T07 Prashansa Singh

INFO20003 Tutorial 7

Starting ~ 2.20 pm

Today's tutorial

- Selection
  - effect of index
  - matching index
- Cost Estimation of Joins
- Exercises
  - group work

Group work: Q1 and Q2 (10 mins)

Exercises:

1. **Question about the effect of index on selection:**

Consider a relation R (a,b,c,d,e) containing 5,000,000 records, where each data page of the relation holds 10 records. R is organized as a sorted file with secondary indexes. Assume that R.a is a candidate key for R, with values lying in the range 0 to 4,999,999, and that R is stored in R.a order. For each of the following relational algebra queries, state which of the following three approaches is most likely to be the cheapest:

1 • Access the sorted file of R directly.
2 • Use a B+ tree index on attribute R.a.
3 • Use a hash index on attribute R.a.

Queries:

a. $\sigma_{a < 50000}$ (R) → 1
b. $\sigma_{a = 50000}$ (R) → 3
c. $\sigma_{a > 50000 \wedge a < 50010}$ (R) → 2

2. **Matching index**

Consider the following schema for the Sailors relation:

Sailors (sid INT, sname VARCHAR(50), rating INT, age DOUBLE)

For each of the following indexes, list whether the index matches the given $\underbrace{\text{selection conditions}}_{}$ σ_c and briefly explain why.

- A B+ tree index on the search key (Sailors.sid)
  a. $\sigma_{Sailors.sid < 50,000}$ (Sailors) ✓
  b. $\sigma_{Sailors.sid = 50,000}$ (Sailors) ✓
- A hash index on the search key (Sailors.sid)
  c. $\sigma_{Sailors.sid < 50,000}$ (Sailors) ✗
  d. $\sigma_{Sailors.sid = 50,000}$ (Sailors) ✓
- A B+ tree index on the search key (Sailors.rating, Sailors.age)
  e. $\sigma_{Sailors.rating < 8 \wedge Sailors.age = 21}$ (Sailors) ✓
  f. $\sigma_{Sailors.rating = 8}$ (Sailors) ✓
  g. $\sigma_{Sailors.age = 21}$ (Sailors) ✗

2. **Matching index**

Consider the following schema for the Sailors relation:

Sailors (sid INT, sname VARCHAR(50), rating INT, age DOUBLE)

For each of the following indexes, list whether the index matches the given selection conditions and briefly explain why.

- A B+ tree index on the search key (Sailors.sid)
  a. $\sigma_{Sailors.sid < 50,000}$ (Sailors)
  b. $\sigma_{Sailors.sid = 50,000}$ (Sailors)
- A hash index on the search key (Sailors.sid)
  c. $\sigma_{Sailors.sid < 50,000}$ (Sailors)
  d. $\sigma_{Sailors.sid = 50,000}$ (Sailors)
- A B+ tree index on the search key (Sailors.rating, Sailors.age)
  e. $\sigma_{Sailors.rating < 8 \wedge Sailors.age = 21}$ (Sailors)
  f. $\sigma_{Sailors.rating = 8}$ (Sailors)
  g. $\sigma_{Sailors.age = 21}$ (Sailors)

How is it sorted?
What does the data file and index file look like?
What are we trying to get in our results?

Exercises:

1. **Question about the effect of index on selection:**

Consider a relation R (a,b,c,d,e) containing 5,000,000 records, where each data page of the relation holds 10 records. R is organized as a sorted file with secondary indexes. Assume that R.a is a candidate key for R, with values lying in the range 0 to 4,999,999, and that R is stored in R.a order. For each of the following relational algebra queries, state which of the following three approaches is most likely to be the cheapest:

- Access the sorted file of R directly.
- Use a B+ tree index on attribute R.a.
- Use a hash index on attribute R.a.

Queries:

a. $\sigma_{a < 50000}$ (R)
b. $\sigma_{a = 50000}$ (R)
c. $\sigma_{a > 50000 \wedge a < 50010}$ (R)

a. $\sigma_{a < 50000}$ (R)

- Access the sorted file directly since it is sorted on attribute R.a in ascending order
  - and we want to know the entire record (all attributes) since we are doing selection
- Start from the beginning of the file, outputing the entire record and continue until you reach a record with a >= 50
- Stop there since everything after won't satisfy the condition

