15.4.4) $B(R) = 1000$ $B(S) = 500$

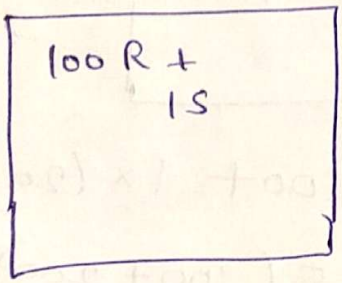
$M = 101$

a) Sorting cost → $4(B(R) + B(S)) = 4(1500)$

$= 6000$

Now there are 2 ways of doing nested loop join : To keep 100 blocks of S & 1 of R or 100 of S & 1 of R.

$M = 101$



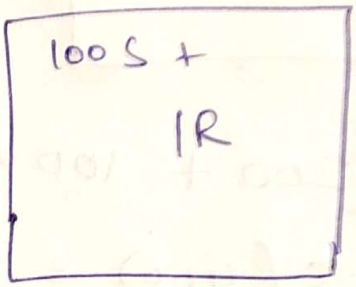
$2(500 + 5 \times 250)$

$= 2(1750)$

$= 3500$

Because we have to bring all of S, 5 times as 500 of R will have same value.

$M = 101$



$2(250 + 3 \times 500)$

$= 2(1750)$

$= 3500$

Same in both

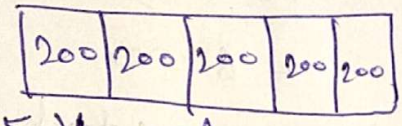
Total Cost → $6000 + 3500$

$= 9500$

b.) Sorting = $4(1000+500) = 6000$

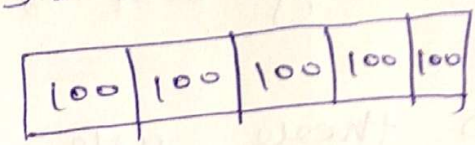
Joining ⇒

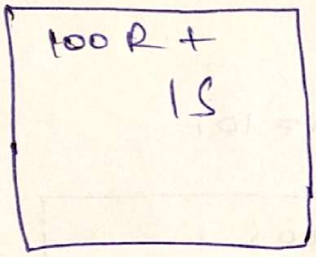
R = 1000

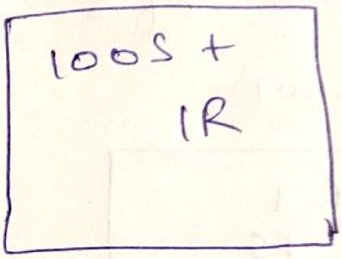


5 values each equally likely.

S = 500







$5(200 + 100 \times 2)$

$= 5(400) = 2000$

$5(100 + 1 \times (200))$

$= 5(100 + 200)$

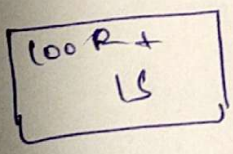
$= 1500$

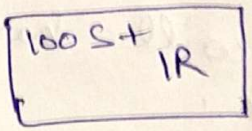
Total Cost ⇒ $6000 + 1500 = 7500$.

c.) Sorting = $4(1000 + 500) = 6000$

R = 100, ... 10 times.

S ⇒ 50 ... 10 times





⇒ $10(100 + 1 \times 50)$

⇒ $10(150) = 1500$

⇒ $5(100 + 200(1)) = 1500$

OR.

$10(50 + 50 \times 2) = 1500$

(Same)

⇒ Total Cost = $6000 + 1500$

$= 7500$

15.4.5.) On this algorithm we combine second stage sorting with joining.

