Physical optimization (select, sort, projection)

Knowledge Objectives

1. Explain how several indexes can be used to solve a complex selection predicate over one table
2. Enumerate three ways to have a table sorted
3. Enumerate seven operations that involve sorting in query processing
4. Explain the merge-sort algorithm
5. Explain how duplicates can be removed from a table depending on the data structure it has (i.e., plain file, B+, Hash, Clustered index)

Understanding Objectives

1. Find which indexes would be used to solve a multi-clause selection predicate over one table
2. Calculate the approximate cost of a merge-sort operation, given the available memory, the number of tuples in the table, and the number of tuples that fit in a disk block

Selection algorithm

1. Put the predicate in CNF
    1. Remove negations of parenthesis
        - NOT (A OR B) = NOT A AND NOT B
        - NOT (A AND B) = NOT A OR NOT B
    2. Move disjunctions into the parenthesis
        - (A AND B) OR C = (A OR C) AND (B OR C)
        - (A AND B) OR (C AND D) = (A OR C) AND (A OR D) AND (B OR C) AND (B OR D)
        - (A AND B) OR (C AND B) = (A OR C) AND B
2. Remove disjunctions if possible
    - For each parenthesis, if indexes can be used for <u>all</u> conditions in it, <u>unite</u> the RID lists resulting from accessing those indexes
3. Remove conjunctions if possible
    - For all parenthesis resolved in the previous step, <u>intersect</u> the RID lists produced
4. For each RID (if any) obtained from previous step, go to the table (<u>by following the RID</u>) to check the remaining predicates

Considerations on the selection algorithm

Let's suppose that we have a predicate in CNF with disjunctions:
$(A_1\ op_1\ v_1$ OR $A_2\ op_2\ v_2)$ AND .... AND $(A_p\ op_p\ v_p)$
being A_i attributes of the same relation and op_i a comparison operator

- If one of the conditions inside the parenthesis does not allow an index to be used, we cannot use any other index
- If no index can be used at all, we should perform a table scan
- Some kinds of indexes are not useful for some conditions
  - B+ useless for "different from"
  - Hash useless for "different from" and inequalities

Example of selection

- We have tree indexes over attributes A, B and C
- We want to select those tuples in R that fulfill:

$$(A=k_1 \text{ AND } B=k_2) \text{ OR } (C=k_3 \text{ AND } D=k_4)$$

Example of selection

- We have tree indexes over attributes A, B and C
- We want to select those tuples in R that fulfill:

$$(A=k_1 \text{ AND } B=k_2) \text{ OR } (C=k_3 \text{ AND } D=k_4)$$

⇓

$$(A \text{ OR } C) \text{ AND } (A \text{ OR } D) \text{ AND } (B \text{ OR } C) \text{ AND } (B \text{ OR } D)$$

Example of selection

- We have tree indexes over attributes A, B and C
- We want to select those tuples in R that fulfill:

$$(A=k_1 \text{ AND } B=k_2) \text{ OR } (C=k_3 \text{ AND } D=k_4)$$

$$\Downarrow$$

$$(\text{A OR C}) \text{ AND } (\text{A OR D}) \text{ AND } (\text{B OR C}) \text{ AND } (\text{B OR D})$$

$\cup$

Example of selection

- We have tree indexes over attributes A, B and C
- We want to select those tuples in R that fulfill:

$$(A=k_1 \text{ AND } B=k_2) \text{ OR } (C=k_3 \text{ AND } D=k_4)$$

$$\Downarrow$$

$$(A \text{ OR } C) \text{ AND } (A \text{ OR } D) \text{ AND } (B \text{ OR } C) \text{ AND } (B \text{ OR } D)$$

$\cup$ $\cup$

Example of selection

- We have tree indexes over attributes A, B and C
- We want to select those tuples in R that fulfill:

$$(A=k_1 \text{ AND } B=k_2) \text{ OR } (C=k_3 \text{ AND } D=k_4)$$

$$\Downarrow$$

$$(A \text{ OR } C) \text{ AND } (A \text{ OR } D) \text{ AND } (B \text{ OR } C) \text{ AND } (B \text{ OR } D)$$

$\cup$ $\cup$

$\cap$

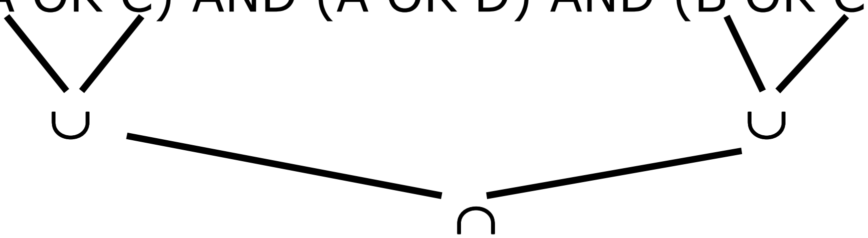
Example of selection

- We have tree indexes over attributes A, B and C
- We want to select those tuples in R that fulfill:

$$(A=k_1 \text{ AND } B=k_2) \text{ OR } (C=k_3 \text{ AND } D=k_4)$$

$$\Downarrow$$

$$(A \text{ OR } C) \text{ AND } (A \text{ OR } D) \text{ AND } (B \text{ OR } C) \text{ AND } (B \text{ OR } D)$$



For each entry in the list of RID resulting from the intersection
Go to the table's file and Check "(A OR D) AND (B OR D)"

External sorting algorithms

- **No index**, with M+1 memory pages
  - $2B \cdot \lceil \log_M B \rceil - B$
- **B+**
  - $\lceil |T|/u \rceil + |T|$

$u = \%carga \cdot 2d = (2/3) \cdot 2d$

- **Clustered**
  - $\lceil 1.5B \rceil$
- **Hash**
  - Useless

External Merge Sort (I)

- Function *sort()*

Assumption: Data is already in memory
Result: Writes the memory pages (blocks) sorted

Disk accesses: B_T

- Function *scan(T)*

Assumption: We have B_T memory pages
Result: T has been read into memory

Disk accesses: B_T

External Merge Sort (II)

