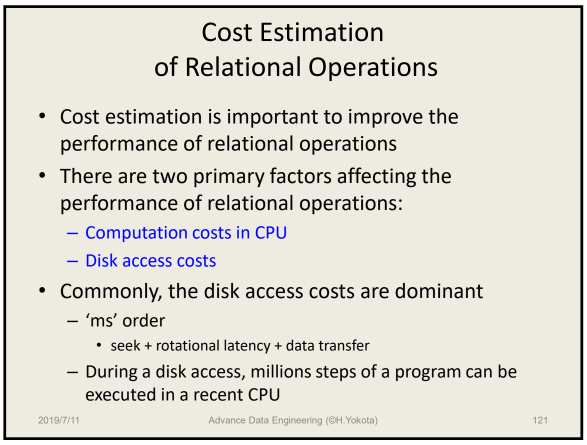
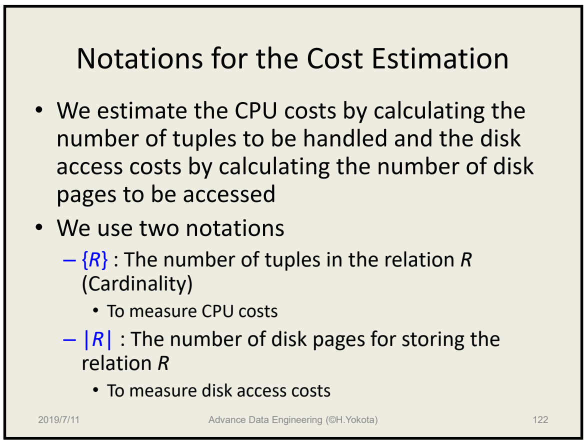


Cost Estimation of Relational Operations

- Cost estimation is important to improve the performance of relational operations
- There are two primary factors affecting the performance of relational operations:
  - Computation costs in CPU
  - Disk access costs
- Commonly, the disk access costs are dominant
  - 'ms' order
    - seek + rotational latency + data transfer
  - During a disk access, millions steps of a program can be executed in a recent CPU

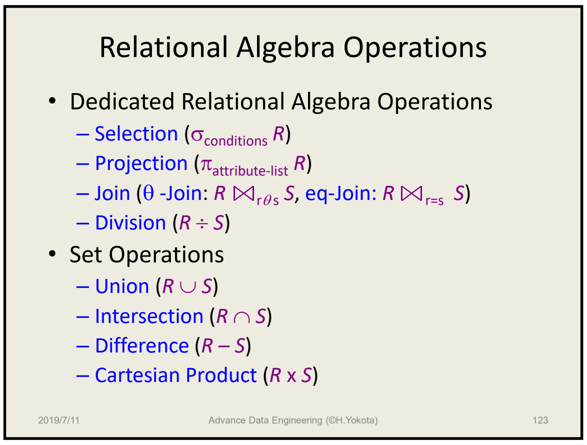
2019/7/11 Advance Data Engineering (©H.Yokota) 121



Notations for the Cost Estimation

- We estimate the CPU costs by calculating the number of tuples to be handled and the disk access costs by calculating the number of disk pages to be accessed
- We use two notations
  - $\{R\}$: The number of tuples in the relation *R* (Cardinality)
    - To measure CPU costs
  - $|R|$: The number of disk pages for storing the relation *R*
    - To measure disk access costs

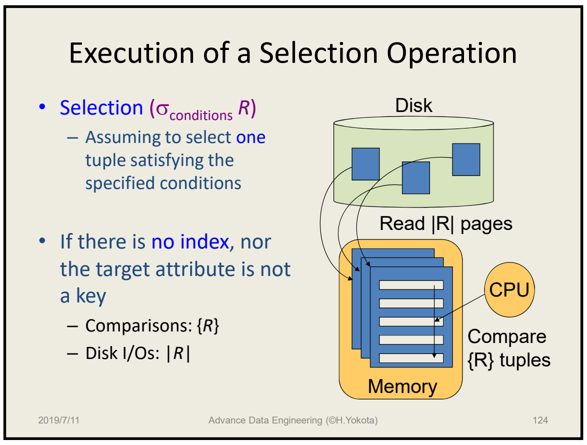
2019/7/11 Advance Data Engineering (©H.Yokota) 122