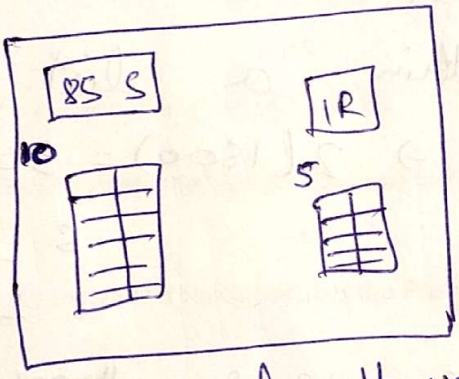
a) So sorting within a list takes 1 read & write ⇒ $2(1000 + 500) = 3000$

Now half values in each R & S are same.

R = | 500 | 500 |

S = | 250 | 250 |

General : There will be 10 sorted list of R & 5 of S as 100 is capacity. So, we have to make 10 list of R & 5 of S.



By general rule of thumb of nested loop we know that we keep lesser value (Blocks) relation in outer loop. So, S is in outer loop.

Joining ⇒ $2\left(250 + \left\lceil \frac{250}{85} \right\rceil \times 500\right)$ ⇒ $2(250 + 3(500))$

⇒ 3500

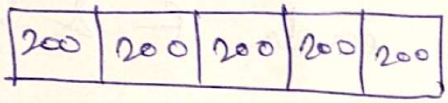
So, Total Cost = 3000 + 3500

= 6500

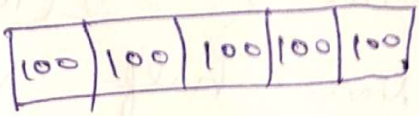
b.) Sorting within a list would take 1 read & write ⇒ 2(1000 + 500) = 3000

Now 5 values are same.

R.



S



Same 85 S + 1R & other four 1 block from each list.

Joining Cost ⇒

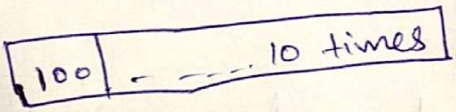
⇒ $5\left(100 + \left\lceil \frac{100}{85} \right\rceil 200\right)$

⇒ 5 (100 + 2(200)) ⇒ 2500

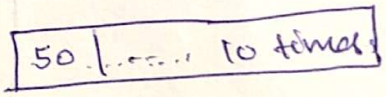
Total Cost = 3000 + 2500 = 5500

c.) Sorting within a list would take 1 read & write ⇒ 2(1500) = 3000

R



S



Same as 85 S + 1R & other four 1 block from each list.

Joining Cost ⇒

⇒ $10\left(50 + \left\lceil \frac{50}{85} \right\rceil \times 100\right)$

⇒ 10 (50 + 100) = 1500

Total Cost = 3000 + 1500

= 4500

15.5.5

a)

Given:

B(R) = 1000

B(S) = 500

M = 101

Also, its given: "to speed up the join, we want to use as few buckets as possible (assuming tuples distribute evenly between buckets)"

Basically, we want to:

1. Use as few buckets as possible
2. Utilize as much of main memory available.
3. During the joining step of hash-join algorithm, fit one complete bucket of a relation into the main memory.

From above statements, we can say that:

$$\frac{500}{k} <= 101$$

Therefore, $k >= 5$ (where k = number of buckets to be used).

Hence, the minimum number of buckets that can be used and satisfy 1,2,3 is k=5.

Now, lets calculate the cost:

R and S are initially stored in consecutive blocks. (as per the Professor, we can assume this)

Hashing Step:

Total Main Memory Blocks = 101

Reserved blocks: 5 (one for each bucket), 1(for calculating hash values)

Therefore, available blocks = 101 - (5+1) = 95 blocks.

Read R:

We will read R in chunks of 95 blocks.

Hence, the read time is: $(100 + 95/2) * ceil(1000/95) = 1622.5\ ms$

Write R:

When we write a block to disk, we can specify the address where we want to write it.