- Function *merge(T_1, …, T_M)*

Assumption: T_i are sorted and we have M+1 memory pages
Result: Writes the sorted union of all T_i

t_1 := first(T_1); t_2 := first(T_2); … t_M := first(T_M);
<u>while</u> not (end(T_1) and end(T_2) and … and end(T_M))
  T^{ord} += t_{min};
  t_{min} := next(T_{min});
<u>endWhile</u>

Disk accesses: $2(B_{T_1} + B_{T_2} + \ldots + B_{T_M})$

min = index (1..M) of the T_i with the minimum current value

first ⇒reads the first block into memory
next ⇒ reads a new block if the memory page is empty
+= ⇒ writes a block if buffer is full

External Merge Sort (III)

- Function *mergeSort(T)*

Assumption: We have M+1 memory pages
Result: Sorted T

```
if  B_T <= M then
  scan(T); T^ord:=sort();
else
  T_1^ord :=mergeSort(T_1); … T_M^ord :=mergeSort(T_M);
   T^ord :=merge(T_1^ord, …, T_M^ord);
endif
```

External Merge Sort (III)

- Function *mergeSort(T)*

Assumption: We have M+1 memory pages
Result: Sorted T

if B_T <= M then
  scan(T); T^{ord}:=sort();
else
  T_1^{ord} :=mergeSort(T_1); … T_M^{ord} :=mergeSort(T_M);
  T^{ord} :=merge(T_1^{ord}, …, T_M^{ord});
endif

Disk accesses: $2B_T \cdot \lceil \log_M B_T \rceil$

$B <= M \to 2B =$	$1 * 2B$
$M < B <= M^2 \to 2B + 2B =$	$2 * 2B$
$M^2 < B <= M^3 \to 4B + 2B =$	$3 * 2B$
$M^3 < B <= M^4 \to 6B + 2B =$	$4 * 2B$

External Merge Sort (III)

- Function *mergeSort(T)*

Assumption: We have M+1 memory pages
Result: Sorted T

if B_T <= M then
  scan(T); T^{ord}:=sort();
else
  T_1^{ord} :=mergeSort(T_1); … T_M^{ord} :=mergeSort(T_M);
  T^{ord} :=merge(T_1^{ord}, …, T_M^{ord});
endif

Disk accesses: $2B_T \cdot \lceil \log_M B_T \rceil$

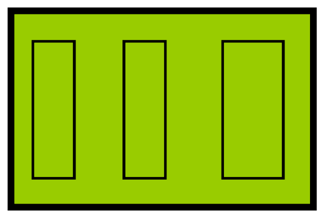
B_T is the real number of blocks (taking into account possible cluster). For the sake of simplicity, we don't distinguish weather output has empty spaces or not

B <= M -> 2B =	1 * 2B
M < B <= M^2 -> 2B + 2B =	2 * 2B
M^2 < B <= M^3 -> 4B + 2B =	3 * 2B
M^3 < B <= M^4 -> 6B + 2B =	4 * 2B

External Merge Sort (IV)

Accesses

M=2



5	3	2	9
3	1	8	4

if B_T <= M then

  scan(T); T^{ord}:=sort();

else

  T_1^{ord} :=mergeSort(T_1);

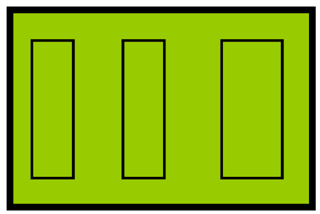
  T_2^{ord} :=mergeSort(T_2);

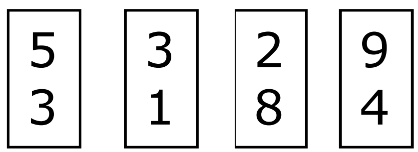
  T^{ord} :=merge(T_1^{ord}, T_2^{ord});

External Merge Sort (IV)

Accesses

M=2

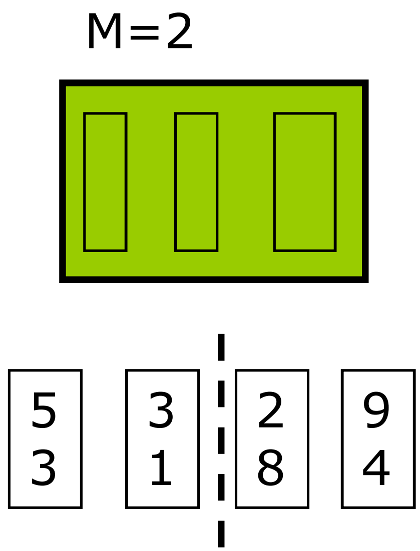




if B_T <= M then
  scan(T); T^{ord}:=sort();
else
  T_1^{ord} :=mergeSort(T_1);
  T_2^{ord} :=mergeSort(T_2);
  T^{ord} :=merge(T_1^{ord}, T_2^{ord});

External Merge Sort (IV)

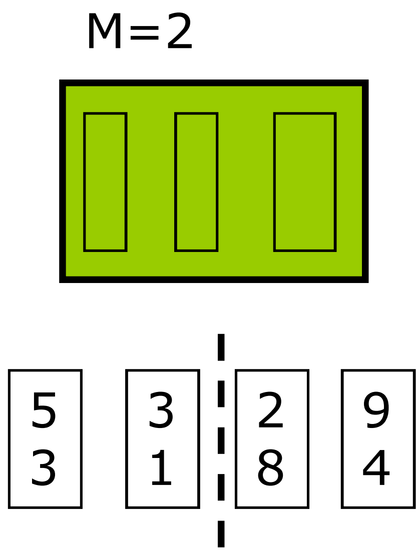
Accesses



if $B_T <= M$ then
    scan(T); T^{ord}:=sort();
else
    T_1^{ord} :=mergeSort(T_1);
    T_2^{ord} :=mergeSort(T_2);
    T^{ord} :=merge(T_1^{ord}, T_2^{ord});

External Merge Sort (IV)

Accesses



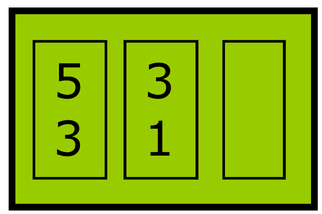
if $B_T <= M$ then
  scan(T); T^{ord}:=sort();
else
  T_1^{ord} :=mergeSort(T_1);
  T_2^{ord} :=mergeSort(T_2);
  T^{ord} :=merge(T_1^{ord}, T_2^{ord});

