week12

Cost = read, not write

B(R) = # of block (pages) for R

T(R) = # of tuples in R

V(R,a) = # of distinct values of attribute a

- when a is key → V(R,a) = T(R)
- else → V(R,a) < T(R)

Join (R⋈S)

1) One-pass Algorithms

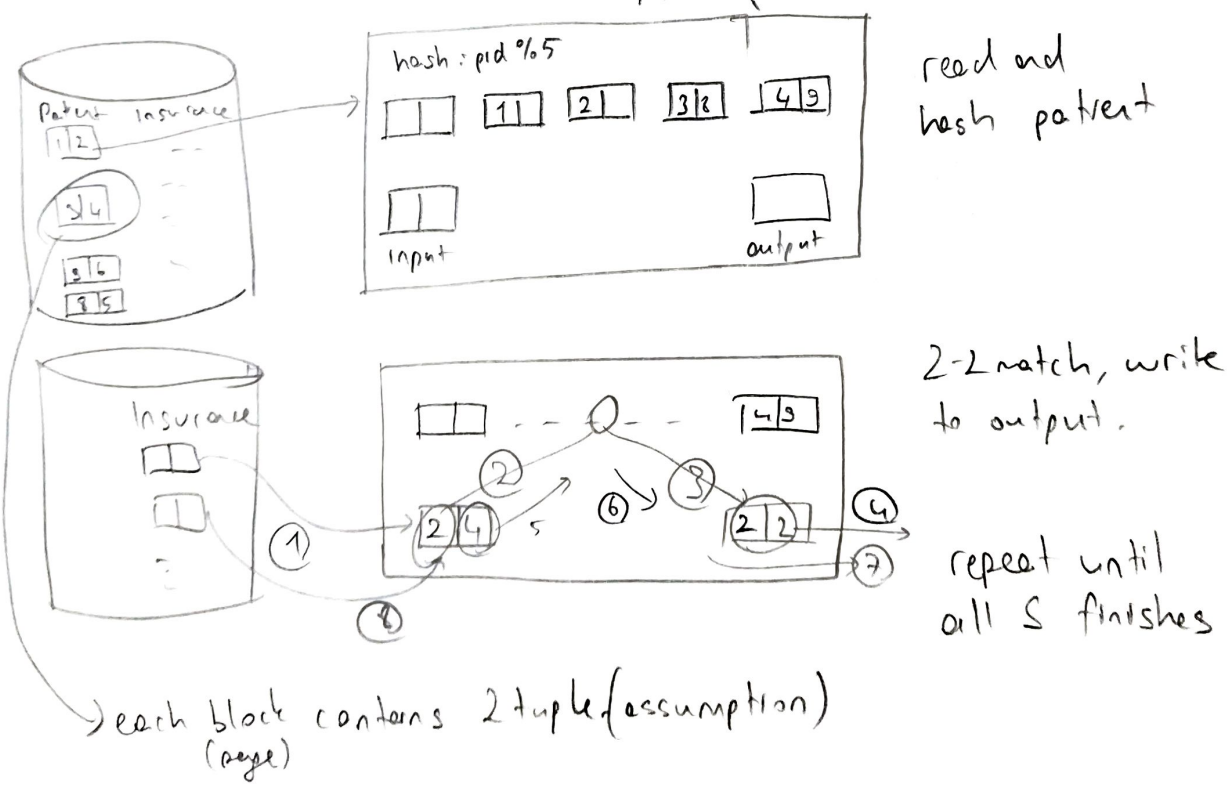
It fits in the memory

a) Hash Join

R⋈S

- scan R, build buckets in mem.
- scan S, join.

Cost B(R)+B(S).



read and hash patient

2-2 match, write to output.

repeat until all S finishes

Joined record, output buffer contains:

(pid, name, addr, provider, policy)

↓ only 1 pid (They have been joined on pid)

Cost
B(R)+B(S)

Algorithm R⋈S

```
Open(){
  H= newHashTable();
  R.open();
  x=R.GetNext();
  while(x!=null){
     H.insert(x);
     x=R.GetNext();
  }
  R.close();
 S.Open();
  buffer=[];
}
```

Pipelined.

```
GetNext(){
  while(buffer==[]){
     x= S.GetNext();
     if(x==null) return null;
     buffer=H.find(x);
  }
  z= buffer.first();
  buffer.reset;
  return z;
}
```

→ Since it is pipelined 1 element at a time.

