ICOM 6005 – Database Management Systems Design

Dr. Manuel Rodríguez-Martínez

Electrical and Computer Engineering Department

Query Evaluation Techniques

- Read :
  - Chapter 12, sec 12.1-12.3
  - Chapter 13
- Purpose:
  - Study different algorithms to execute (evaluate) SQL relational operators
    - Selection
    - Projection
    - Joins
    - Aggregates
    - Etc.

Selection and SF

- Selections can have where clause with CNF, or DNF.
- Selectivity factor for CNF case:
  - If where clause is of the form

    p1 and p2 and … and pn,

  - then

$$SF = SF_{p1} \wedge SF_{p2} \wedge \ldots \wedge SF_{pn}$$

  - Assume that all predicates are independent
  - Exsmple:

    Select sid, sname

    From Students

    Where gpa= 4.0 AND age < 25 AND sname <> “Bob”;

Some examples

- Query:

  Q1: Select sname, sage
  From Students where gpa = 4.0;

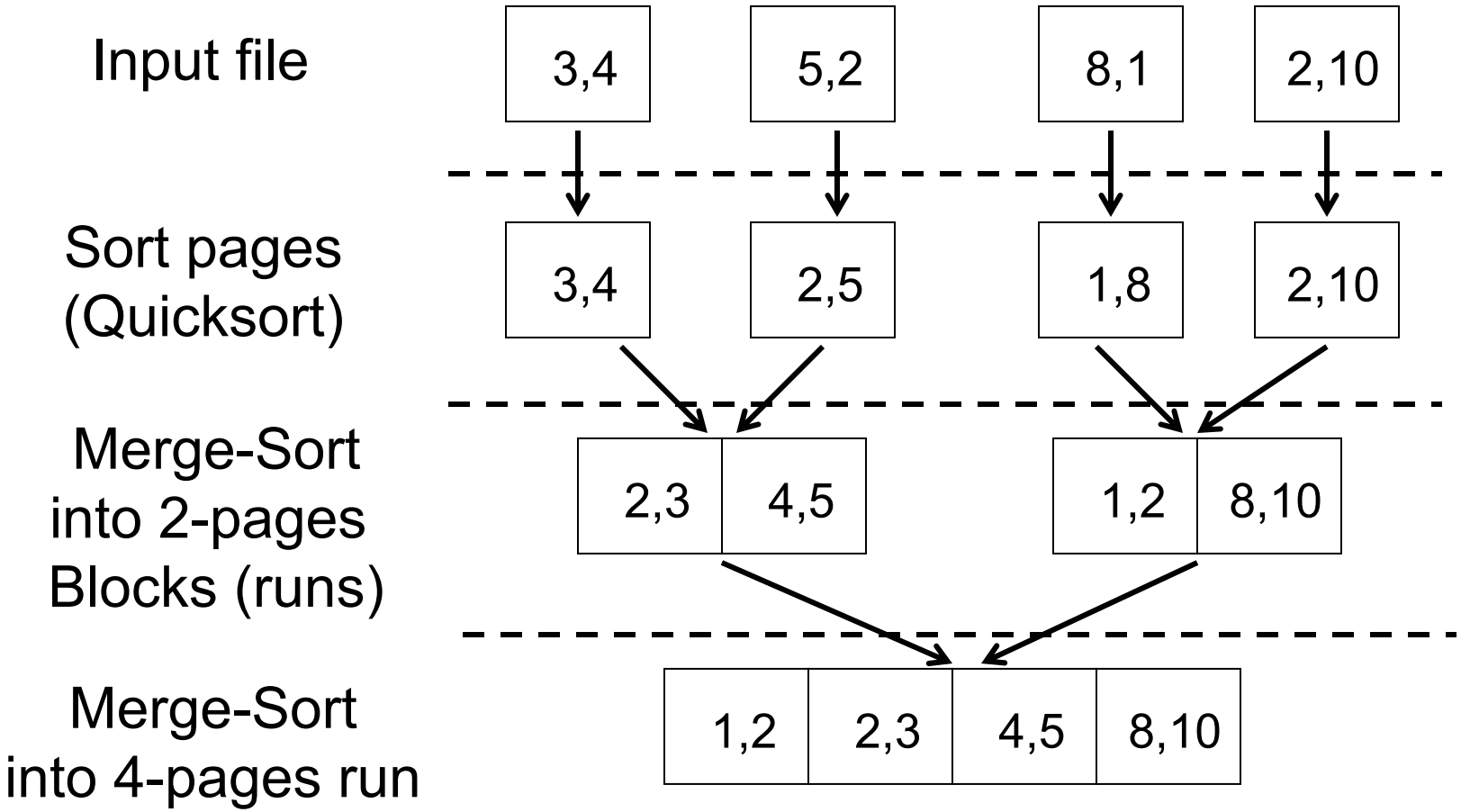
  Q2: Select sname, gpa
  From Students where gpa > 3.50 AND age < 25;

- Information for Students:
  - Cardinality = 5,000
  - #tuples / page = 100
  - SF gpa> 3.50 = 10%
  - SF age < 25 = 90%
- Get selectivity of each predicate? What about where clause? Which predicate should go first?