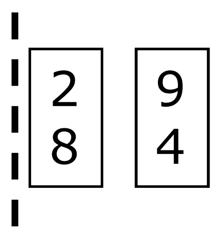
External Merge Sort (IV)

Accesses

R
R

M=2



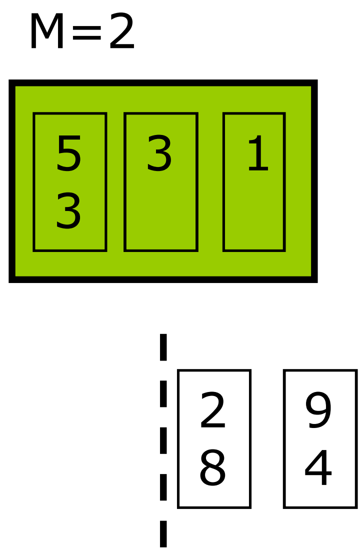


if B_T <= M then
scan(T); T^{ord}:=sort();
else
T_1^{ord} :=mergeSort(T_1);
T_2^{ord} :=mergeSort(T_2);
T^{ord} :=merge(T_1^{ord}, T_2^{ord});

External Merge Sort (IV)

Accesses

R
R

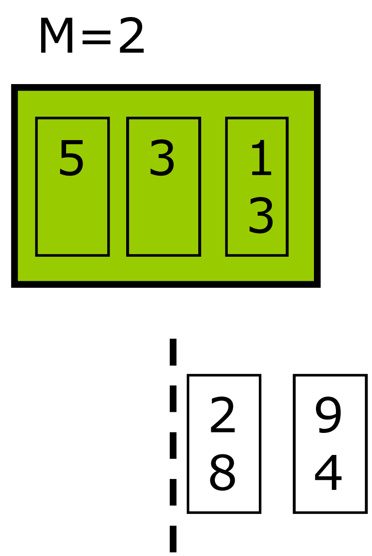


if $B_T <= M$ then
  scan(T); T^{ord}:=sort();
else
  T_1^{ord} :=mergeSort(T_1);
  T_2^{ord} :=mergeSort(T_2);
  T^{ord} :=merge(T_1^{ord}, T_2^{ord});

External Merge Sort (IV)

Accesses

R
R



if B_T <= M then
scan(T); T^{ord}:=sort();

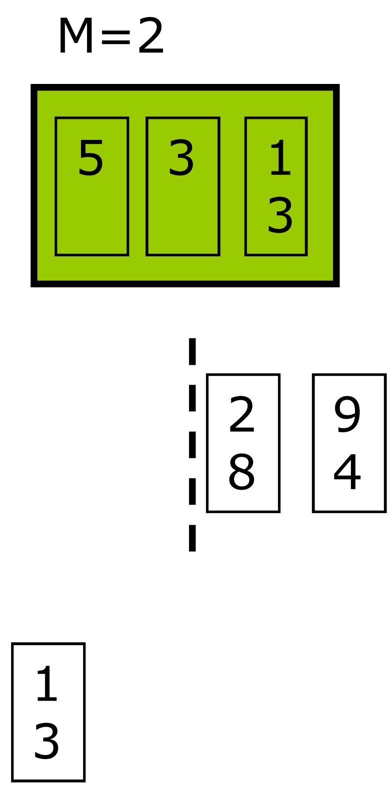
else

T_1^{ord} :=mergeSort(T_1);
T_2^{ord} :=mergeSort(T_2);
T^{ord} :=merge(T_1^{ord}, T_2^{ord});

External Merge Sort (IV)

Accesses

R
R
W



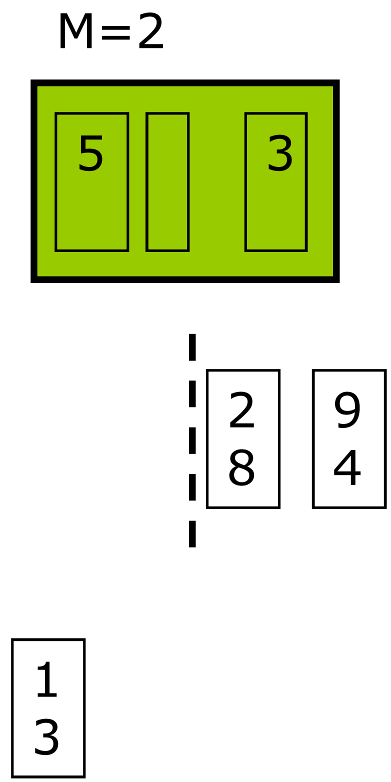
if B_T <= M then
    scan(T); T^{ord}:=sort();

else
    T_1^{ord} :=mergeSort(T_1);
    T_2^{ord} :=mergeSort(T_2);
    T^{ord} :=merge(T_1^{ord}, T_2^{ord});

External Merge Sort (IV)

Accesses

R
R
W

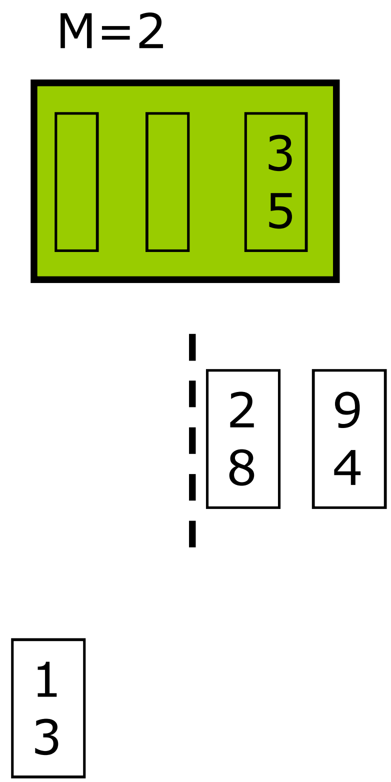


if B_T <= M then
    scan(T); T^{ord}:=sort();
else
    T_1^{ord} :=mergeSort(T_1);
    T_2^{ord} :=mergeSort(T_2);
    T^{ord} :=merge(T_1^{ord}, T_2^{ord});