Relational Algebra Operations

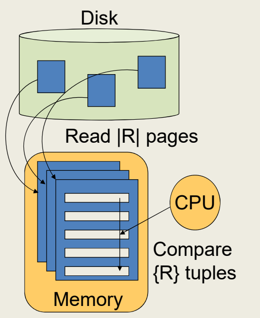
- Dedicated Relational Algebra Operations
  - Selection ($\sigma_{conditions}\ R$)
  - Projection ($\pi_{attribute\text{-}list}\ R$)
  - Join (θ -Join: $R \bowtie_{r\theta s} S$, eq-Join: $R \bowtie_{r=s} S$)
  - Division ($R \div S$)
- Set Operations
  - Union ($R \cup S$)
  - Intersection ($R \cap S$)
  - Difference ($R - S$)
  - Cartesian Product (R x S)

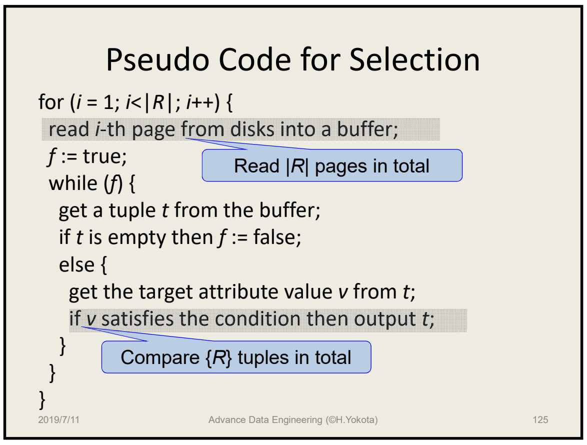
2019/7/11 Advance Data Engineering (©H.Yokota) 123



Execution of a Selection Operation

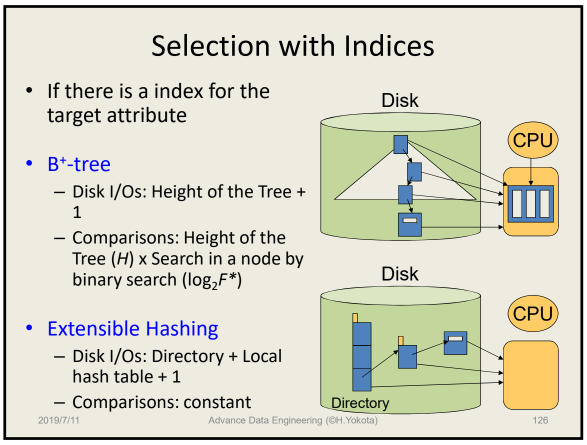
- Selection ($\sigma_{conditions}$ R)
  - Assuming to select one tuple satisfying the specified conditions
- If there is no index, nor the target attribute is not a key
  - Comparisons: $\{R\}$
  - Disk I/Os: $|R|$





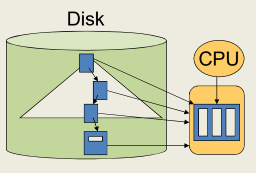
Pseudo Code for Selection

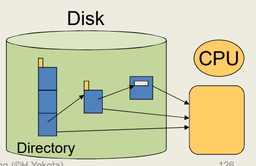
```
for (i = 1; i<|R|; i++) {
  read i-th page from disks into a buffer;
  f := true;                    [Read |R| pages in total]
  while (f) {
    get a tuple t from the buffer;
    if t is empty then f := false;
    else {
      get the target attribute value v from t;
      if v satisfies the condition then output t;
    }                           [Compare {R} tuples in total]
  }
}
```

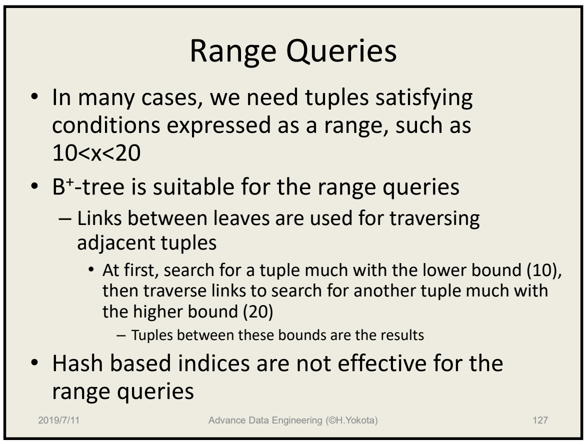


Selection with Indices

- If there is a index for the target attribute
- B$^+$-tree
  - Disk I/Os: Height of the Tree + 1
  - Comparisons: Height of the Tree (H) x Search in a node by binary search ($\log_2 F^*$)
- Extensible Hashing
  - Disk I/Os: Directory + Local hash table + 1
  - Comparisons: constant