Hence, each block is going to be written to disk with a random I/O cost.
Thus, the writing time is:
$100.5 * 1000 = 100500\ ms$

Read S:

We will read S in chunks of 95 blocks.
Hence, the read time is: $(100 + 95/2) * ceil(500/95) = 885\ ms$

Write S:

Each block is going to be written to disk with a random I/O cost.
Thus, the writing time is:
$100.5 * 500 = 50250\ ms$

Joining Step:

During the hashing step, the write were such that all blocks of a bucket got stored consecutively onto disk. So, we can read chunks of blocks of the same bucket.
In this step, we will read into main memory a complete bucket of S, and then read the corresponding bucket of R, one block at a time. We'll repeat this for all buckets.
Hence, the joining time is:

Read S: $(100 + 100/2) * 5 = 750\ ms$

Read R: $(100.5 * 1000) = 100500\ ms$

Therefore, total I/O time is:

$$1622.5 + 100500 + 885 + 50250 + 750 + 100500 = 254507.5\ ms = approx\ 254.5s$$

b)

Given:

B(R) = 1000

B(S) = 500

M = 101

Also, its given: "to speed up the join, we want to use as few buckets as possible (assuming tuples distribute evenly between buckets)"

For hybrid hash join, to save some disk I/O's, we keep one COMPLETE bucket of one relation in main memory.

Yet again, the conditions here are similar to the one in the previous part:

We want to:

1. Use as few buckets as possible => make each bucket size as close to the main memory size as possible.
2. Utilize as much of main memory available.
3. During the hashing step of hash-join algorithm, one complete bucket of a relation and one block for rest of the buckets should fit into the main memory.

From the above conditions, we have:

$\frac{500}{k} <= 101 \quad => \quad k >= 5$

For $k = 5$, a bucket size would be $500/5 = 100$ blocks, and 4 other blocks would be needed for rest of the buckets during hashing step. So the total memory requirement would be $100 + 4 = 104$ blocks whereas we have only 101 blocks. So, we choose the next higher value of k i.e. **k = 6**.

Now, lets calculate the cost of hybrid hash join:

R and S are stored initially in consecutive blocks respectively (as per the Prof., we can assume this)

Hashing Step:

Total Main Memory Blocks = 101

Reserved blocks: ceil(500/6) = 84 (one complete bucket), 5 (one block for rest of the buckets), 1 (for calculating hash)

Therefore, available blocks = 101 - (84+5+1) = 11 blocks.

Read S:

We read S in chunks of 11 blocks.

Hence, the read time is:

$(100 + 11/2) * ceil(500/11) = 4853\ ms$

Write S:

We will write back $500 - ceil(500/6) = 500 - 84 = 416$ blocks into disk, as blocks of one complete bucket will keep on residing in main memory.

Furthermore, when we write a block to disk, we can specify the address where to write it. Hence, each block is going to be written to disk with a random I/O cost.

Thus, the writing time is:

$416 * 100.5 = 41808\ ms$

Read R:

We read R in chunks of 11 blocks:
Hence, the read time is:
$(100 + 11/2) * ceil(1000/11) = 9600.5\ ms$

Write R:

We will write back $1000 - ceil(1000/6) = 1000 - 167 = 833$ blocks into disk, as join output for the corresponding bucket of R, whose S bucket already resides in the main memory, can be done while calculating the hash for blocks of R i.e. we write back 5 of the 6 buckets of R.

Furthermore, when we write a block to disk, we can specify the address where to write it. Hence, each block is going to be written to disk with a random I/O cost.

Thus, the writing time is:

$833 * 100.5 = 83716.5\ ms$

Joining Step:

During the hashing step, the write were such that all blocks of a bucket got stored consecutively onto disk. So, we can read chunks of blocks of the same bucket.

In this step, we will read into main memory a complete bucket of S, and then read the blocks of the corresponding bucket of R (number of free blocks left after reading a complete bucket of S). We'll repeat this for all buckets.

Total Main Memory Blocks = 101
Blocks for S = ceil(500/6) = 84 (one complete bucket of S),
Blocks for R = 101 - 84 = 17 blocks.

