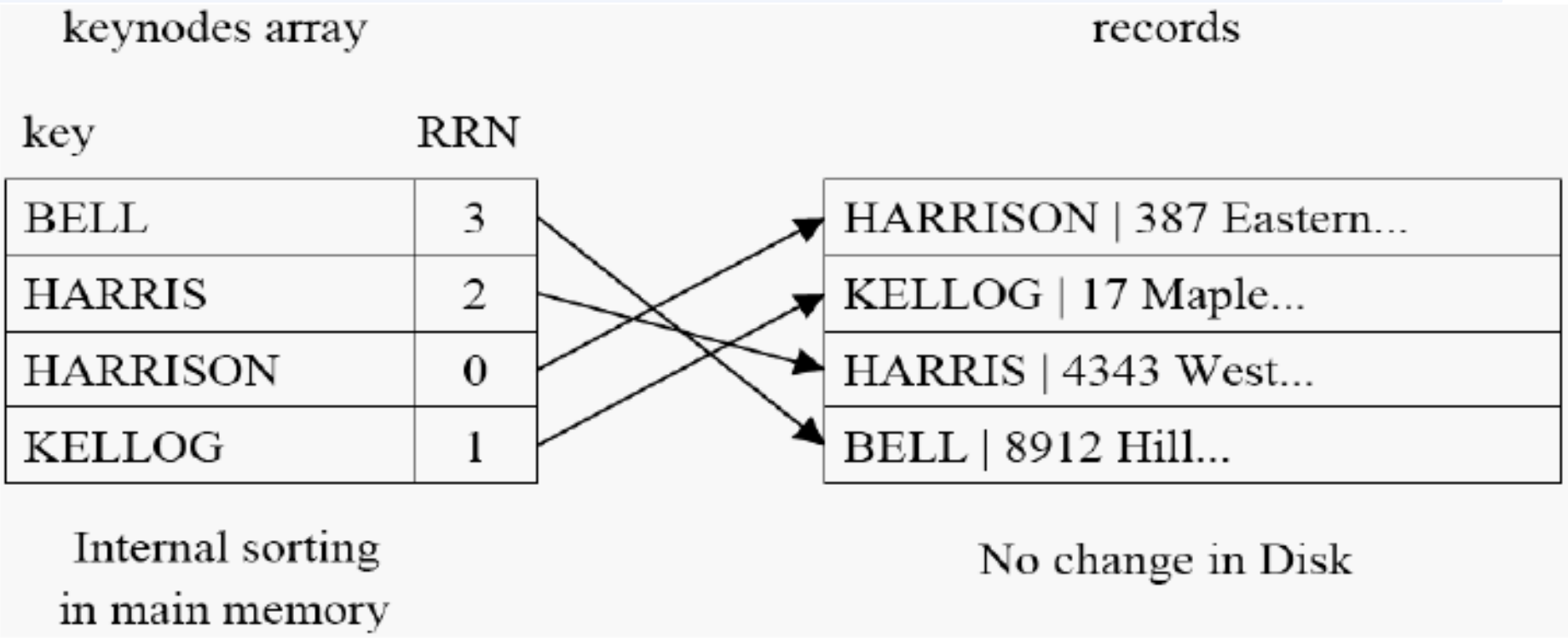
Key Sorting

- Suppose a file needs to be sorted, but it is too big to fit into main memory.
- To sort the file, we only need the keys.
- Suppose that all the keys fit into main memory
- **Idea:**
  - Bring the keys to main memory plus corresponding RRN
  - Do internal sorting of keys
  - Rewrite the file in sorted order

Key Sorting


keynodes array
records
key
RRN
HARRISON
0
KELLOG
1
HARRIS
2
BELL
3
HARRISON | 387 Eastern...
KELLOG | 17 Maple...
HARRIS | 4343 West...
BELL | 8912 Hill...
Main Memory
Disk

Key Sorting


keynodes array
records
key
RRN
BELL
3
HARRIS
2
HARRISON
0
KELLOG
1
HARRISON | 387 Eastern...
KELLOG | 17 Maple...
HARRIS | 4343 West...
BELL | 8912 Hill...
Internal sorting
in main memory
No change in Disk

Key Sorting

keynodes array

BELL	3
HARRIS	2
HARRISON	0
KELLOG	1

records

BELL | 8912 Hill...
HARRIS | 4343 West...
HARRISON | 387 Eastern...
KELLOG | 17 Maple...

create new sorted file to replace previous

Key Sorting

- **Why bother to write the file back?**

index file

BELL	3
HARRIS	2
HARRISON	0
KELLOG	1

records

HARRISON | 387 Eastern...
KELLOG | 17 Maple...
HARRIS | 4343 West...
BELL | 8912 Hill...

leave file unchanged

This is called indexing!