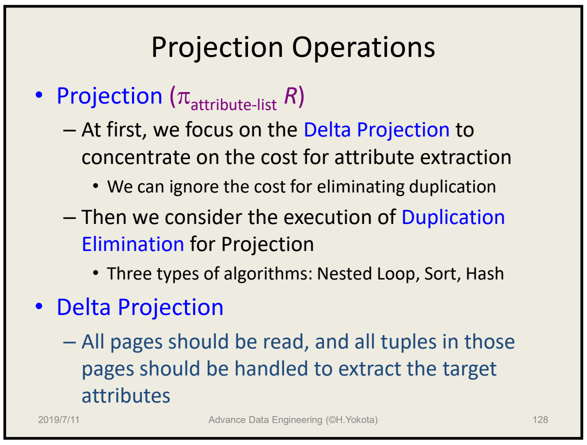




Range Queries

- In many cases, we need tuples satisfying conditions expressed as a range, such as 10<x<20
- B$^+$-tree is suitable for the range queries
  - Links between leaves are used for traversing adjacent tuples
    - At first, search for a tuple much with the lower bound (10), then traverse links to search for another tuple much with the higher bound (20)
      - Tuples between these bounds are the results
- Hash based indices are not effective for the range queries

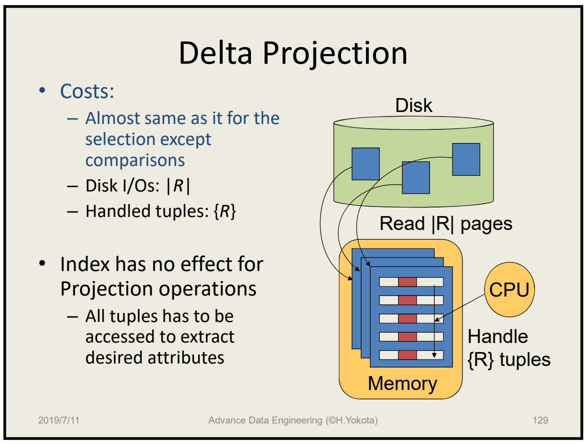
2019/7/11 Advance Data Engineering (©H.Yokota) 127



Projection Operations

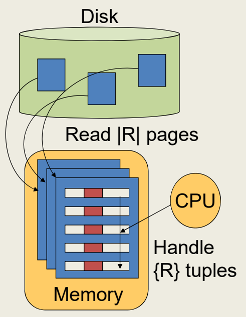
- Projection ($\pi_{\text{attribute-list}}\ R$)
  - At first, we focus on the Delta Projection to concentrate on the cost for attribute extraction
    - We can ignore the cost for eliminating duplication
  - Then we consider the execution of Duplication Elimination for Projection
    - Three types of algorithms: Nested Loop, Sort, Hash
- Delta Projection
  - All pages should be read, and all tuples in those pages should be handled to extract the target attributes

2019/7/11 Advance Data Engineering (©H.Yokota) 128



Delta Projection

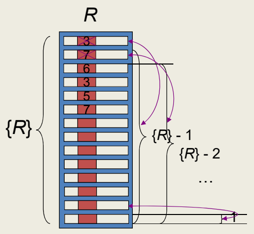
- Costs:
  - Almost same as it for the selection except comparisons
  - Disk I/Os: $|R|$
  - Handled tuples: $\{R\}$
- Index has no effect for Projection operations
  - All tuples has to be accessed to extract desired attributes



2019/7/11 Advance Data Engineering (©H.Yokota) 129

Nested Loop base Duplication Elimination (NLDE)

- At first, we concentrate on a straightforward algorithm ignoring disk I/Os
- Search the same value from the top by scanning rest tuples
- Comparisons:
  - ({R} - 1) + ({R} - 2) +…+ 1
  - ({R} x ({R} - 1))/ 2