---

b) Nested loop Join

for each tuple t_1 in R

  for each tuple t_2 in S

    if t_1 and t_2 join then output (t_1, t_2)

Cost
B(R) + T(R)·B(S)

read S # of tuples of R times

C) Page at a time

for each ~~page~~ of R → B(R)

  for each ~~page~~s of S → B(R)·B(S)

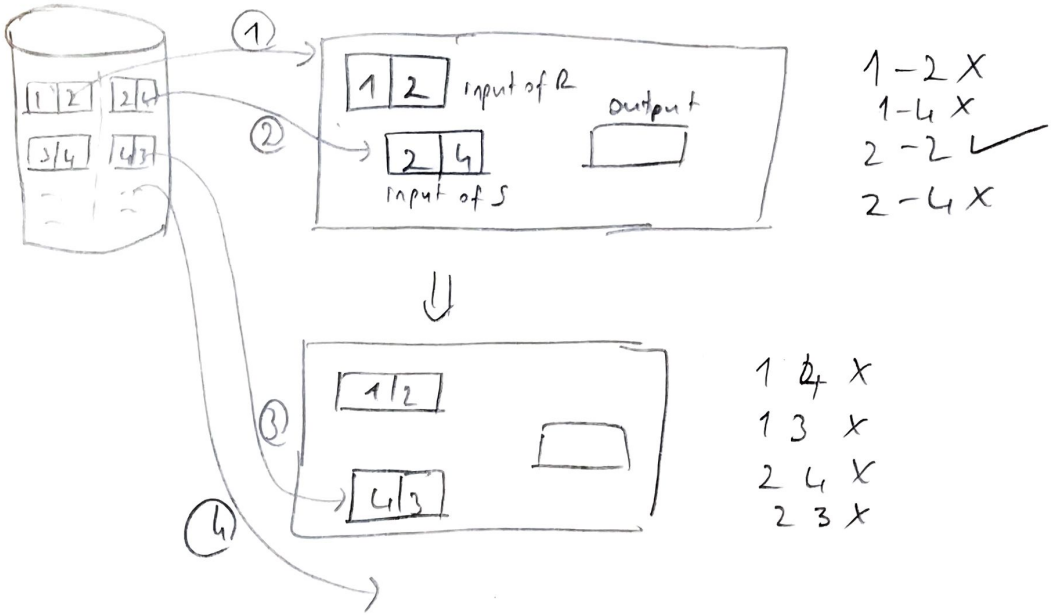
    ↳ S will be read ↳ # of pages of R times

    for all pairs of tuples t_1 in r, t_2 in s

      if t_1 and t_2 join, output (t_1, t_2)

Cost
B(R)+B(R)·B(S)

visualization of page at a time



1-2 X
1-4 X
2-2 ✓
2-4 X

1 4 X
1 3 X
2 4 X
2 3 X

d) Block Nested loop

Memory has M pages. Use 1 for input
1 for output.

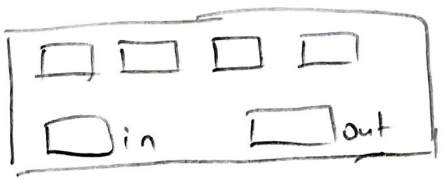
read (M-2) pages of R to memory.

for each group of (M-2) pages r in R: $\longrightarrow B(R)$
  for each page of tuples s in S: $\longrightarrow \frac{B(R)}{(M-2)} \cdot B(S)$
    for all pairs of tuples t_1 in r, t_2 in s
      output

Cost
$B(R) + \frac{B(R)}{(M-2)} \cdot B(S)$

ex:

M=6



if R is 4 pages.
I can fit R into mem

$$\underbrace{B(R)}_{4} + \frac{B(R)}{M-2} \to \frac{4}{4} \to 1 \cdot B(S)$$

I need to traverse B 2 times.

if R is 8:

$$\underbrace{B(R)}_{8} + \frac{B(R)}{M-2} \to \frac{8}{4} \to 2 \cdot B(S)$$

e) Sort-Merge

Scan R, sort in mem
Scan S, //
Merge R&S.

cost: $B(R)+B(S)$ ← if it fits mem

typically this is not one pass alg.