The Role of sorting

- Sorting plays a pivotal role in the implementation of relational operators and in choosing access path
- Idea: If some operator need to pass several times over the tuples of a table, sorting might speed things up
- Examples:
  - Sorting for duplicate elimination in projections
  - Sort-merge join
  - Sorting for aggregation
  - Sorting for order by clauses
- Question: Since a table R can have gigabytes of worth of data, how do we sort?
  - Answer: External Sorting

External Sorting



Progression of the algorithm

- Pass 0: produces 2^k sorted runs of 1 page
- Pass 1: produces 2^{k-1} sorted runs of 2 pages
- Pass 2: produces 2^{k-2} sorted runs of 4 pages
- Pass 3: produces 2^{k-3} sorted runs of 6 pages
- …
- Pass k: produces 1 sorted run 2^k tuples
- The costs of external sorting:
  - 2 I/O per page per pass (1 read, 1 write)
  - Number of passes: $\lceil \log_2 N \rceil + 1$, N is the # of pages
  - Total cost: $2N * (\lceil \log_2 N \rceil + 1)$

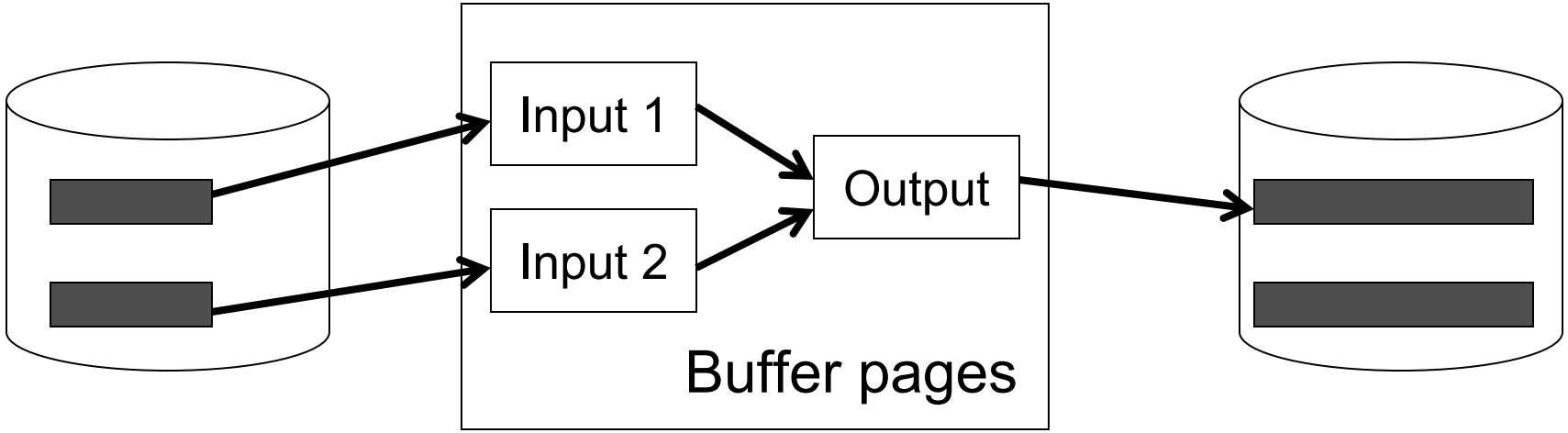
The idea behind External sorting

- Phase I, sort each page in memory using an in-memory sorting algorithm
  - Often Quicksort is used
  - Requires 1 pass over the relation to sort
  - Each page is called a 1-page run
    - Run is a collection of pages with sorted tuples, stored as a file
- Phase II, merge sort
  1. Let i = 1
  2. Start sort merging pairs of runs of size i to build runs of size 2i
  3. When all runs of size i have been consumed
     - If only one run remains, finish
     - else set i = 2i, and goto step 2