External Merge Sort (IV)

Accesses

R
R
W

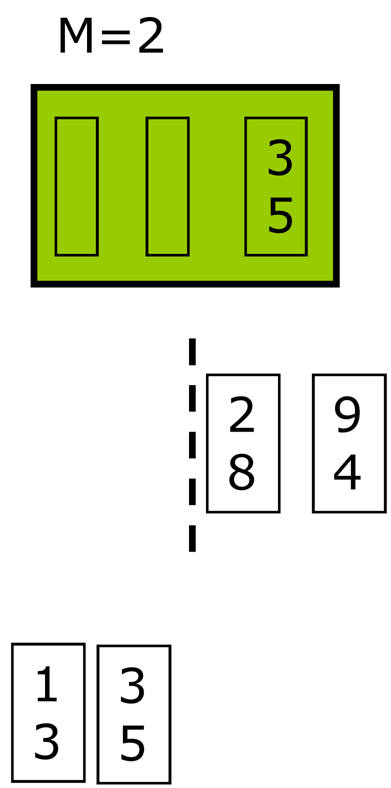


if B_T <= M then
  scan(T); T^{ord}:=sort();
else
  T_1^{ord} :=mergeSort(T_1);
  T_2^{ord} :=mergeSort(T_2);
  T^{ord} :=merge(T_1^{ord}, T_2^{ord});

External Merge Sort (IV)

Accesses

R
R
W
W

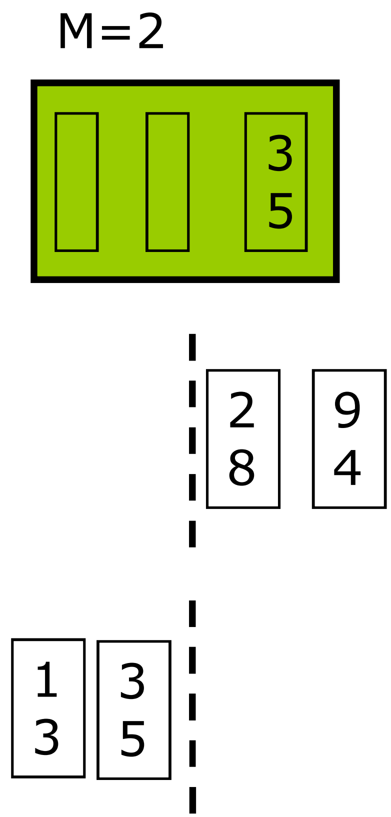


if B_T <= M then
    scan(T); T^{ord}:=sort();
else
    T_1^{ord} :=mergeSort(T_1);
    T_2^{ord} :=mergeSort(T_2);
    T^{ord} :=merge(T_1^{ord}, T_2^{ord});

External Merge Sort (IV)

Accesses

R
R
W
W

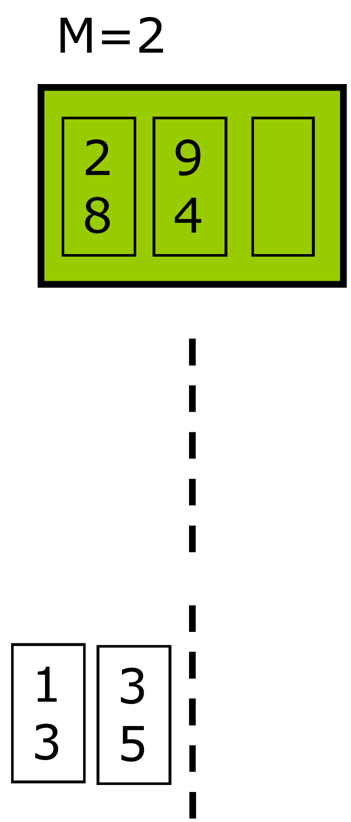


if B_T <= M then
    scan(T); T^{ord}:=sort();
else
    T_1^{ord} :=mergeSort(T_1);
    T_2^{ord} :=mergeSort(T_2);
    T^{ord} :=merge(T_1^{ord}, T_2^{ord});

External Merge Sort (IV)

Accesses

R
R
W
W
R
R

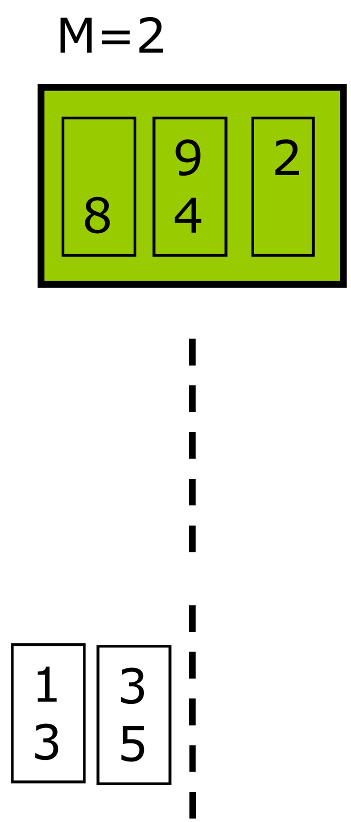


if B_T <= M then
    scan(T); T^{ord}:=sort();
else
    T_1^{ord} :=mergeSort(T_1);
    T_2^{ord} :=mergeSort(T_2);
    T^{ord} :=merge(T_1^{ord}, T_2^{ord});

External Merge Sort (IV)

Accesses

R
R
W
W
R
R



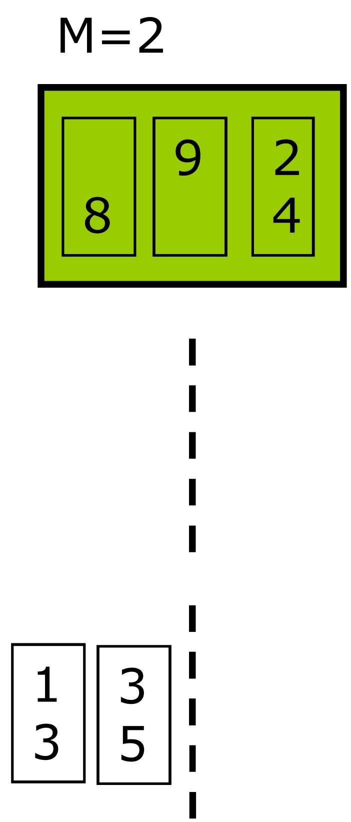
if B_T <= M then
    scan(T); T^{ord}:=sort();

else
    T_1^{ord} :=mergeSort(T_1);
    T_2^{ord} :=mergeSort(T_2);
    T^{ord} :=merge(T_1^{ord}, T_2^{ord});