3. Indexing Cost

a. B+-tree index
   i. Just a single tuple (selection over a primary key)
      Cost = Height(I) + 1
   ii. Clustered index (multiple tuples)
      Cost = (NPages(I)+NPages(R)) * Π RF$_i$
   iii. Unclustered (multiple tuples)
      Cost = (NPages(I)+NTuples(R)) * Π RF$_i$

b. Hash Index
   i. Just a single tuple (selection over a primary key)
      Cost = 1.2 + 1 = 2.2
   ii. Clustered index (multiple tuples)
      Cost = (NPages (R)) * Π RF$_i$ * 2.2
   iii. Unclustered index (multiple tuples)
      Cost = (NTuples(R)) * Π RF$_i$ * 2.2

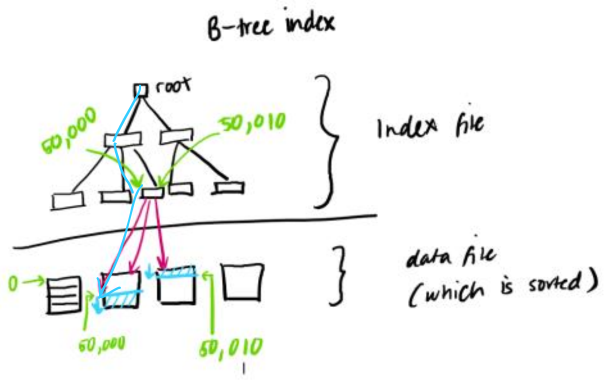
There is a cost of accessing the B-tree Index
- depends on Npages of Index

b. $\sigma_{a = 50000}$ (R)

- This is an equality condition. So hash index will be the cheapest option.

c. $\sigma_{a > 50000 \wedge a < 50010}$ (R)

- This is a range query which does not correspond to records at the beginning of the sorted file. So B+ tree is best option
- Cheaper to do B+ tree than hash index



2. **Matching index**

Consider the following schema for the Sailors relation:

Sailors (sid INT, sname VARCHAR(50), rating INT, age DOUBLE)

For each of the following indexes, list whether the index matches the given selection conditions and briefly explain why.

- A B+ tree index on the search key (Sailors.sid)
  - a. $\sigma_{\text{Sailors.sid} < 50{,}000}(\text{Sailors})$
  - b. $\sigma_{\text{Sailors.sid} = 50{,}000}(\text{Sailors})$
- A hash index on the search key (Sailors.sid)
  - c. $\sigma_{\text{Sailors.sid} < 50{,}000}(\text{Sailors})$
  - d. $\sigma_{\text{Sailors.sid} = 50{,}000}(\text{Sailors})$
- A B+ tree index on the search key (Sailors.rating, Sailors.age)
  - e. $\sigma_{\text{Sailors.rating} < 8 \wedge \text{Sailors.age} = 21}(\text{Sailors})$
  - f. $\sigma_{\text{Sailors.rating} = 8}(\text{Sailors})$
  - g. $\sigma_{\text{Sailors.age} = 21}(\text{Sailors})$

- In this question, just need to determine whether this index **CAN** be used. Don't need to worry about whether this index is the cheapest/best option

  ○ The index matches if the selection condition **matches** the index

  ○ The index matches and can be used if:

  ▪ the type of selection conditions **match** the type index
    □ e.g. your selection condition can be equality, inequality conditions etc
    □ e.g. your index can be B-tree, Hash etc

  ▪ AND the attributes in your selection condition are a **prefix** of the search key of the index
    □ e.g. if your index is on <a, b>, this matches selection conditions on **(a, b) and (a)** but NOT (b)
      ◆ **Why not?**
        ◇ It is not primarily sorted on b. Need to check every tuple to check condition on b.

      ◆ For selection conditions on (a, b) - **order doesn't matter** when thinking about whether index can be used.
        ◇ We just need both attributes a and b to be in the condition

ok

▸ e. $\sigma_{\text{Sailors.rating} < 8 \wedge \text{Sailors.age} = 21}(\text{Sailors})$

Terminology
- **"predicates"** = selection conditions
- **"primary conjuncts"** = matching predicates
  □ matching predicates = are the selection conditions that match the index

  ▪ e.g. primary conjuncts of index on <a,b> are selection conditions of the form (a, b) and (a)
  ▪ Will see more examples below

- A B+ tree index on the search key (Sailors.sid)
  - a. $\sigma_{\text{Sailors.sid} < 50{,}000}(\text{Sailors})$
  - b. $\sigma_{\text{Sailors.sid} = 50{,}000}(\text{Sailors})$

- a)
  ▪ Yes, the index matches
    □ as B-tree indexes can be used on range queries
    □ And the attributes (sid) in selection condition are **prefix** of search key of index <sid>

  ▪ Primary conjuncts are
    □ **Sailors.sid < 50,000**
    □ (only 1 primary conjunct as index is only built on a search key with 1 attribute)

- b)
  ▪ Yes, the index matches
    □ as B-tree indexes can be used on equality queries
    □ And the attributes (sid) in selection condition are prefix of search key of index <sid>

  ▪ **What are the primary conjuncts (what are the matching selection conditions)?**
    □ Sailors.sid = 50,000.