Implementing External Sorting

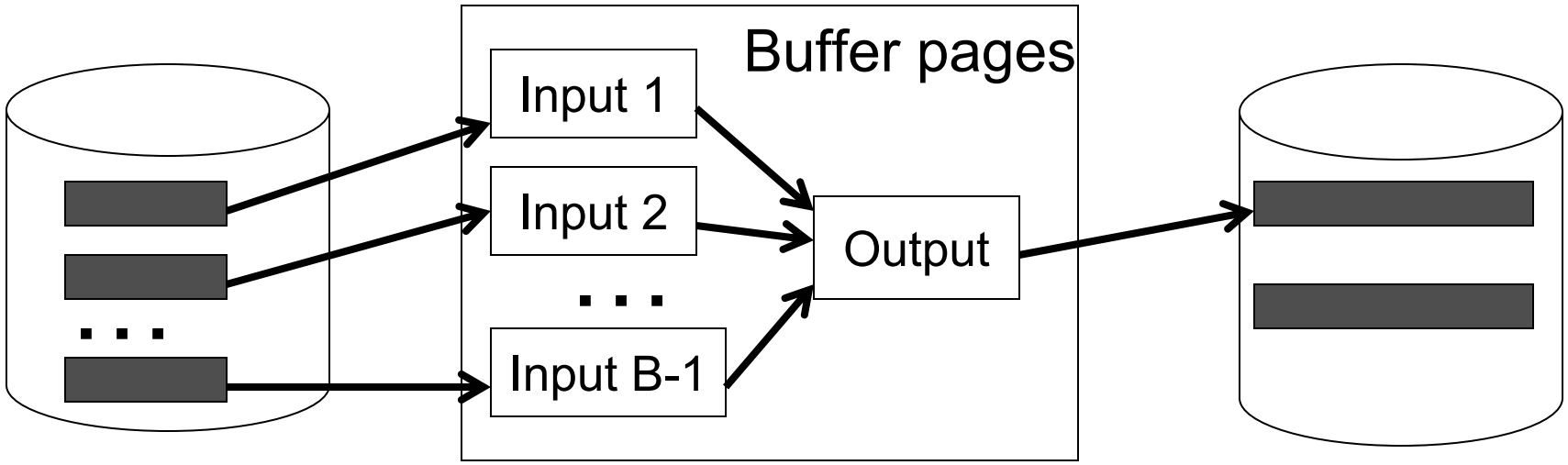
- Each run is stored in a temporary file
- Worst case, you need three buffer pages
  - 2 pages for input
  - 1 page for output
- Usage of pages
  - One is used to keep a page of tuples from a run A
  - The other is used to keep a page of tuples from a run B
  - Third page becomes a merged and sorted page to become part of a new run C, with size twice that of the runs A and B
- In practice, you have B pages available
  - B – 1 are used to input runs
  - 1 is used for output