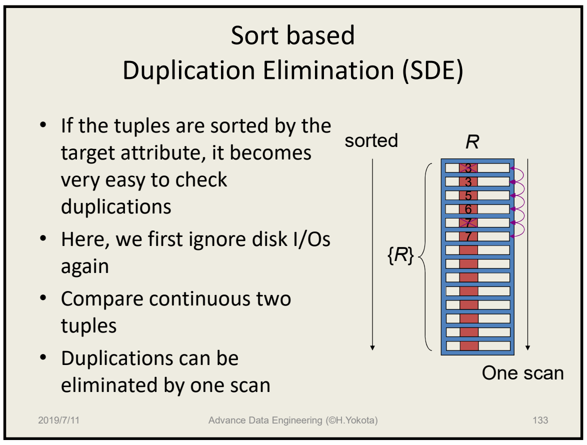
Pseudo Code for NLDE

```
for ($i$ = 1; $i \leq$ {$R$}; $i$++) {          Outer Loop
  $f$ := true;
  get $i$-th tuple $t_x$ and attribute value $v_x$ in $t_x$;
  for ($j$ = $i$ +1: $j \leq$ {$R$}; $j$++) {   Inner Loop
    get $j$-th tuple $t_y$ and attribute value $v_y$ in $t_y$;
    if $v_x$ == $v_y$ then $f$ := false;
  }
  if $f$ then output $t_x$
}
```

- This is a very straightforward algorithm and you can apply many optimizations on it

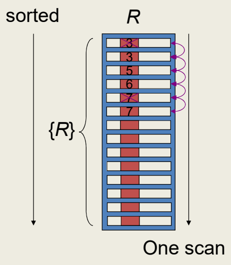
Costs for NLDE with Disk I/Os

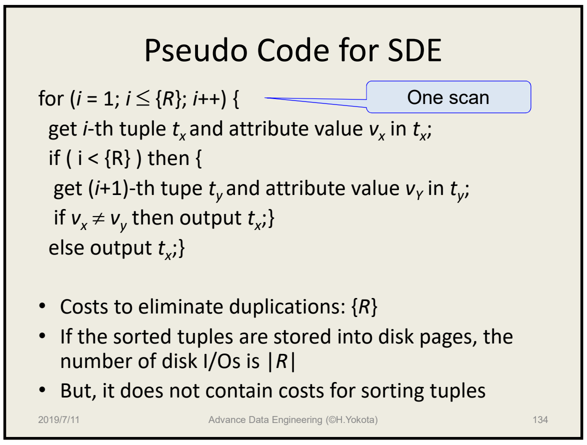
- The costs depend on the size of buffer
- If the buffer size is enough to keep all disk pages into the memory:
  - Disk I/Os: |R|
- If the buffer size is just the same as a disk page:
  - We have to apply the nested loop algorithm for disk I/Os
  - Disk I/Os : (|R| x (|R| + 1)) / 2



Sort based Duplication Elimination (SDE)

- If the tuples are sorted by the target attribute, it becomes very easy to check duplications
- Here, we first ignore disk I/Os again
- Compare continuous two tuples
- Duplications can be eliminated by one scan