- A hash index on the search key (Sailors.sid)
  - c. $\sigma_{Sailors.sid < 50,000}$(Sailors)
  - d. $\sigma_{Sailors.sid = 50,000}$(Sailors)

- c)
  - ▪ The index doesn't match
    - □ as Hash indexes cannot be used for range queries

- d)
  - ▪ Yes, the index matches
    - □ as Hash indexes can be used on equality queries
    - □ And the attributes (sid) in selection condition are prefix of search key of index <sid>
  - ▪ What are the primary conjuncts?
    - □ Sailors.sid = 50,000.

- A B+ tree index on the search key (Sailors.rating, Sailors.age)
  - e. $\sigma_{Sailors.rating < 8 \wedge Sailors.age = 21}$(Sailors)
  - f. $\sigma_{Sailors.rating = 8}$(Sailors)
  - g. $\sigma_{Sailors.age = 21}$(Sailors)

- e)
  - ▪ Yes, the index matches
    - □ as B-tree indexes can be used on range and equality queries
    - □ And the attributes (rating, age) in selection condition are **prefix** of search key of index **<rating, age>**
  - ▪ What are all the primary conjuncts (matching predicates which are part of the prefix)?
    - □ There will be 2 primary conjuncts as index is built on a search key of 2 attributes <rating, age>. This has primary conjucts which relate to conditions of the form **(rating)** and **(rating, age)**.
    - □ What are the primary conjuncts?
      - □ Sailors.rating < 8.
      - □ Sailors.rating < 8 ∧ Sailors.age = 21.

- A B+ tree index on the search key (Sailors.rating, Sailors.age)
  - e. $\sigma_{Sailors.rating < 8 \wedge Sailors.age = 21}$(Sailors)
  - f. $\sigma_{Sailors.rating = 8}$(Sailors)
  - g. $\sigma_{Sailors.age = 21}$(Sailors)

- f)
  - ○ Yes, the index matches
    - ▪ as B-tree indexes can be used on equality queries
    - ▪ And the attributes (rating) in selection condition are **prefix** of search key of index <rating, age>
  - ○ What are the primary conjuncts?
    - □ Sailors.rating = 8.

- g)
  - ○ No, index does not match
    - ▪ Although B-tree indexes can be used on equality queries
    - ○ The attributes (age) in selection condition are **not a prefix** of search key of index <rating, age>
      - □ Possible prefix conditions are (rating) and (rating, age)
  - ○ The index on <rating, age> is primarily sorted on rating, so we need to search the entire relation to find the sailors with age = 21

<u>Group work (15-20 mins)</u>

3. Question about the cost analysis of different joins:

5. Joins (between relations R and S, R = outer, S = inner) Cost

Group work (15-20 mins)

3. Question about the cost analysis of different joins:

Consider the join R $\bowtie_{R.a = S.b}$ S, given the following information about the relations to be joined:

- Relation R contains 10,000 tuples and has 10 tuples/page. 1000
- Relation S contains 2,000 tuples and also has 10 tuples/page. 200
- Attribute b of relation S is the primary key for S.
- Both relations are stored as simple heap files.
- Neither relation has any indexes built on it.
- 52 buffer pages are available.

The cost metric is the number of page I/Os unless otherwise noted and the cost of writing out the result should be uniformly ignored.

~~201,000~~

a. What is the cost of joining R and S using the **page-oriented Simple Nested Loops** algorithm? What is the minimum number of buffer pages (in memory) required in order for this cost to remain unchanged?

200,200

b. What is the cost of joining R and S using the **Block Nested Loops** algorithm? What is the minimum number of buffer pages required in order for this cost to remain unchanged?

52

c. What is the cost of joining R and S using the **Sort-Merge Join** algorithm? Assume that the external merge sort process can be completed in 2 passes. → 6000 I/O

d. What is the cost of joining R and S using the **Hash Join** algorithm? 3600

~~5000~~

4200

e. What would the lowest possible I/O cost be for joining R and S using any join algorithm, and how much buffer space would be needed to achieve this cost? Explain briefly.