Read S:

We read S in chunks of 84 blocks (i.e. one complete bucket)
Hence, total read time:
$(100 + 84/2) * ceil(416/84) = 710\ ms$

Read R:
We read R in chunks of 17 blocks:
Hence, total read time:
$(100 + 17/2) * ceil(833/17) = 5316.5\ ms$

Hence, the total I/O time is:

$$4853 + 41808 + 9600.5 + 83716.5 + 710 + 5316.5 = 146004.5\ ms = approx\ 146\ sec$$

c) Sort-based join

Given: We write sorted sublists to consecutive blocks of disks.

Creating sorted sublists:

Read R:

We read R in chunks of 100 blocks.
Hence, read time:
$(100 + 100/2) * ceil(1000/100) = 1500\ ms$

Write R:

We write R in chunks of 100 blocks.
Hence, write time:
$(100 + 100/2) * ceil(1000/100) = 1500\ ms$

Read S:
We read S in chunks of 100 blocks.

Hence, read time:

$(100 + 100/2) * ceil(500/100) = 750\ ms$

Write S:

We write S in chunks of 100 blocks.

Hence, write time:

$(100 + 100/2) * ceil(500/100) = 750\ ms$

Sorting the sorted sublists:

Read R:

To maximally utilize the main memory space and the fact that the sublists were written in consecutive blocks, we read 10 blocks of each sorted sublist at a time. Thus, main memory used is => 10*10 = 100 blocks

Hence, the read time is:

$(10 * (100 + 10/2)) * 10 = 10500\ ms$

Write R:

We write back one block at a time. Hence, the write cost is:

$1000 * 100.5 = 100500\ ms$

Read S:

To maximally utilize the main memory space and the fact that the sublists were written in consecutive blocks, we read 20 blocks of each sorted sublist at a time. Thus, main memory used is => 20*5 = 100 blocks

Hence, the read time is:

$5 * (5 * (100 + 20/2)) = 2750\ ms$

Write S:

We write back one block at a time. Hence, the write cost is:

$500 * 100.5 = 50250\ ms$

Joining:

During the write step of sorting the sorted sublists, both R and S got stored in consecutive blocks. The minimum joining time is found when, we simply read 100 blocks of R and 1 block of S. This gives us, joining time:

Read R: $(100 + 100/2) * ceil(1000/100) = 1500\ ms$
Read S: $100.5 * 500 = 50250\ ms$

Hence, the total I/O time is:

$$1500 + 1500 + 750 + 750 + 10500 + 100500 + 2750 + 50250 + 1500 + 50250 = 220250\ ms = approx\ 220s$$

16.2.8.) a)

" Lets consider a table :

a	b
1	5
1	6
1	5
2	4
2	4

L.H.S ⇒ $(\gamma_{a, SUM(b) \to x}(R)) \Rightarrow$

a	b
1	16
2	8

$\gamma_{MIN(a) \to y, x}(\gamma_{a, SUM(b) \to x}(R)) \Rightarrow$

a	b
1	16
2	8

R.H.S ⇒ $(\gamma_{MIN(a) \to y, b}(R)) \Rightarrow$

a	b
1	5
1	6
2	4

$(\gamma_{y, SUM(b) \to x}(\gamma_{MIN(a) \to y, b}(R)) \Rightarrow$

a	b
1	11
2	4

So, L.H.S ≠ R.H.S.

They are not equal.