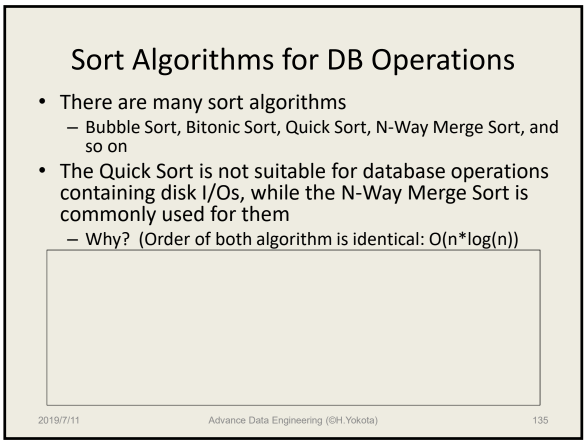




Pseudo Code for SDE

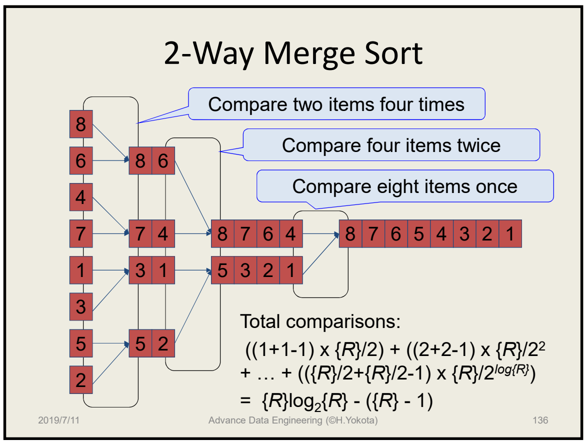
```
for ($i$ = 1; $i \leq \{R\}$; $i$++) {          One scan
  get $i$-th tuple $t_x$ and attribute value $v_x$ in $t_x$;
  if ( i < {R} ) then {
   get ($i$+1)-th tupe $t_y$ and attribute value $v_Y$ in $t_y$;
   if $v_x \neq v_y$ then output $t_x$;}
  else output $t_x$;}
```

- Costs to eliminate duplications: $\{R\}$
- If the sorted tuples are stored into disk pages, the number of disk I/Os is $|R|$
- But, it does not contain costs for sorting tuples



Sort Algorithms for DB Operations

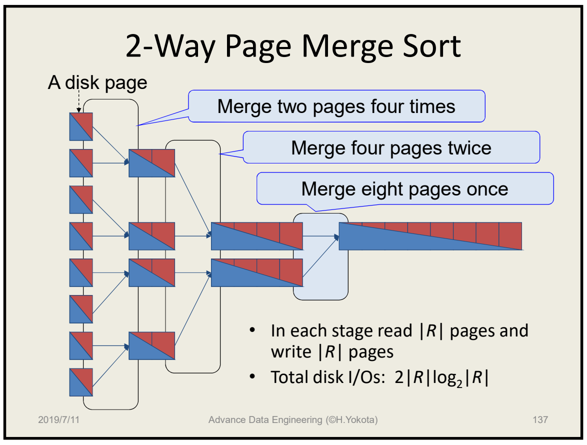
- There are many sort algorithms
  - Bubble Sort, Bitonic Sort, Quick Sort, N-Way Merge Sort, and so on
- The Quick Sort is not suitable for database operations containing disk I/Os, while the N-Way Merge Sort is commonly used for them
  - Why? (Order of both algorithm is identical: O(n*log(n))



2-Way Merge Sort

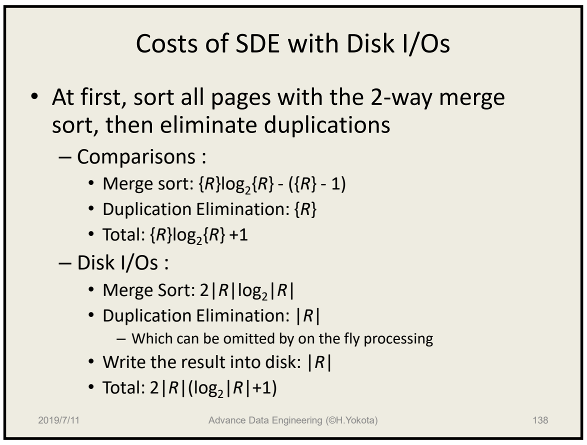
Total comparisons:

$((1+1-1) \times \{R\}/2) + ((2+2-1) \times \{R\}/2^2$
$+ \ldots + ((\{R\}/2+\{R\}/2-1) \times \{R\}/2^{\log\{R\}})$
$= \{R\}\log_2\{R\} - (\{R\} - 1)$



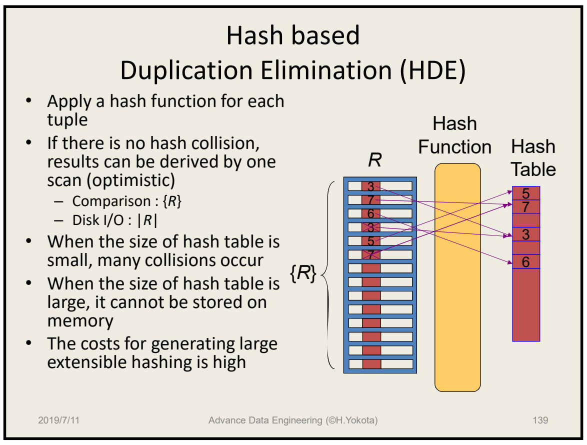
2-Way Page Merge Sort

- In each stage read $|R|$ pages and write $|R|$ pages
- Total disk I/Os: $2|R|\log_2|R|$