5. Joins (between relations R and S, R = outer, S = inner) Cost

a. NLJ

i. Tuple-oriented NLJ

Cost = NPages(R) + NTuples(R) * NPages(S)

ii. Page-oriented NLJ

200 + 1000 × 200

Cost = NPages(R) + NPages (R) * NPages(S)

iii. Block-oriented NJL (for block_size B)

Cost = NPages(R) + ceil(NPages (R)/(B-2)) * NPages(S)

b. Hash Join

Cost = 3*(NPages(R) + NPages(S))

c. Sort-Merge Join

Cost$_{SMJ}$ = NPages(R) + NPages(S) +

2* NPages(R)* num_passes(R) +

2* NPages(S)* num_passes(S)

On Canvas:

Query processing cost formulae

Legend

Symbol	Description
NKeys(Col)	The number of distinct values of column Col
High(Col)	The highest value of column Col
Low(Col)	The lowest value of column Col
NTuples(R)	The number of tuples of relation R
NPages(R)	The number of pages of relation R
NPages(I)	The number of pages of index I
Height(I)	The height of index I
Π RF$_i$	The product of all reduction factors
Π NTuples(R$_i$)	The product of the numbers of tuples of all relations taking part in a join
num_passes(R)	The number of passes for sorting relation R
PF	Projection factor (portion of all columns)

5. Joins (between relations R and S, R = outer, S = inner) Cost

a. NLJ

i. Tuple-oriented NLJ

Cost = NPages(R) + NTuples(R) * NPages(S)

ii. Page-oriented NLJ

Cost = NPages(R) + NPages (R) * NPages(S)

iii. Block-oriented NJL (for block_size B)

Cost = NPages(R) + ceil(NPages (R)/(B-2)) * NPages(S)

b. Hash Join

Cost = 3*(NPages(R) + NPages(S))

c. Sort-Merge Join

Cost$_{SMJ}$ = NPages(R) + NPages(S) +

2* NPages(R)* num_passes(R) +

2* NPages(S)* num_passes(S)

a. What is the cost of joining R and S using the **page-oriented Simple Nested Loops** algorithm? What is the minimum number of buffer pages (in memory) required in order for this cost to remain unchanged?

Working out:

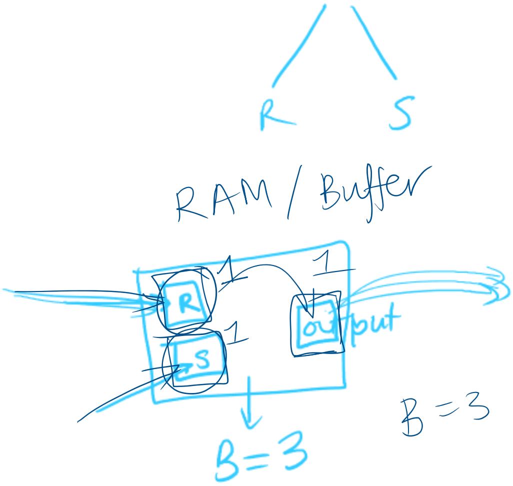
Let
- M be number of pages in R = 10000/10 = 1000
- N be number of pages in S = 2000/10 = 200
- B be number of buffer pages

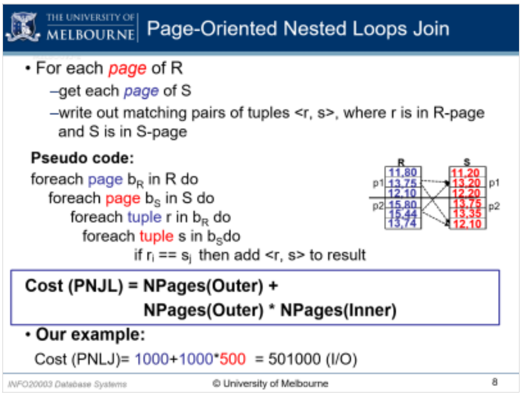
The basic idea of page-oriented nested loops join is to do a page by page scan of the outer relation and for each outer page, do a page-by-page scan of the inner relation.
The cost of joining R and S can be **minimised** by selecting the **smaller relation as the outer relation**. We will select S as the outer relation and compute cost as follows:

Total cost
= (# of pages in **outer**) + (# of pages in **outer** × # of pages in inner)
= N+ (N× M)
= 200 + (200× 1000) = 200,200 I/O