(b) Lets consider a table:

a	b
1	5
2	5
1	6
4	4

L.H.S ⇒ $(\gamma_{a, Max(b) \to x}(R)) \Rightarrow$

a	b
1	6
2	5
4	4

$(\gamma_{MIN(a) \to y, x}(\gamma_{a, Max(b) \to x}(R))) \Rightarrow$

a	b
1	6
2	5
4	4

R.H.S ⇒ $(\gamma_{MIN(a) \to y, b}(R)) \Rightarrow$

a	b
1	5
1	6
4	4

$(\gamma_{y, Max(b) \to x}(\gamma_{MIN(a) \to y, b}(R)))$

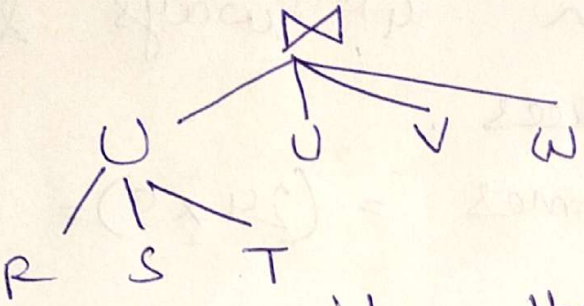
$\Rightarrow$

a	b
1	6
4	4

So, L.H.S ≠ R.H.S.

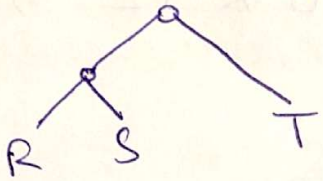
They are not equivalent.

16.3.5) This is the final tree :

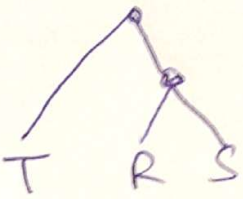


Let us first consider the union part.

Two structures are possible ⇒

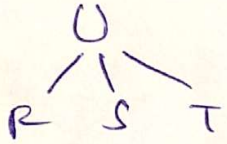


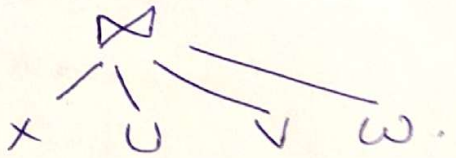
and



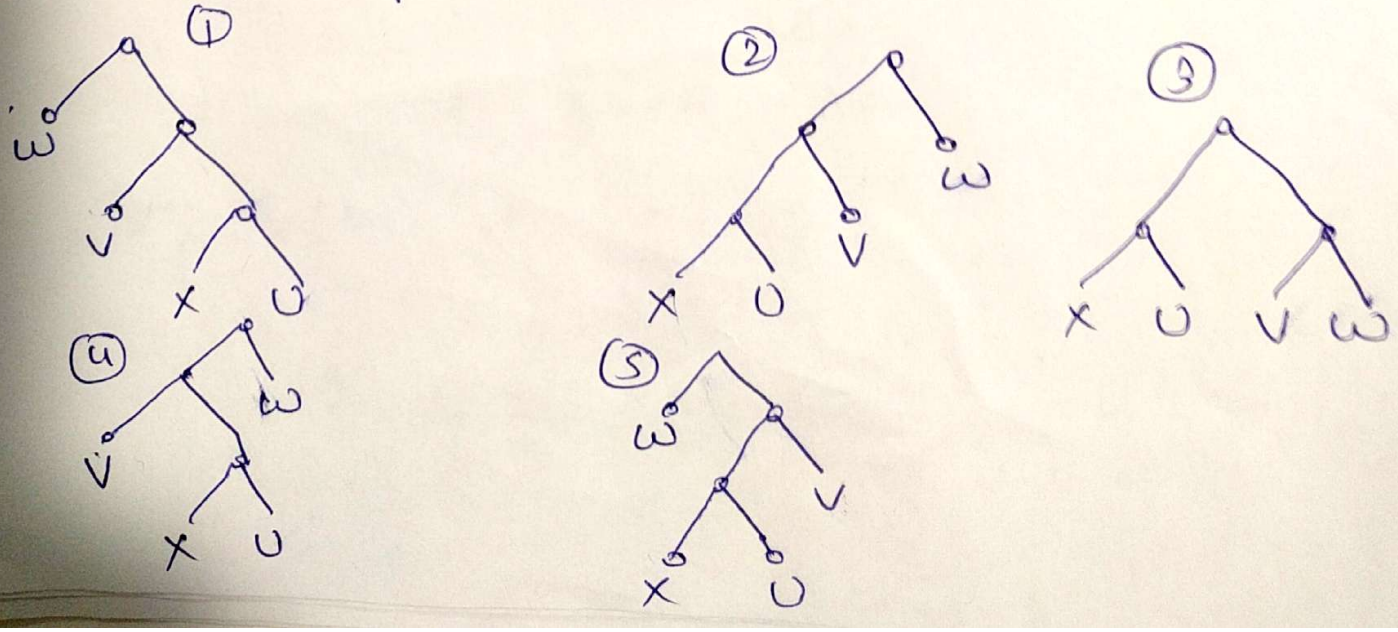
The leaves can be arranged or parenthesized in $3! = 6$ ways. There are 2 trees.