---

2) Index Based Algorithms

-Existing index on join attribute (inner)

Product $\bowtie_{id}$ Purchase
↓ outer ↓ inner

Selection on equality: $\sigma_{a=v}(R)$ ← Before join

Index based selection

Clustered index on a: $\frac{B(R)}{V(R,a)}$
(ignore I/O for index access)

Unclustered index on a: $\frac{T(R)}{V(R,a)}$

---

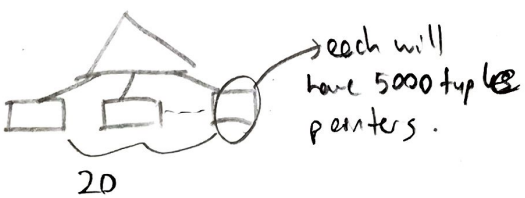
ex: $B(R)=2000$
$T(R)=100{,}000$
$V(R,a)=20$ → 20 distinct vals on a.

$\sigma_{a=v}(R)=?$

a) Table scan
$B(R)=2000$ I/O

b) Clustered index $\frac{B(R)}{V(R,a)}=100$ → 2000 page var. 20 farklı a var. ortalama $\frac{2000}{20}=100$ page'de bu $a=v$ eşitliği vardır. Index clustered olduğu için ilk olanı bul, sonra linear search.

c) Unclustered index $\frac{T(R)}{V(R,a)}=5000$ → indexten hangi page olduğunu bulduk one unclustered



in the worst case you need 5000 I/O.

Dont build unclustered indexes when V(R,a) is small.

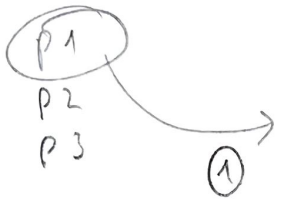
end of index based selection

a) Index Nested loop join

R ⋈ S → assume inner(S) has index on join attribute.

Iterate over R, foreach tuple, fetch corresponding tuples from S.

Product ⋈pid Purchase



c1 p1
c2 p1 → clustered index on pid
c1 p2
—
— ps

① Find matching in S.
② next in Product.

B(R) → read outer relation.

$B(R) + T(R) \cdot \frac{B(S)}{V(S,a)}$

↓ for each row of the R

→ diyelim 30 page var.
3 distinct a value'miz var.
Bu a value'si 10 tane page de'dir.
T(R).10 → here okey.

$\text{clustered index} = B(R) + T(R) \cdot \frac{B(S)}{V(S,a)}$

$\text{Unclustered index} = B(R) + T(R) \cdot \frac{T(S)}{V(S,a)}$

→ unclustered oldugu için bütün row lara bakabilmiz gerekebilir.

3) Two Pass Algorithms

Files are big, dont fit in the memory.

- sorting & hashing.

a) Sort & Merge

External Sorting

- Sort a file B → size with mem M.

- Sorting is two pass when $B < M^2$

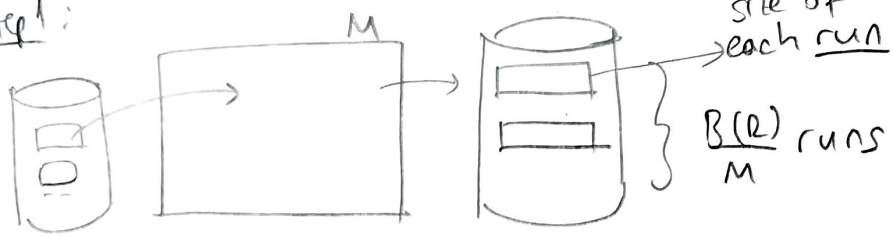
ex: Page size 32KB R can be large as 10^{12} pages

Mem 32GB

$= 10^6$ pages

$10^{12} \times 32 \times 10^3 = 32\,PB$

Step 1:

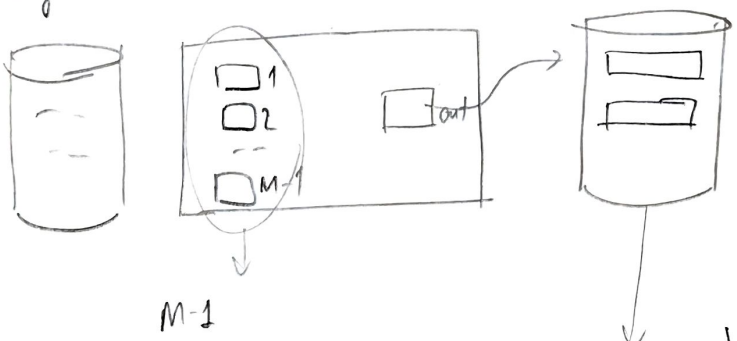


size of each run is M.