External Merge Sort (IV)

Accesses

R
R
W
W
R
R



if B_T <= M then
    scan(T); T^{ord}:=sort();

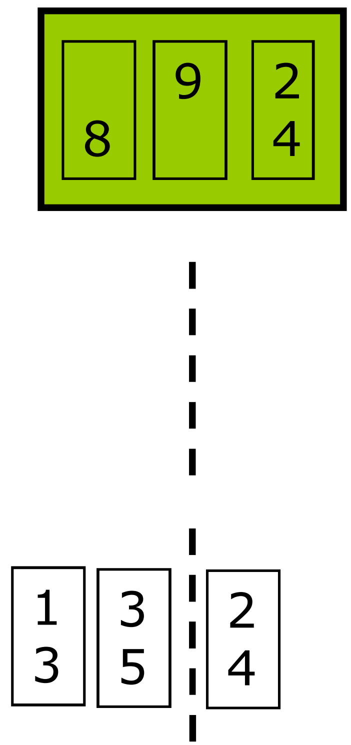
else
    T_1^{ord} :=mergeSort(T_1);
    T_2^{ord} :=mergeSort(T_2);
    T^{ord} :=merge(T_1^{ord}, T_2^{ord});

External Merge Sort (IV)

Accesses

R
R
W
W
R
R
W

M=2

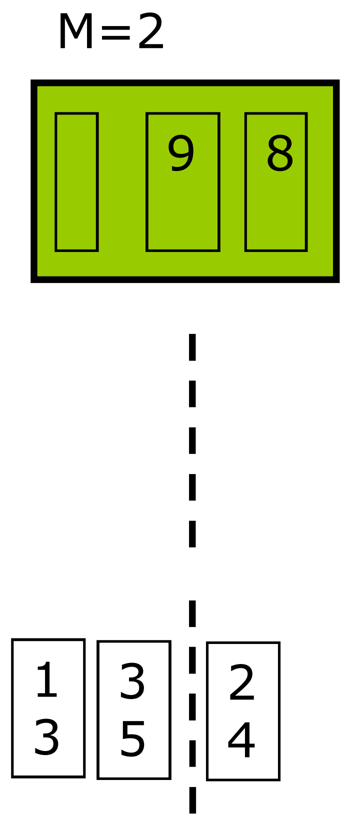


if B_T <= M then
  scan(T); T^{ord}:=sort();
else
  T_1^{ord} :=mergeSort(T_1);
  T_2^{ord} :=mergeSort(T_2);
  T^{ord} :=merge(T_1^{ord}, T_2^{ord});

External Merge Sort (IV)

Accesses

R
R
W
W
R
R
W



if $B_T <= M$ then
    scan(T); T^{ord}:=sort();

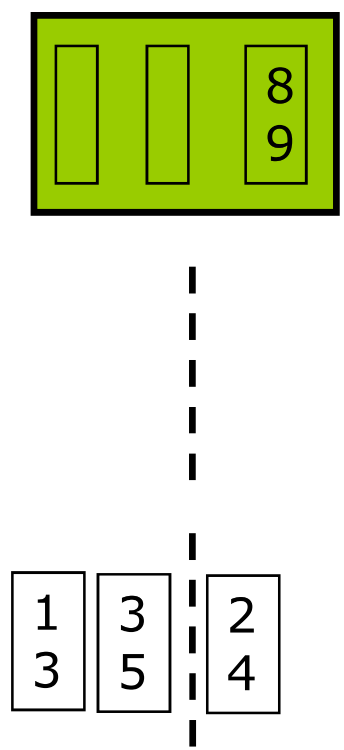
else
    T_1^{ord} :=mergeSort(T_1);
    T_2^{ord} :=mergeSort(T_2);
    T^{ord} :=merge(T_1^{ord}, T_2^{ord});

External Merge Sort (IV)

Accesses

R
R
W
W
R
R
W

M=2



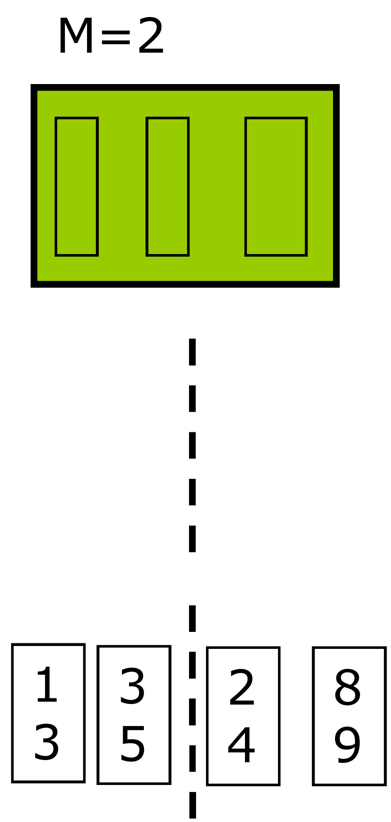
if B_T <= M then
  scan(T); T^{ord}:=sort();

else
  T_1^{ord} :=mergeSort(T_1);
  T_2^{ord} :=mergeSort(T_2);
  T^{ord} :=merge(T_1^{ord}, T_2^{ord});

External Merge Sort (IV)

Accesses

R
R
W
W
R
R
W
W



if $B_T <= M$ then
  scan(T); T^{ord}:=sort();

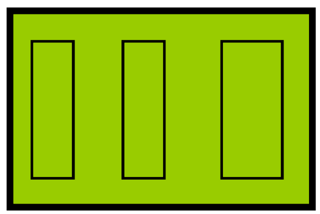
else
  T_1^{ord} :=mergeSort(T_1);
  T_2^{ord} :=mergeSort(T_2);
  T^{ord} :=merge(T_1^{ord}, T_2^{ord});

External Merge Sort (IV)