Indexing

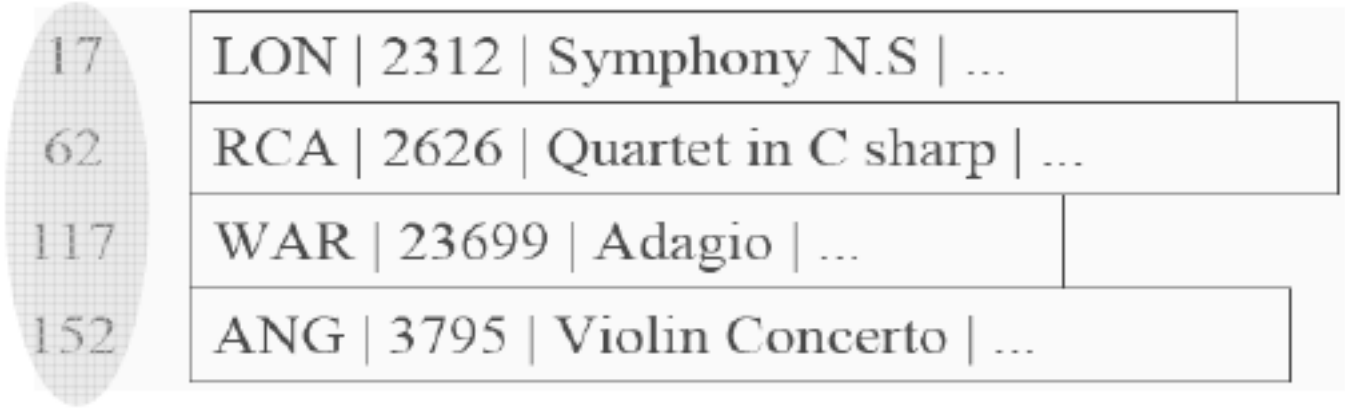
Introduction to Indexing

- Simple indexes use simple arrays.
- An **index** lets us impose order on a file without rearranging the file.
- Indexes provide multiple access paths to a file – multiple indexes (like library catalog providing search for author, book and title)
- An index can provide keyed access to variable-length record files

A Simple Index for Entry-Sequenced File

- Records (Variable-length)

address of record



- Primary key = company label + record ID

index:

key	reference field
ANG3795	152
LON2312	17
RCA2626	62
WAR23699	117

Index

- Index is sorted (main memory).
- Records appear in file in the order they entered.
- How to search for a recording with given LABEL ID?
  - Binary search (in main memory) in the index: find LABEL ID, which leads us to the referenced field.
  - Seek for record in position given by the reference field.

Some Issues

- How to make a persistent index
- i.e. how to store the index into a file when it is not in main memory
- How to guarantee that the index is an accurate reflection of the contents of the file
- This is tricky when there are lots of additions, deletions and updates

Indexing

- Operations in order to maintain an Indexed File

1. Create the original empty and data files.
2. Load the index file into memory before using it.
3. Rewrite the index file from memory after using it.
4. Add data records to the data file.
5. Delete records from the data file.
6. Update the index to reflect changes in the data file

Rewrite the index file from memory

- When the data file is closed, the index in memory needs to be written to the index file.

- An important issue to consider is what happens if the rewriting does not take place (power failures, turning the machine off, etc.)

Two Important Safeguards

1. Keep an status flag stored in the header of the index file.

   - The status flag is “on” whenever the index file is not up-to-date.
   - When changes are performed in the index residing on main memory the status flag in the file is turned on.
   - Whenever the file is written from main memory the status flag is turned off.