So total ways $= 6 \times 2 = 12$

Now let us denote  by x.

So our tree becomes 

5 trees are possible ⇒



In each of these leaves can be arranged in 4! ways & there are 5 such trees.

So, total becomes = $(24 \times 5) = 120$

So total exp tree are ⇒

$120 \times 12 \Rightarrow 1440$ trees.

16.4.1.)

$W(a,b)$	$X(b,c)$	$Y(c,d)$	$Z(d,e)$
$T(W) = 100$	$T(X) = 200$	$T(Y) = 300$	$T(Z) = 400$
$V(W,a) = 20$	$V(X,b) = 50$	$V(Y,c) = 50$	$V(Z,d) = 40$
$V(W,b) = 60$	$V(X,c) = 100$	$V(Y,d) = 50$	$V(Z,e) = 100$

(a) $W \bowtie X \bowtie Y \bowtie Z$

$$T(W \bowtie X) \Rightarrow \frac{T(W)\,T(X)}{\max\{V(W,b), V(X,b)\}} = \frac{100 \times 200}{\max\{60,50\}}$$

$$\Rightarrow \frac{100 \times 200}{60} \Rightarrow \frac{2000}{6}$$

Now we use preservation of values.

$\Rightarrow V(W \bowtie X, a) = 20$ $\quad V(X,c) = 100$

$V(W \bowtie X, b) = \min(60,50) = 50$

Now, $T(W \bowtie X \bowtie Y) = \frac{2000}{6} \times \frac{300^{50}}{\max(100,50)}$

$= \frac{2000 \times 50}{100} = 1000$

$T(W \bowtie X \bowtie Y \bowtie Z) \Rightarrow \frac{1000 \times 400^{8}}{50}$

$\Rightarrow 8000 \Rightarrow$ Ans.

b) $\sigma_{a=10}(W) \Rightarrow \frac{T(W)}{V(W,a)}$

$\Rightarrow \frac{100}{20} = 5$ Ans.

c) $\sigma_{c=20}(Y) = \dfrac{T(Y)}{V(Y,c)}$

$= \dfrac{300}{50} = 6$ Ans.

d) $\sigma_{c=20}(Y) \bowtie Z$

Now, $\sigma_{c=20}(Y) = \dfrac{T(Y)}{V(Y,c)} = \dfrac{300}{50} = 6$

$T(\sigma_{c=20}(Y) \bowtie Z) = \dfrac{T(\sigma_{c=20} Y) \times T(Z)}{\max\{\min\{V(Y,d), T(Y)\}, V(Z,d)\}}$

$\Rightarrow \dfrac{6 \times 400}{\max(\min(50, 6), 40)} = \dfrac{6 \times 400}{40} = 60$

e) $W \times Y = T(W) \times T(Y) = 100 \times 300$

$= 30000$

f) $\sigma_{d>10}(Z) \Rightarrow \dfrac{1}{3} T(Z)$ Assume that 1/3 values are greater than 10.