Accesses

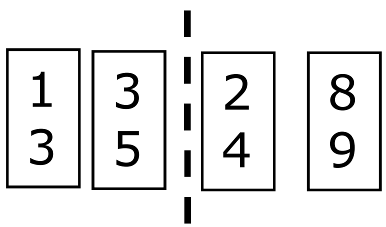
R
R
W
W
R
R
W
W

M=2



if B_T <= M then
scan(T); T^{ord}:=sort();

else
T_1^{ord} :=mergeSort(T_1);
T_2^{ord} :=mergeSort(T_2);
T^{ord} :=merge(T_1^{ord}, T_2^{ord});

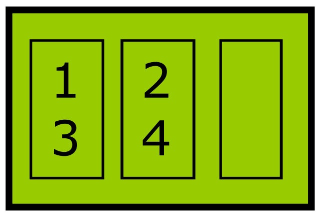


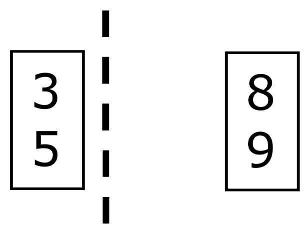
External Merge Sort (IV)

Accesses

R
R
W
W
R
R
W
W
R
R

M=2





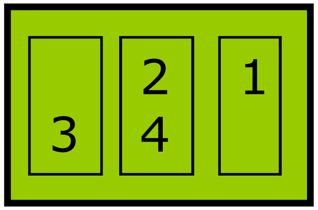
if B_T <= M then
    scan(T); T^{ord}:=sort();
else
    T_1^{ord} :=mergeSort(T_1);
    T_2^{ord} :=mergeSort(T_2);
    T^{ord} :=merge(T_1^{ord}, T_2^{ord});

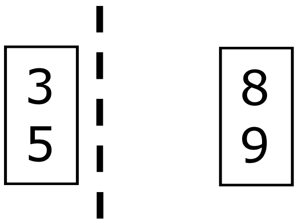
External Merge Sort (IV)

Accesses

R
R
W
W
R
R
W
W
R
R

M=2





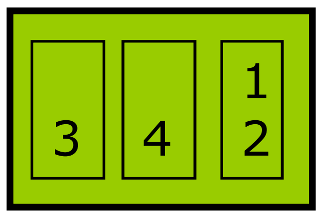
if B_T <= M then
    scan(T); T^{ord}:=sort();
else
    T_1^{ord} :=mergeSort(T_1);
    T_2^{ord} :=mergeSort(T_2);
    T^{ord} :=merge(T_1^{ord}, T_2^{ord});

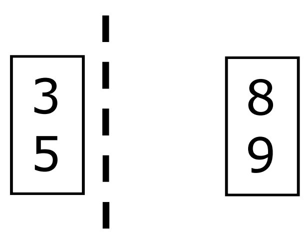
External Merge Sort (IV)

Accesses

R
R
W
W
R
R
W
W
R
R

M=2





if B_T <= M then
scan(T); T^{ord}:=sort();

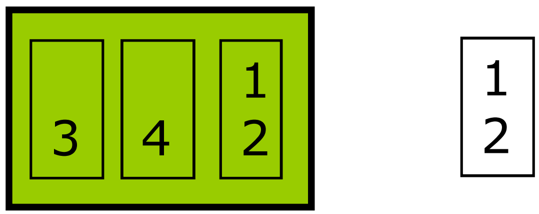
else
T_1^{ord} :=mergeSort(T_1);
T_2^{ord} :=mergeSort(T_2);
T^{ord} :=merge(T_1^{ord}, T_2^{ord});

External Merge Sort (IV)

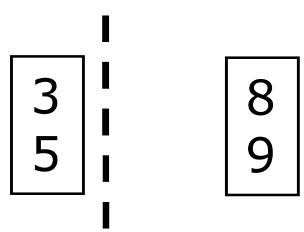
Accesses

R
R
W
W
R
R
W
W
R
R

M=2



if B_T <= M then
scan(T); T^{ord}:=sort();
else
T_1^{ord} :=mergeSort(T_1);
T_2^{ord} :=mergeSort(T_2);
T^{ord} :=merge(T_1^{ord}, T_2^{ord});

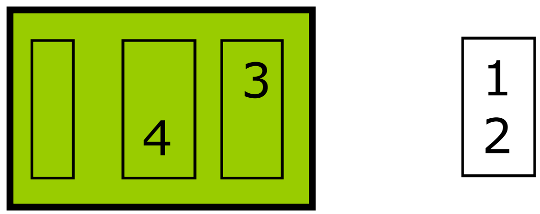


External Merge Sort (IV)

Accesses

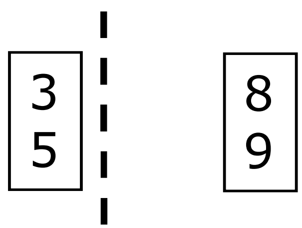
R
R
W
W
R
R
W
W
R
R

M=2



if B_T <= M then
scan(T); T^{ord}:=sort();

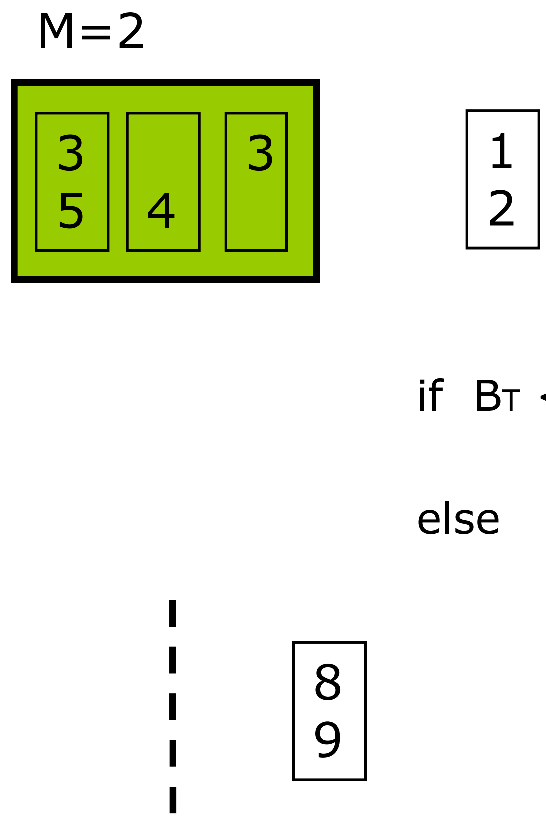
else
T_1^{ord} :=mergeSort(T_1);
T_2^{ord} :=mergeSort(T_2);
T^{ord} :=merge(T_1^{ord}, T_2^{ord});