$\frac{B(R)}{M}$ runs

Step 2

merge M-1 runs to a new run



? $M(M-1) \cong M^2$?

Cost: Read + Write + Read (final write is not included) → $3B(R)$

Assumption $B(R) <= M^2$

Merge Join

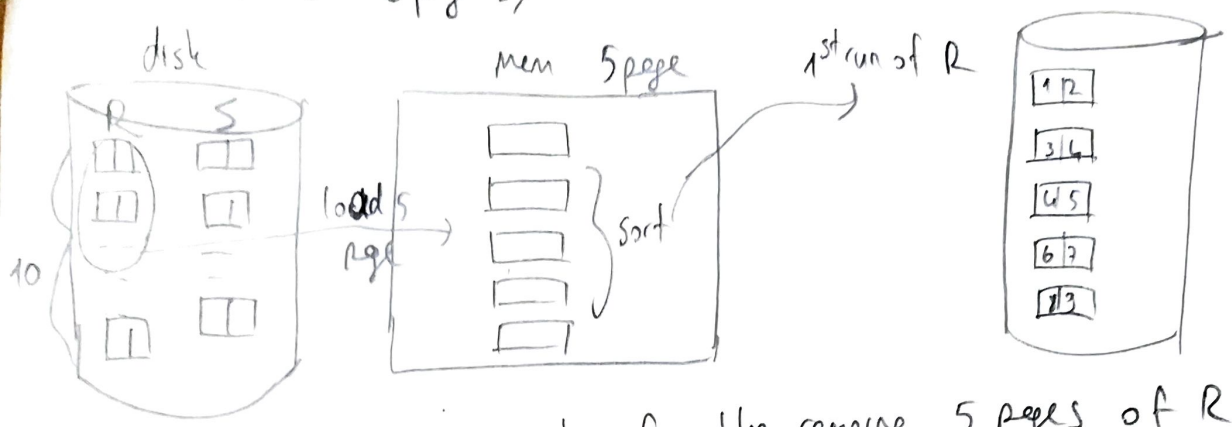
1a: Generate initial runs of R

1b: " " S

2: Merge & join → either → 1st merge then join

or → at the same time

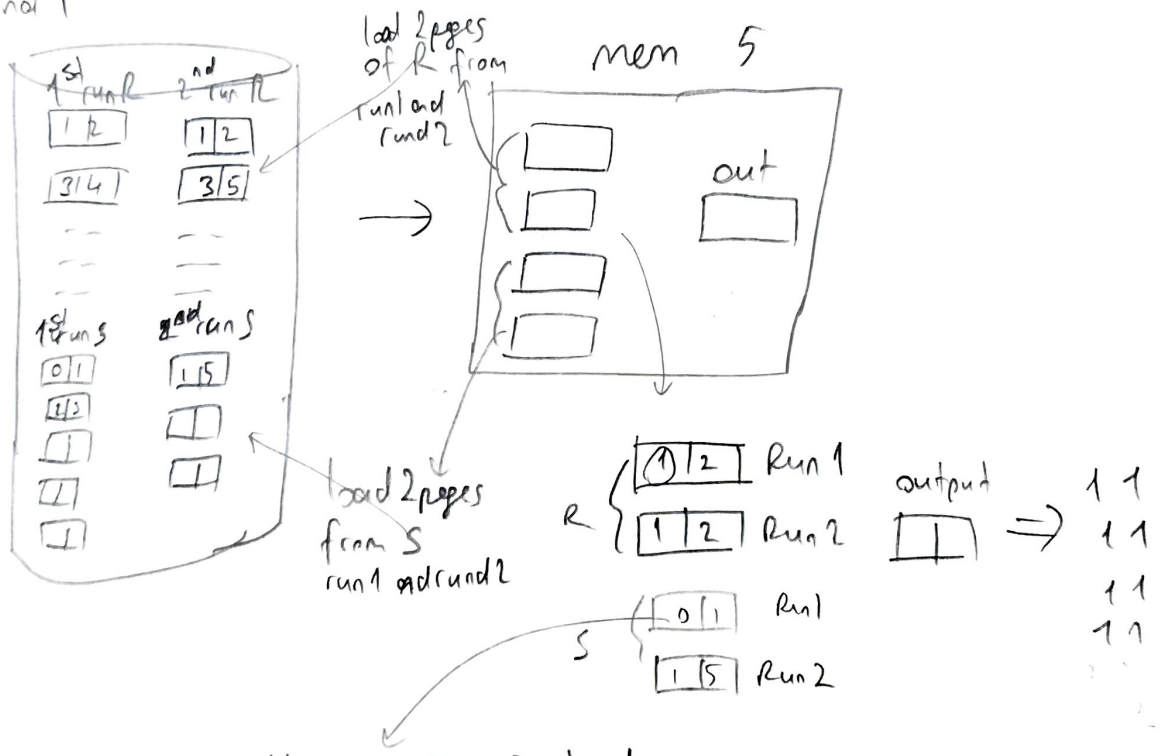
ex:
R has 10 pages, 2 tuples per page
S has 8 pages,