2. If the program detects the is index is out-of-date it calls a procedure that reconstruct the index from the data file

Record Addition

- This consists of appending the data file and inserting a new record in the index.
- The rearrangement of the index consists of "sliding down" the records with keys larger than the inserted key and then placing the new record in the opened space.
- Note that this rearrangement is done in main memory

Record Deletion

- This should use the techniques for reclaiming space in files when deleting from the data file.

- We must delete the corresponding entry from the index:
  - Shift all records with keys larger than the key of the deleted record to the previous position (in main memory); or
  - Mark the index entry as deleted in main memory

Record Update

- There are two cases to consider:

- The update **changes the value of the key field**:
  - Treat this as a deletion followed by an insertion

- The update **does not affect the key field**:
  - If record size is unchanged, just modify the data record.
  - If record size changes treat this as a delete/insert sequence.

Indexes too Large to Fit into Main Memory

- The indexes that we have considered before could fit into main memory.
- If this is not the case, we have the following problems:
  - Binary searching of the index file is done on disk, involving several “seekg()” calls
  - Index rearrangement (record addition or deletion) requires shifting on disk

Two Main Alternatives

- Tree-structured index such as B-trees and B+ trees.
- Hashed Organization.

A Simple Index is still Useful

- It allows binary search to obtain a keyed access to a record in a variable-length record file.
- Sorting and maintaining an index is less costly than sorting and maintaining the data file, since the index is smaller
- We can rearrange keys, without moving the data records when there are pinned records

Indexing to Provide Access by Multiple Keys

- Suppose that you are looking at a collection of recordings with the following information about each of them:
  - Identification Number
  - Title
  - Composer or Composers
  - Artist or Artists
  - Label (publisher)

Data File

Address of Record	Actual data record
32	LON|2312|Romeo and Juliet|Prokofiev . . .
77	RCA|2626|Quarter in C Sharp Minor . . .
132	WAR|23699|Touchstone|Corea . . .
167	ANG|3795|Sympony No. 9|Beethoven . . .
211	COL|38358|Nebeaska|Springsteen . . .
256	DG|18807|Symphony No. 9|Beethoven . . .
300	MER|75016|Coq d'or Suite|Rimsky . . .
353	COL|31809|Symphony No. 9|Dvorak . . .
396	DG|139201|Violin Concerto|Beethoven . . .
442	FF|245|Good News|Sweet Honey In The . . .

Indexing to Provide Access by Multiple Keys

- So far, our index only allows key access. i.e., you can retrieve record DG188807, but you cannot retrieve a recording of Beethoven's Symphony no. 9.
- We need to use secondary key fields consisting of album titles, composers, and artists.
- Although it would be possible to relate a secondary key to an actual byte offset, this is usually not done.
- Instead, we relate the secondary key to a primary key which then will point to the actual byte offset.

Indexing to Provide Access by Multiple Keys

- Composer Index:

Secondary key	Primary key
Beethoven	ANG3795
Beethoven	DG139201
Beethoven	DG18807
Beethoven	RCA2626
Corea	WAR23699
Dvorak	COL31809
Prokofiev	LON2312

Record Addition

- When adding a record, an entry must also be added to the secondary key index.
- Store the field in Canonical Form
- There may be duplicates in secondary keys. Keep duplicates in sorted order of primary key

Rerecord Deletion

- Deleting a record implies removing all the references to the record in the primary index and in all the secondary indexes.

- This is too much rearrangement, specially if indexes cannot fit into main memory

An Alternative to Record Deletion

- Delete the record from the data file and the primary index file reference to it. Do not modify the secondary index files.
- When accessing the file through a secondary key, the primary index file will be checked and a deleted record can be identified.
- This results in a lot of saving when there are many secondary keys
- The deleted record still occupy space in the secondary key indexes.
- If a lot of deletions occur, we can periodically cleanup these deleted records also from the secondary key indexes

Rerecord Updating

- There are three types of updates:

1. **The update changes the secondary key:**
   - We have to rearrange the secondary key index to stay in sorted order.

2. **The update changes the primary key:**
   - Update and reorder the primary key index
   - Update the references to primary key index in the secondary key indexes (it may involve some reordering of secondary indexes if secondary key occurs repeated in the file)

Rerecord Updating

3. **Update confined to other fields**

   - This will not affect secondary key indexes.

   - The primary key index may be affected if the location of record changes in the data file.

Thank You !