Three buffer pages



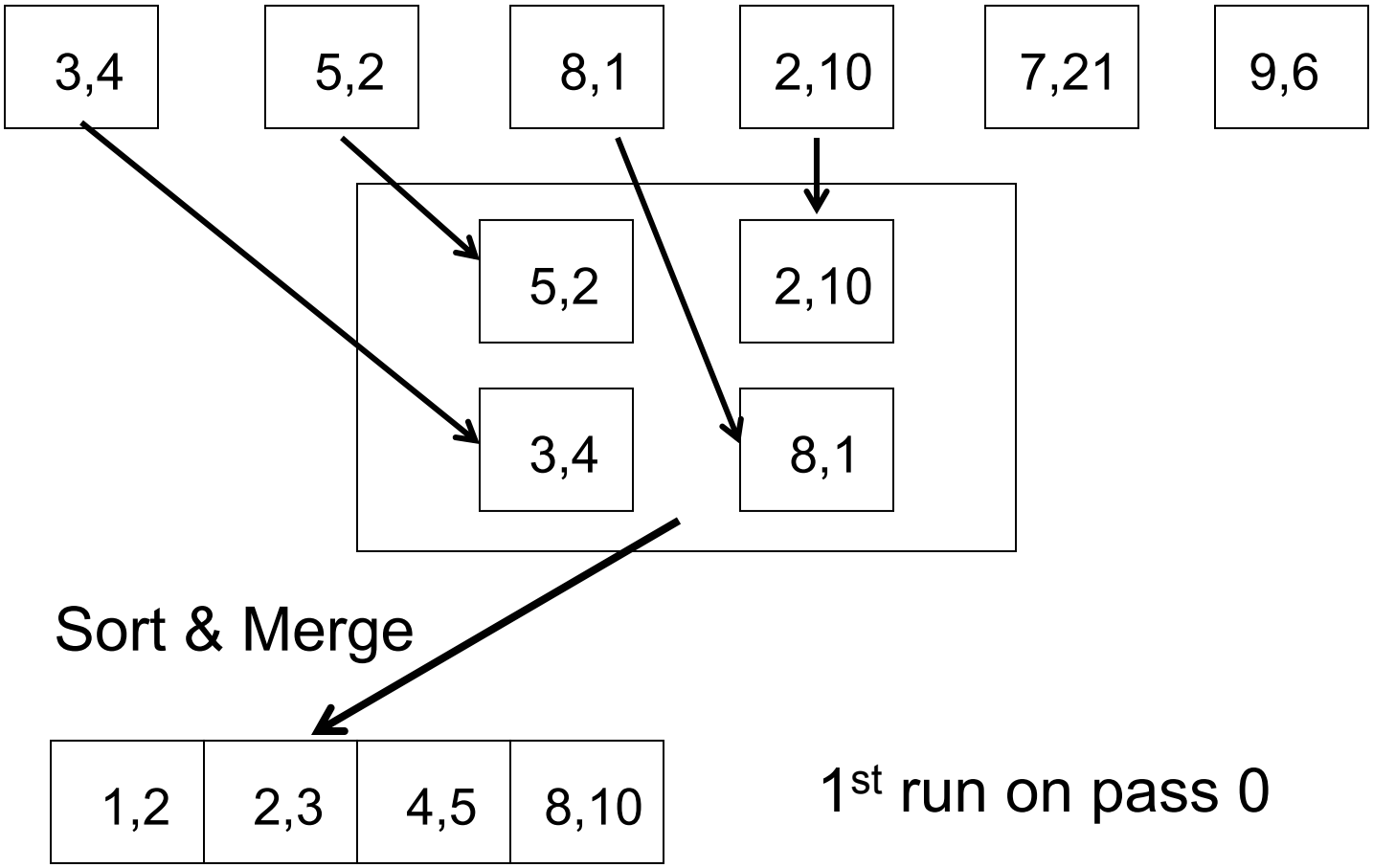
This is the minimal barebones scheme

Reality: B buffer pages

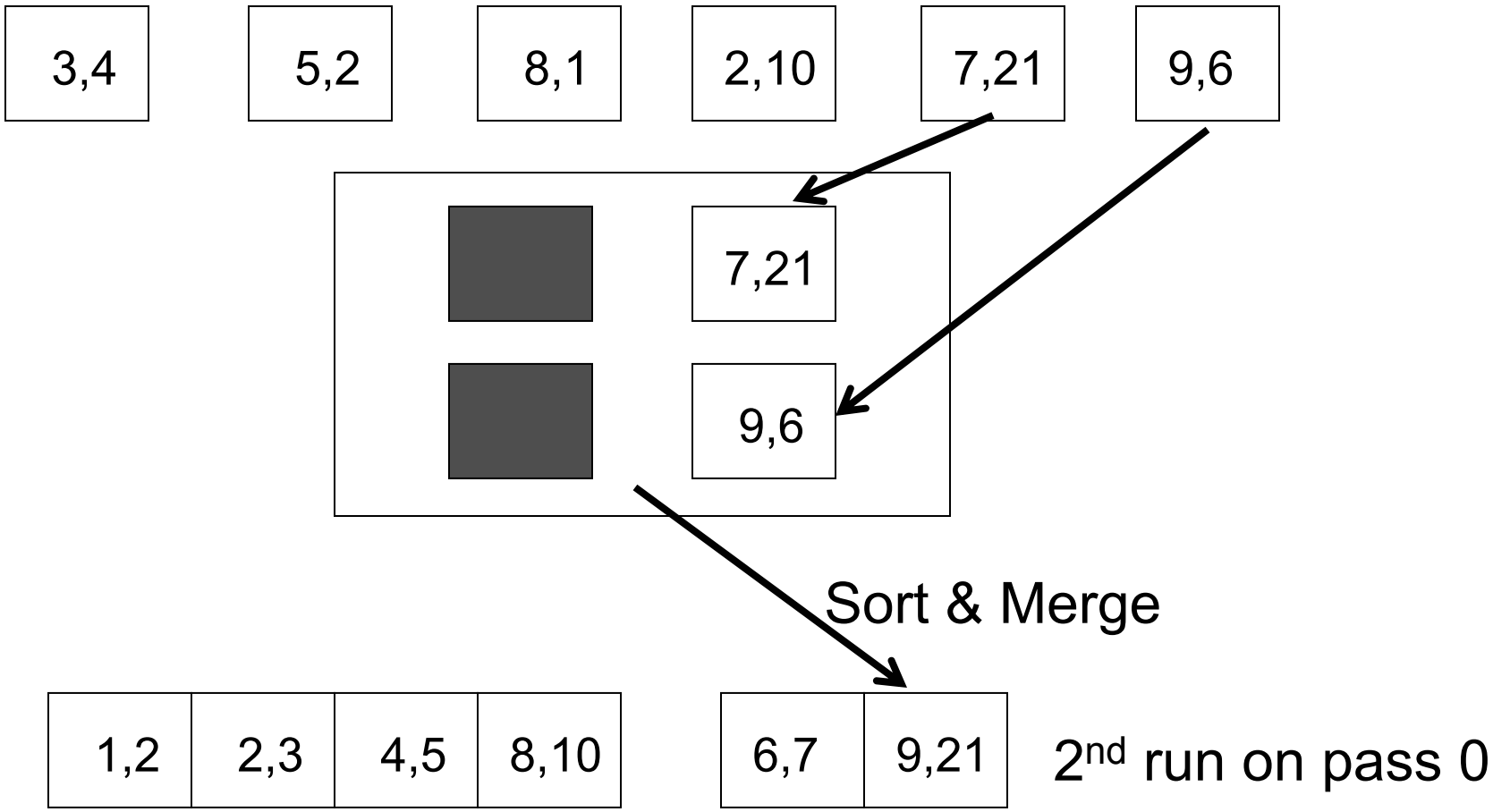


This is the minimal barebones scheme

Sorting with B buffers: Pass 0



Sorting with B buffers: Pass 0 (cont.)



Sketch of Algorithm

- Pass 0:
  - Read B pages at a time,
  - sort each page in memory (e.g. quick sort or heap sort)
  - Write run of size B to disk. This will produce N/B runs, where N = NPages(R) for relation R
- Passes 1, 2, …, K
  1. Use B -1 buffers to read a page from each run of size i
  2. Do a (B-1)-way sort merge to produce a run with the size 2i. Each page is first kept in ouput buffer, then written
  3. Repeat (2) until all runs of the current size have been merged
  4. If only only run is left, exit.
  5. Goto (1)

Leveraging on buffers

- By using B buffers first pass builds $\lceil N/B \rceil$ runs, where N is NPages(R) for a relation R
  - Originally we had N runs …
- Moreover, the total number of passes to do sorting is decreased by doing (B-1)-way merging operations
- The cost for sorting a relation R, with N pages using B buffers (1 for output, B-1 for input) is:
  - I/O Cost = $\lceil \log_{B-1} \lceil N/B \rceil \rceil + 1$
  - Must do (B-1)-way merging operations
- Example: For table R, N = 10000, B = 5
  - I/O Cost = $\lceil \log4 \lceil 10000/5 \rceil \rceil + 1 = 7$ I/Os
  - I/O Cost with 3 buffers= $\lceil \log2\ 10000 \rceil + 1 = 15$ I/Os

Implementing Selection Operator

- Selection operator can be done via:
  - File Scan
    - Unsorted data
    - Sorted data
  - Index Scan
    - B+-tree
    - Hash Tree