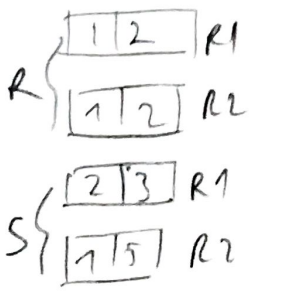
do this step for the remaing 5 pages of R.

do these steps for S too. $\Rightarrow$ $2B(R)+2B(S)$

Final



remove this page, it is finished



→ 2,2
2,2

Continue

In our example total cost = $3B(R)+3B(S)$

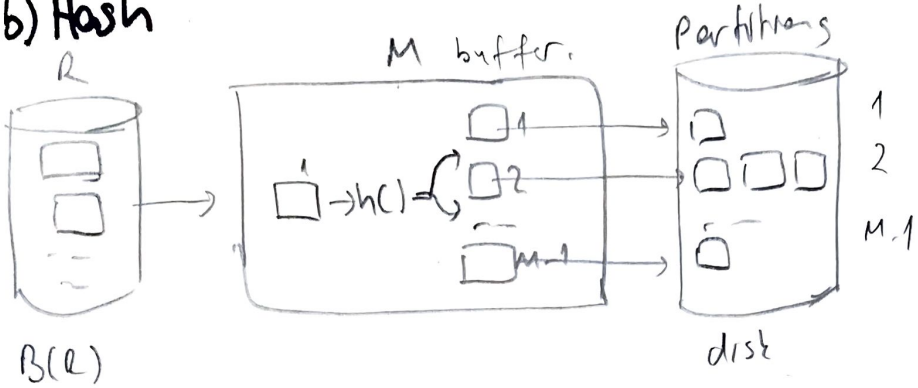
normal formula = $M1 = \frac{B(R)}{M}$ for R, in our case $\frac{10}{5} = 2$

$M2 = \frac{B(S)}{M}$ in our case $\frac{8}{5} = 2$

if $M1+M2 <= M$, it can be done
otherwise recursively (sort merge)

We can also do two pass based on hashing.

b) Hash



hash fonksiyonunun eşit böldüğünü düşünelim.

each bucket will have $\frac{B(R)}{M}$ size

if $\frac{B(R)}{M} <= M$ i.e $B(R) <= M^2$ → each bucket will fit in memory.

Step 1: Hash S into M-1 buckets
Send all buckets to disk

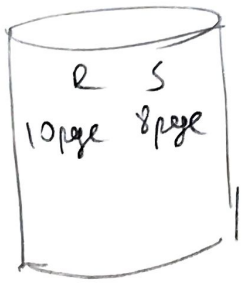
Step 2: R - -

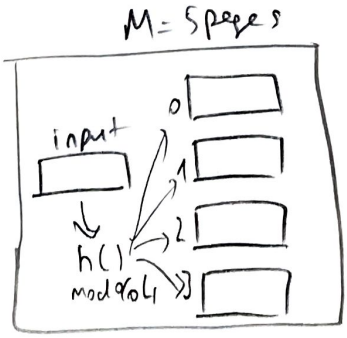
} Same hash function

Step 3: Join every pair of buckets } $h2 \neq h1$ (different hash)