$\Rightarrow \left\lceil \dfrac{1}{3} (400) \right\rceil \Rightarrow 134$

g) $\sigma_{a=1 \text{ AND } b=2}(W) = \dfrac{T(W)}{V(W,a) \times V(W,b)}$

$= \dfrac{100}{20 \times 60} = \dfrac{1}{12} = 1$ tuple

We consider 1/12 as 1 as otherwise none of conditions could be satisfied.

h) $\sigma_{a=1 \text{ AND } b>2}(\omega)$

$\Rightarrow$ We assume 1/3 b has value greater than 2.

$\Rightarrow \dfrac{T(\omega)}{V(\omega, a) \times 3} \Rightarrow \dfrac{\cancel{10}^{5}\cancel{\phi}}{\cancel{2\phi} \times 3} = \left\lceil \frac{5}{3} \right\rceil$

$= 2$ tuples.

i) $X \bowtie_{X.C < Y.C} Y$

In this we do cartesian product followed by condition. So we assume only 1 third satisfies that condition.

$\Rightarrow \dfrac{T(X) \times T(Y)}{3} \Rightarrow$

$= \dfrac{200 \times \cancel{300}^{100}}{\cancel{3}} = 20000$

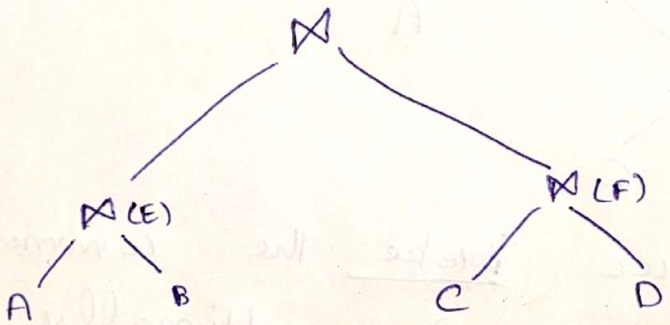
7) 16.5.7.) On this question we have to prove that if E ⋈ F are not on top, then we would get an optimal value. So let us consider an example of a query plan.

A ⋈ B ⋈ C ⋈ D

E = A ⋈ B F = C ⋈ D

Optimal plan of E is only A ⋈ B

Optimal plan for F is only C ⋈ D



Now we have to make an example where we first combine E & F.

Let us consider that B & C do not have any common value for join attribute.

There can be many examples for that.

1) B could be $\sigma_{d=100}$ X

C could be $\sigma_{d=50}$ Y

where d is join attribute.

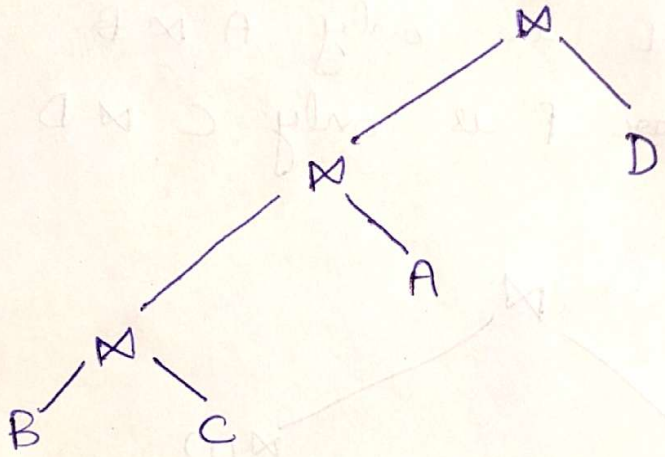
2) B could be $\sigma_{p>500}$ X

C could be $\sigma_{p<300}$ X

p is join attribute.

Now if we join $B \bowtie C$ first, then we are left with relation with 0 tuples. Now combining relation with 0 tuple with A & D would result in minimum cost.

Exp tree :



So, this way we broke the common idea that calculating E & F optimally & then joining $E \bowtie F$ at top would result in optimal query plan.