- Each access path has very different costs
- File Scans fetch the data and then apply predicates.
- Index scan combine the search with predicate evaluation
  - Search on the index is implicit predicate evaluation.

Scenario

- Query:

  Q1: Select sname, sage
  From Students where gpa = 4.0;

  Q2: Select sname, gpa
  From Students where gpa > 3.50 AND age < 25;

- Information for Students:
  - Cardinality = 5,000
  - #tuples / page = 100
  - SF gpa> 3.50 = 10%
  - SF age < 25 = 90%
- Get selectivity of each predicate? What about where clause? Which predicate should go first?

Costs for Selection Access paths

1. No index, and data not sorted for table R
   - Algorithm: Read each tuples, and evaluate the predicates in the where clause for each one.
   - Cost: Read entire relation R = *NPages(R) I/Os*
2. Data Sorted on attribute in a predicate: *Attr = value*
   - Algorithm: Do binary search to find first attribute that matches, then scan subsequent pages until no attribute matches the condition
   - Cost: (*log2(NPages(R)) + # of pages with matches) !/Os*

- **Do not sort the data just to run a selection!**

Cost for selections with Hash Index

- Hash index, and predicate : *Attr* = *value*
  - Algorithm: Probe hash table to find page with values.
    - Clustered: Fetch records, and read any overflow page
    - Cost: (2 + #number of overflow pages) * I/O
    - Unclustered: Fetch records, fetch data pages, and do the same for overflow pages.
    - Cost: Variable, in the worst case each search key forces us to read a different page in the data file.
    - Often we don't use unclustered index for selections
    - Use selectivity to make a good guess