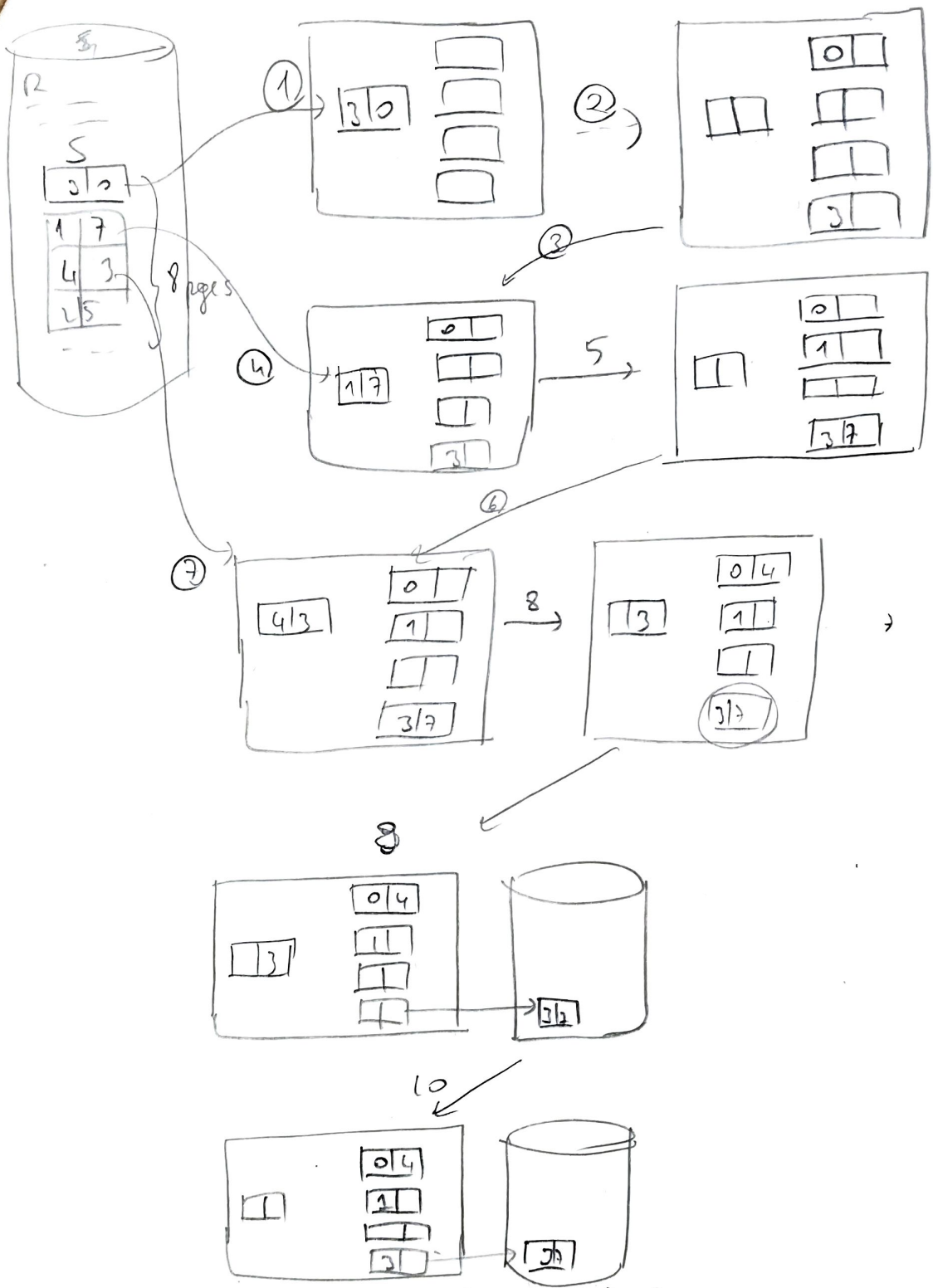
Cost $3B(R) + 3B(S)$, assumption $\min(B(R), B(S)) <= M^2$

example



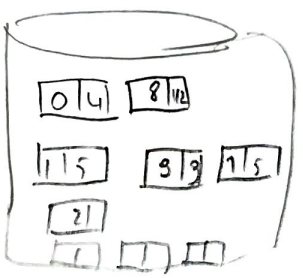


Read S. Step 1:



do this for all pages of S.