External Merge Sort (IV)

Accesses

R
R
W
W
R
R
W
W
R
R
R

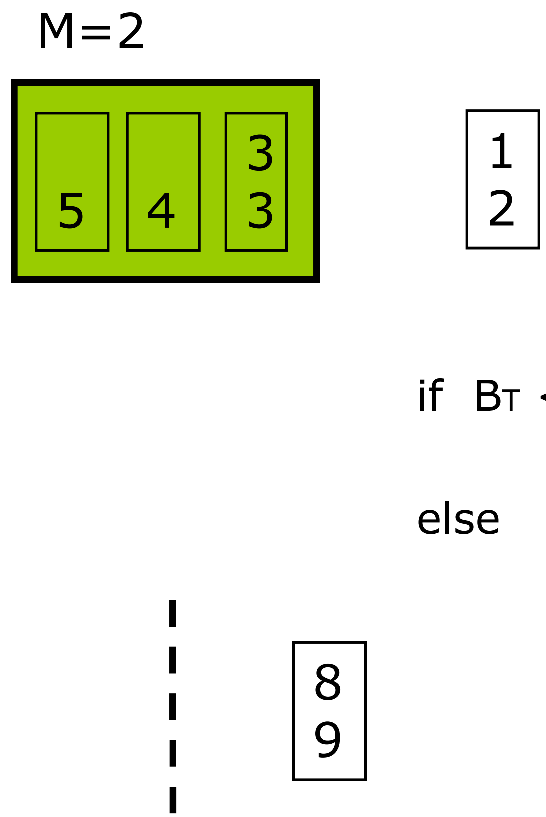


if B_T <= M then
    scan(T); T^{ord}:=sort();
else
    T_1^{ord} :=mergeSort(T_1);
    T_2^{ord} :=mergeSort(T_2);
    T^{ord} :=merge(T_1^{ord}, T_2^{ord});

External Merge Sort (IV)

Accesses

R
R
W
W
R
R
W
W
R
R
R




```
if  BT <= M then
          scan(T); Tord:=sort();
else
          T1ord :=mergeSort(T1);
          T2ord :=mergeSort(T2);
          Tord :=merge(T1ord, T2ord);
```

External Merge Sort (IV)

Accesses

R
R
W
W
R
R
W
W
R
R
R

M=2

5 4

1
2

3
3

8
9

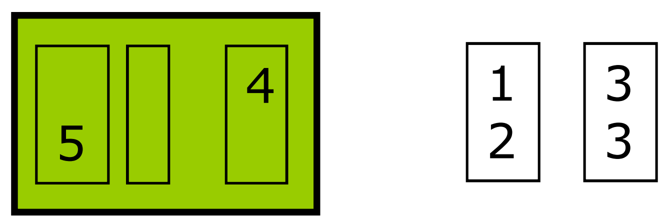
if B_T <= M then
scan(T); T^{ord}:=sort();
else
T_1^{ord} :=mergeSort(T_1);
T_2^{ord} :=mergeSort(T_2);
T^{ord} :=merge(T_1^{ord}, T_2^{ord});

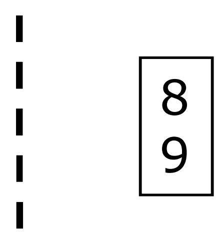
External Merge Sort (IV)

Accesses

R
R
W
W
R
R
W
W
R
R
R

M=2



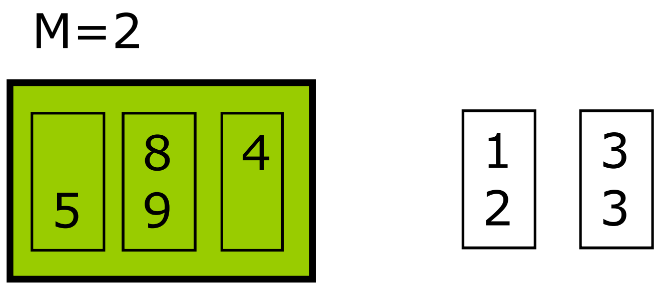


if B_T <= M then
    scan(T); T^{ord}:=sort();
else
    T_1^{ord} :=mergeSort(T_1);
    T_2^{ord} :=mergeSort(T_2);
    T^{ord} :=merge(T_1^{ord}, T_2^{ord});

External Merge Sort (IV)

Accesses

R
R
W
W
R
R
W
W
R
R
R
R



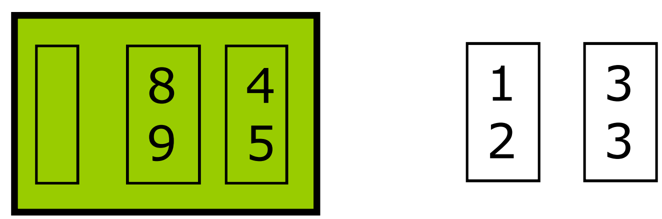
if B_T <= M then
    scan(T); T^{ord}:=sort();
else
    T_1^{ord} :=mergeSort(T_1);
    T_2^{ord} :=mergeSort(T_2);
    T^{ord} :=merge(T_1^{ord}, T_2^{ord});

External Merge Sort (IV)

Accesses

R
R
W
W
R
R
W
W
R
R
R
R

M=2

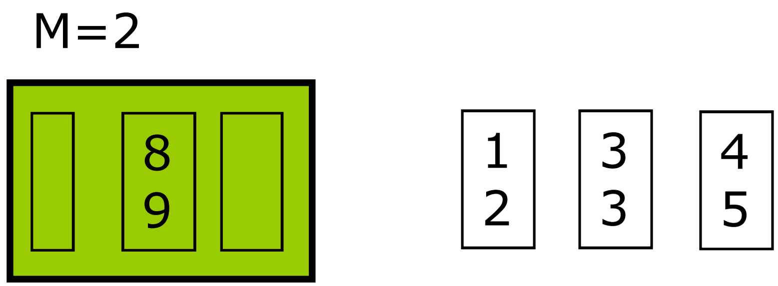


if B_T <= M then
  scan(T); T^{ord}:=sort();
else
  T_1^{ord} :=mergeSort(T_1);
  T_2^{ord} :=mergeSort(T_2);
  T^{ord} :=merge(T_1^{ord}, T_2^{ord});