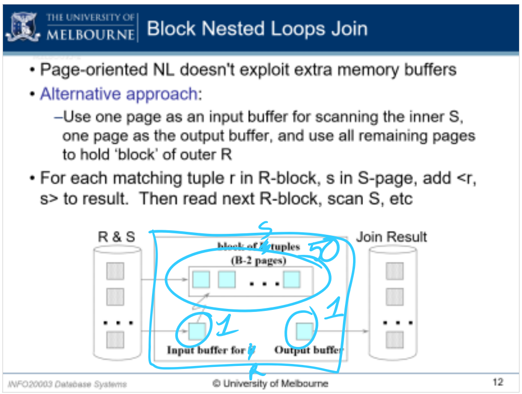
= 1000 + 200*1000 = 201000

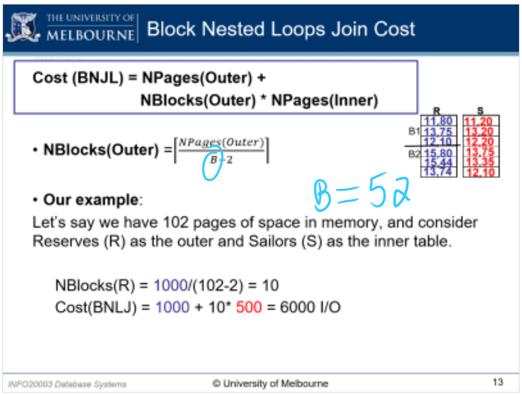
In this algorithm we don't use multiple buffers at a time,so the minimum requirement is one input buffer (for each of the two relations) to page through the relation and one output buffer to store the output. Hence in total 2 input +1 output = 3 buffer pages are required





b. What is the cost of joining R and S using the **Block Nested Loops** algorithm? What is the minimum number of buffer pages required in order for this cost to remain unchanged?





S

• **NBlocks(Outer)** $= \left\lceil \frac{NPages(Outer)}{B-2} \right\rceil = \left\lceil \frac{200}{52-2} \right\rceil = \left\lceil \frac{200}{50} \right\rceil = 4$ blocks

In block nested loops join,the **outer relation is read in blocks** (groups of pages that will fit into whatever buffer pages are available), and, for each block, do a page-by-page scan of the inner relation.
The outer relation is scanned once, and the inner relation is scanned once for each outer block. Assuming there are B buffers available,
# of blocks= ceil(# of pages in outer / (B–2))= ceil(200/50)=4

Total cost
= (# of pages in outer) + (# of blocks × # of pages in inner)
= 200 + (4 × 1000)
= 4200 I/O

Notice:
- For this example, choosing **S** (the relation with fewer pages) as the **outer** relation which we create blocks for **minimises the cost.**
- Try it the other way where R is the outer relation and compare the cost!

What if we have fewer buffers?

If we have fewer buffers available, the cost will increase as the # of blocks will increase. The minimum number of buffer pages is 52 for this cost.

• **NBlocks(Outer)** $= \left\lceil \frac{NPages(Outer)}{B-2} \right\rceil$

- Lower B means higher Nblocks

c. What is the cost of joining R and S using the **Sort-Merge Join** algorithm? Assume that the external merge sort process can be completed in 2 passes.

<u>**Note**</u>: You only need to sort tables if they are not sorted on the attributes you are joining on.

We will use external merge sort for this case.

The general formula for external merge sort is 2N× # of passes. (Where N = Number of

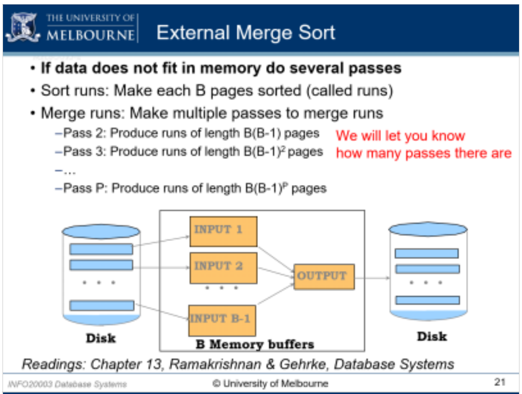
pages)
As we only require two passes to sort R and S, the cost analysis is as follows:

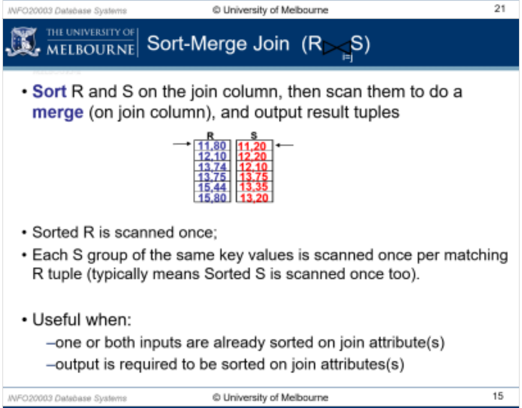
Cost of sorting R
=2× # of passes × # of pagesof R
= 2 × 2 × 1000
= 4000

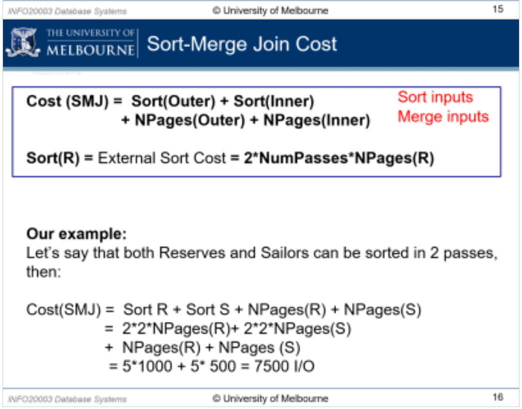
Cost of sorting S
= 2× 2× 200
= 800

Cost of merging R and S
= # of pages read of R + # of pages read of S
= 1000+200
= 1200

Total cost
= Cost of sorting R + Cost of sorting S + Cost of merging R and S
= 4000 + 800 + 1200
= 6000 I/O







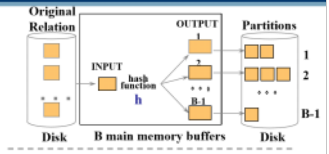
d. What is the cost of joining R and S using the **Hash Join** algorithm?

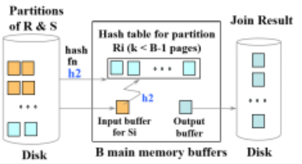
In hash join, each relation is partitioned and then the join is performed by "matching" elements from corresponding partitions.

Total cost
= 3(M+N)
= 3(1000+200)
= 3600 I/O

Hash-Join

- Partition both relations using hash function h: R tuples in partition I will only match S tuples in partition I
- Read in a partition of R, hash it using h2 (<> h!). Scan matching partition of S, probe hash table for matches





Hash-Join Cost

1. In partitioning phase, we read+write both relations
2. In matching phase, we read both relations

Cost (HJ) = 2 * NPages(Outer) + 2* NPages(Inner) Create partitions
+ NPages(Outer) + NPages(Inner) Match partitions

- **Our example:**

Cost(HJ) = 2*NPages(R) + 2*NPages(S) + NPages(R) + NPages(S)
= 3 * 1000 + 3* 500 = 4500 I/Os

e. What would the lowest possible I/O cost be for joining R and S using any join algorithm, and how much buffer space would be needed to achieve this cost? Explain briefly.

1000

In the very best case, how many times do we have to read the relations R and S?

200

1200

COST BNLJ = (# of pages in outer) + (# of blocks × # of pages in inner)

200 + 1 1000

= 1200

1200

R S

The optimal cost would be achieved if each relation was read only once.

We could do such a join by **storing the entire smaller relation in memory**, reading in the larger relation page by page and for each tuple in the larger relation we search the smaller relation (which exists entirely in memory) for matching tuples.

The buffer pool would have to hold the entire smaller relation, one page for reading in the larger relation and one page to serve as an output buffer.

Min Cost BLNJ = (# of pages in outer) + $(\underbrace{\text{\# of blocks}}_{1}$ × # of pages in inner)

- **NBlocks(Outer)** $= \left\lceil \frac{NPages(Outer)}{B-2} \right\rceil$

$= 1$

200

$\therefore B = 202$

The minimum number of buffer pages for this cost is min{M, N}+1+1 = 202.

Total cost = M+N = 1200

Block Nested Loops Join

- Page-oriented NL doesn't exploit extra memory buffers
- Alternative approach:
  - –Use one page as an input buffer for scanning the inner S, one page as the output buffer, and use all remaining pages to hold 'block' of outer R
- For each matching tuple r in R-block, s in S-page, add <r, s> to result. Then read next R-block, scan S, etc

$B = 200 + 1 + 1 = 202$