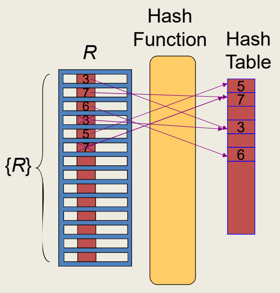
Costs of SDE with Disk I/Os

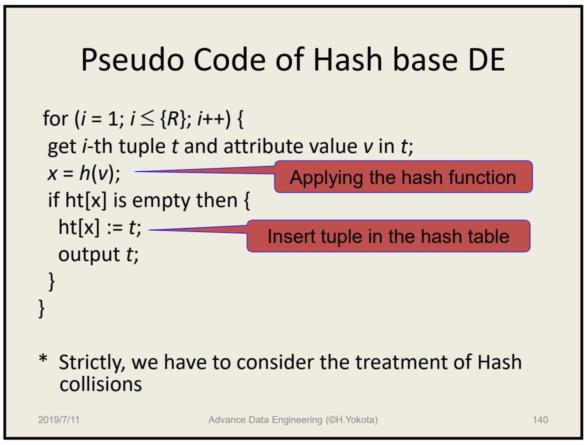
- At first, sort all pages with the 2-way merge sort, then eliminate duplications
  - Comparisons :
    - Merge sort: $\{R\}\log_2\{R\} - (\{R\} - 1)$
    - Duplication Elimination: $\{R\}$
    - Total: $\{R\}\log_2\{R\} + 1$
  - Disk I/Os :
    - Merge Sort: $2|R|\log_2|R|$
    - Duplication Elimination: $|R|$
      - Which can be omitted by on the fly processing
    - Write the result into disk: $|R|$
    - Total: $2|R|(\log_2|R|+1)$



Hash based Duplication Elimination (HDE)

- Apply a hash function for each tuple
- If there is no hash collision, results can be derived by one scan (optimistic)
  - Comparison : {*R*}
  - Disk I/O : |*R*|
- When the size of hash table is small, many collisions occur
- When the size of hash table is large, it cannot be stored on memory
- The costs for generating large extensible hashing is high

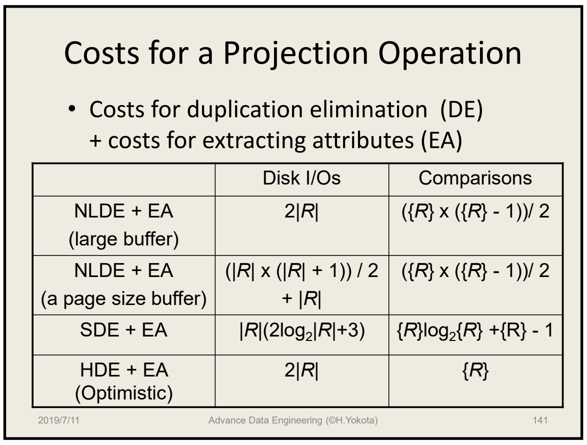




Pseudo Code of Hash base DE

```
for (i = 1; i ≤ {R}; i++) {
 get i-th tuple t and attribute value v in t;
 x = h(v);        Applying the hash function
 if ht[x] is empty then {
   ht[x] := t;        Insert tuple in the hash table
   output t;
 }
}
```

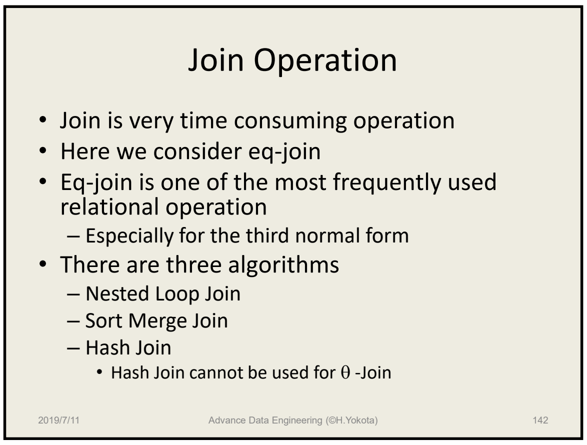
* Strictly, we have to consider the treatment of Hash collisions