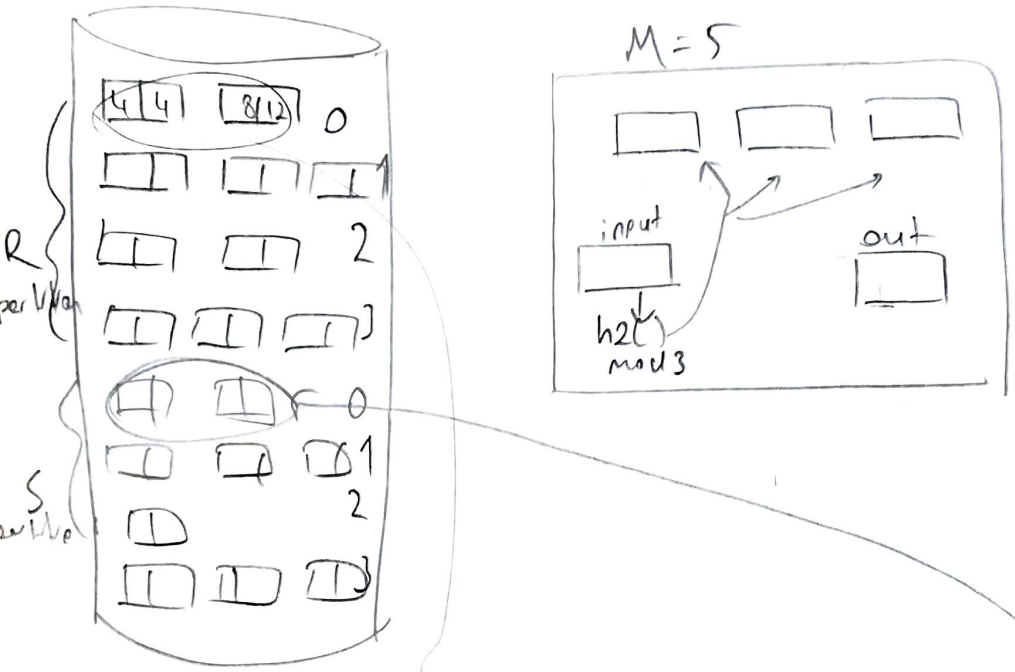
At the end, we will 4



They are sorted not but partitioned

Step 2: do the same step for R

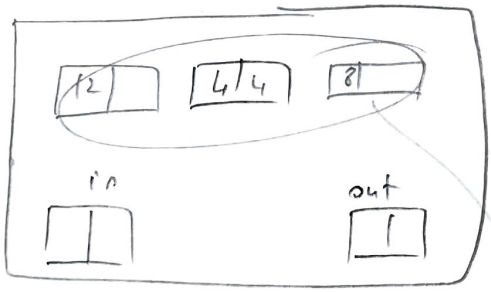
Step 3: Using dif. hash funct.



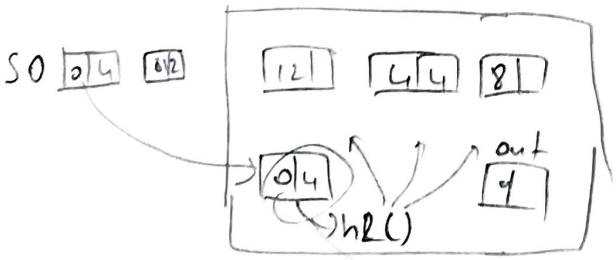
Cost

3B(R) + 3B(S)

① read 1st partition of R into M, h2 it.



② scan only matching partition of S.



Buraya R'deki hash leyerek koydugum icin, her gelen S'in partitionu lineer search yapmiyor! ✓

Daha hizli.

Hem gez

0 mod 3 → 0 | 0, 12 not match | take next 8 12

4 mod 3 → 1 | 1, 4 // | 8 mod 3 → 2, 8,8 ✓ match

1, 4 | 12 mod 3 → 0, 12,12 ✓ match

(10)