External Merge Sort (IV)

Accesses

R
R
W
W
R
R
W
W
R
R
R
R



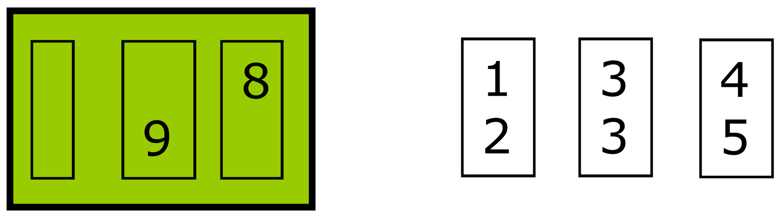
if B_T <= M then
  scan(T); T^{ord}:=sort();
else
  T_1^{ord} :=mergeSort(T_1);
  T_2^{ord} :=mergeSort(T_2);
  T^{ord} :=merge(T_1^{ord}, T_2^{ord});

External Merge Sort (IV)

Accesses

R
R
W
W
R
R
W
W
R
R
R
R

M=2



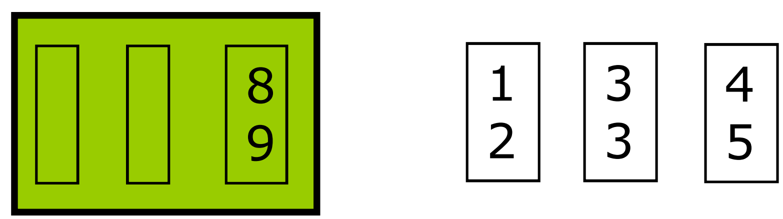
if B_T <= M then
    scan(T); T^{ord}:=sort();
else
    T_1^{ord} :=mergeSort(T_1);
    T_2^{ord} :=mergeSort(T_2);
    T^{ord} :=merge(T_1^{ord}, T_2^{ord});

External Merge Sort (IV)

Accesses

R
R
W
W
R
R
W
W
R
R
R
R

M=2

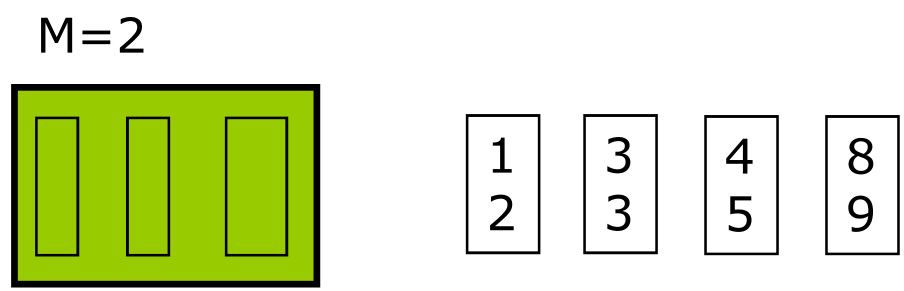



```
if  BT <= M then
            scan(T); Tord:=sort();
else
            T1ord :=mergeSort(T1);
            T2ord :=mergeSort(T2);
            Tord :=merge(T1ord, T2ord);
```

External Merge Sort (IV)

Accesses

R
R
W
W
R
R
W
W
R
R
R
R



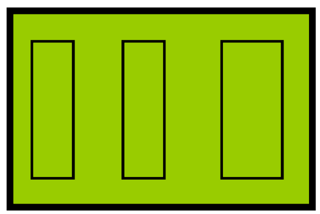
if B_T <= M then
  scan(T); T^{ord}:=sort();
else
  T_1^{ord} :=mergeSort(T_1);
  T_2^{ord} :=mergeSort(T_2);
  T^{ord} :=merge(T_1^{ord}, T_2^{ord});

External Merge Sort (IV)

Accesses

R
R
W
W
R
R
W
W
R
R
R
R

M=2



1 2	3 3	4 5	8 9

if B_T <= M then
    scan(T); T^{ord}:=sort();
else
    T_1^{ord} :=mergeSort(T_1);
    T_2^{ord} :=mergeSort(T_2);
    T^{ord} :=merge(T_1^{ord}, T_2^{ord});

Usefulness of sorting algorithms

- ORDER BY
- Duplicates removal
  - DISTINCT
  - UNION
- GROUP BY
- Join
- Difference (anti-join)
- Massive index load

Projection algorithms

1. Attribute removal
   a) There is another operation
      - 0
   b) There is no other operation
      - B
2. Duplicate removal
   a) No index, with M+1 memory pages
      - $2B \cdot \lceil \log_M B \rceil - B$
   b) B+, useful if M and R are small with regard to B
      - $\lceil |T|/u \rceil$ (probably $+|T|$)
   c) Clustered
      - $\lceil 1.5B \rceil$
   d) Hash, useful if M and R are small with regard to B
      - $\lceil 1.25(|T|/2d) \rceil$ (probably $+|T|$)

$u = \%load \cdot 2d = (2/3) \cdot 2d$

Summary

- Selection algorithms
- External sort algorithms
- Projection algorithms

Bibliography

- G. Gardarin and P. Valduriez. *Relational Databases and Knowledge Bases*. Addison-Wesley, 1998
- R. Ramakrishnan and J. Gehrke. *Database Management Systems.* McGraw-Hill, 3rd Edition, 2003
- S. Lightstone, T. Teorey and T. Nadeau. *Physical Database Design*. Morgan Kaufmann, 2007
- J. Sistac. *Sistemes de Gestió de Bases de Dades*. Editorial UOC, 2002
- Y. Ioannidis. *Query Optimization*. ACM Computing Surveys, vol. 28, num. 1, March 1996.
- J. Lewis. *Cost-Based Oracle Fundamentals*. Apress, 2006