Costs for a Projection Operation

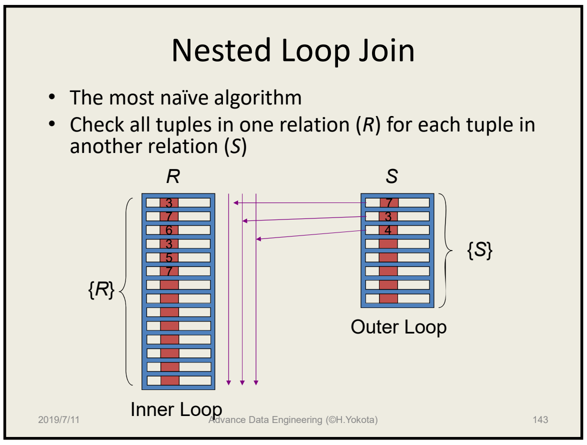
- Costs for duplication elimination (DE) + costs for extracting attributes (EA)

	Disk I/Os	Comparisons
NLDE + EA (large buffer)	2|*R*|	({*R*} x ({*R*} - 1))/ 2
NLDE + EA (a page size buffer)	(|*R*| x (|*R*| + 1)) / 2 + |*R*|	({*R*} x ({*R*} - 1))/ 2
SDE + EA	|*R*|(2$\log_2$|*R*|+3)	{*R*}$\log_2${*R*} +{R} - 1
HDE + EA (Optimistic)	2|*R*|	{*R*}



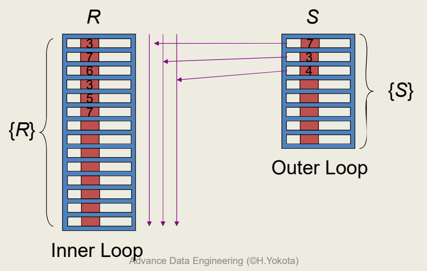
Join Operation

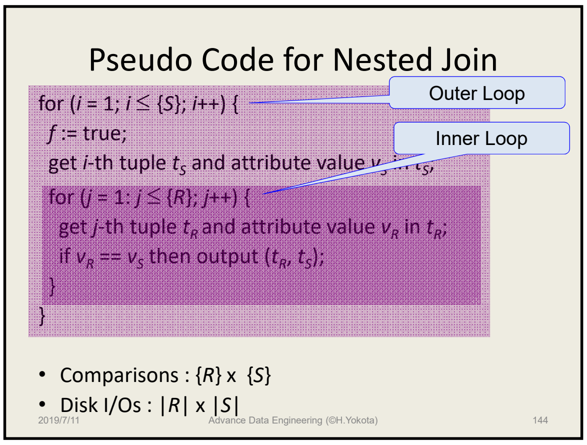
- Join is very time consuming operation
- Here we consider eq-join
- Eq-join is one of the most frequently used relational operation
  – Especially for the third normal form
- There are three algorithms
  – Nested Loop Join
  – Sort Merge Join
  – Hash Join
    - Hash Join cannot be used for θ -Join



Nested Loop Join

- The most naïve algorithm
- Check all tuples in one relation (R) for each tuple in another relation (S)



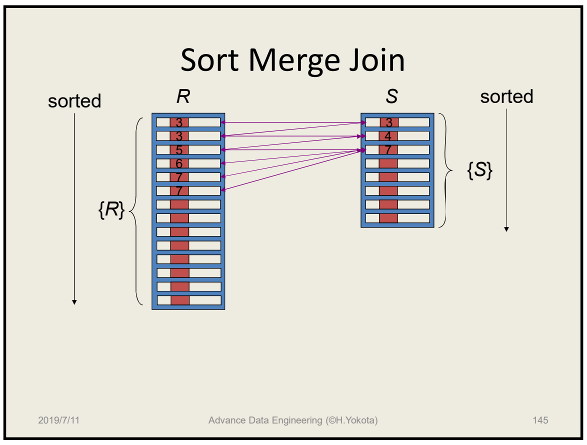


Pseudo Code for Nested Join

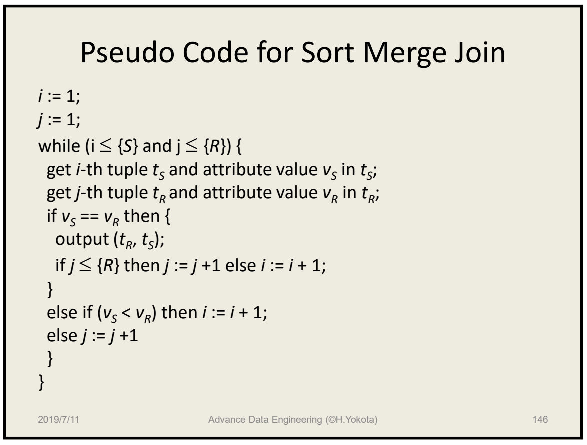
```
for ($i$ = 1; $i \leq \{S\}$; $i$++) {
  $f$ := true;
  get $i$-th tuple $t_S$ and attribute value $v_S$ in $t_S$;
  for ($j$ = 1: $j \leq \{R\}$; $j$++) {
    get $j$-th tuple $t_R$ and attribute value $v_R$ in $t_R$;
    if $v_R$ == $v_S$ then output ($t_R$, $t_S$);
  }
}
```

Outer Loop
Inner Loop

- Comparisons : $\{R\}$ x $\{S\}$
- Disk I/Os : $|R|$ x $|S|$

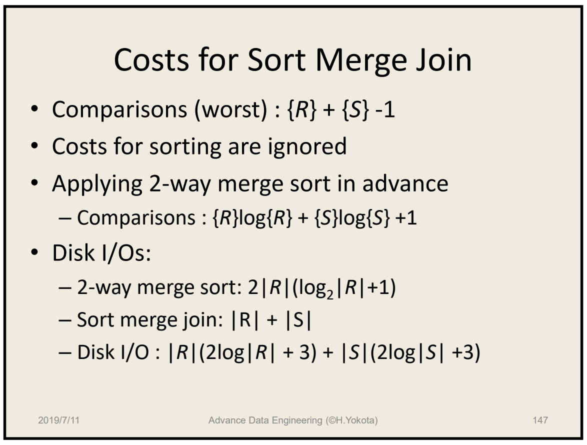


Sort Merge Join



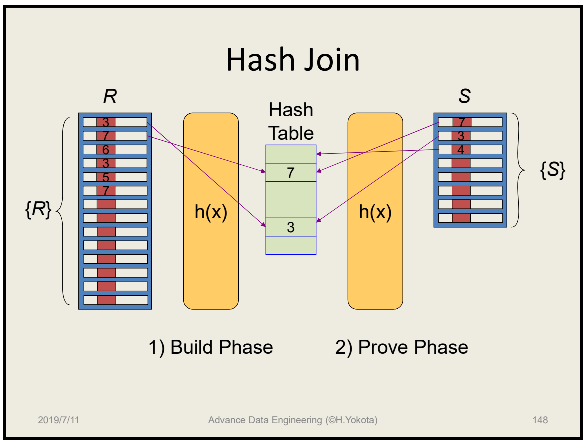
Pseudo Code for Sort Merge Join

```
$i$ := 1;
$j$ := 1;
while ($i \leq \{S\}$ and $j \leq \{R\}$) {
  get $i$-th tuple $t_S$ and attribute value $v_S$ in $t_S$;
  get $j$-th tuple $t_R$ and attribute value $v_R$ in $t_R$;
  if $v_S == v_R$ then {
    output ($t_R$, $t_S$);
    if $j \leq \{R\}$ then $j := j + 1$ else $i := i + 1$;
  }
  else if ($v_S < v_R$) then $i := i + 1$;
  else $j := j + 1$
  }
}
```



Costs for Sort Merge Join

- Comparisons (worst) : $\{R\} + \{S\} - 1$
- Costs for sorting are ignored
- Applying 2-way merge sort in advance
  - Comparisons : $\{R\}\log\{R\} + \{S\}\log\{S\} + 1$
- Disk I/Os:
  - 2-way merge sort: $2|R|(\log_2|R|+1)$
  - Sort merge join: $|R| + |S|$
  - Disk I/O : $|R|(2\log|R| + 3) + |S|(2\log|S| + 3)$



Hash Join

2019/7/11 Advance Data Engineering (©H.Yokota) 148