ASSIGNMENT:7-

QUESTION:1-
SOLUTION:
The attribute Headquarter of the relation Company has fewer duplicates than the other attribute of the relation i.e. Cname. Hence, indexing using the attribute with more duplicates appears to be a good option. If we keep only the record in the index for the first data record, Cname, with each search key value and find all the data record with that search key, follow the pointer in the index and then move forward to find the records of the attribute Headquarter.

QUESTION:2-
SOLUTION:
Transaction (tid: integer, timestamp: date, amount: float)

The relation transaction undergoes 10000 modifications every second. In such a situation, indexing the relation will help in reducing the execution time and speed up the execution of the queries.

However, making a large number of changes so quickly can be very expensive. It is also important to find the appropriate technique for indexing the relation. An optimized solution must be able to help in reduce the execution time of the query without increasing the expenses beyond limit or budget.

QUESTION:3-
SOLUTION:
block size = 4096 bytes
block-address size = 12 bytes
block access time (I/O operation) = 15 μs
record size = 150 bytes
record primary key size = 10 bytes

A)
Finding the largest value of n for the B+ tree such that:
$n \leq \frac{blocksize-|block\ address|}{|block\ address|+|key|}$

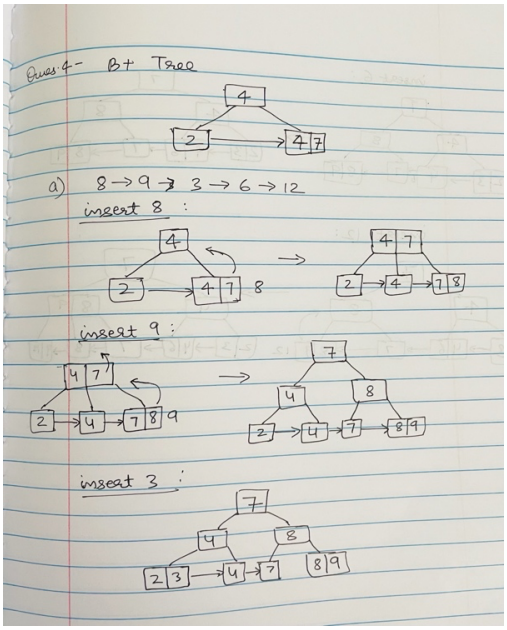
$n \leq \frac{4096-12}{12+10}$
thus, n=185
The minimum time be:
$(\lceil \log_{186} 10^9 \rceil + 1) * 15$
$= (\lceil 3.96 \rceil + 1) * 15$
$= 75$ μs

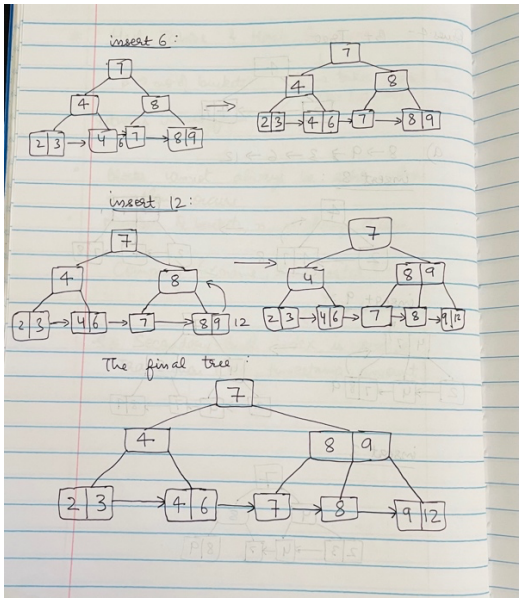
B)
The branching factor is minimum when the height of the tree is maximum:
$\lceil 185/2 \rceil + 1 = 94$
Height of the tree $= \lceil \log_{94}(10^9)/2 \rceil$
$= \lceil \log_{94} 5 * 10^8 \rceil$

Height = 4.408
Maximum time to insert a record = 4.408 * 15 µs
= 66.12 µs

C)
Memory size to hold the first two levels of the B+Tree = (n+1)* |blocksize|
Memory = 186 * 4096
=761856 bytes

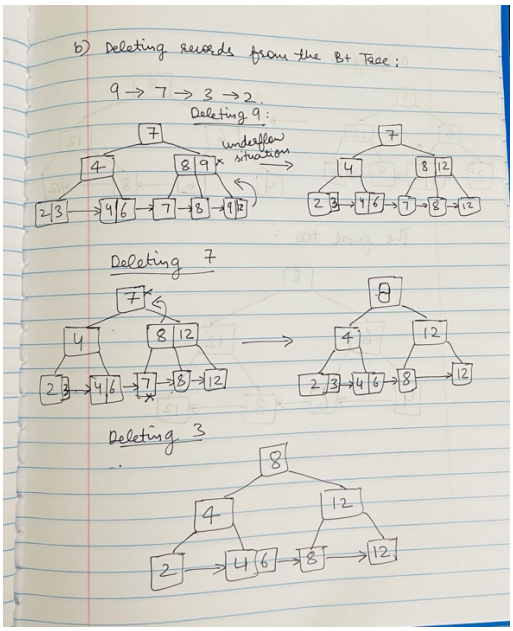
QUESTION:4-
SOLUTION:
A)
INSERTION INTO B+ TREE

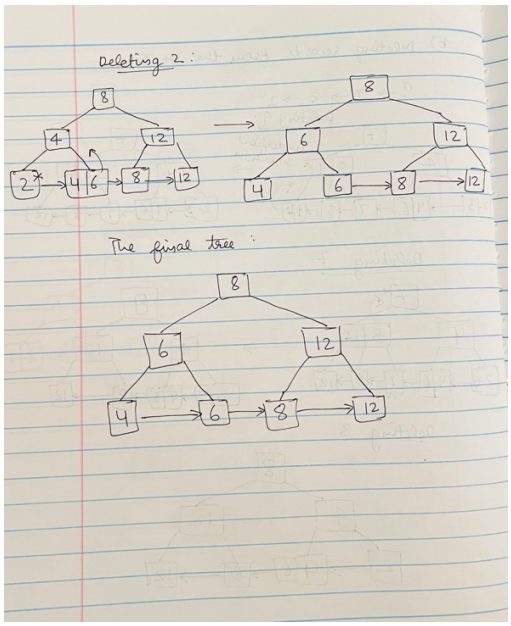




B)

DELETING KEYS FROM B+ TREE:





QUESTION:5-

A)

```
select pid, cname
from worksFor
where salary between s1 and s2;
```

For salary range 15000 and 30000

Records	Execution time without index	Execution time with index
100	0.057 ms	0.032 ms
1000	0.257 ms	0.163 ms
10000	1.391 ms	1.352 ms

The above calculations are performed by creating an index using hashing. The salary difference chosen is of $15000.

B)

Now, using B-Tree for indexing and keeping the same salary range 15000 & 30000:

Records	Execution time without index	Execution time with index
100	0.133 ms	0.040 ms

1000	0.303 ms	0.150 ms
10000	1.773 ms	1.392 ms

As per the results, it can be concluded that using B-Tree for indexing and performing the above-mentioned query makes the execution faster as compared to the case when hashing is used for indexing.

Now, making changes to the salary range of the query and slightly increasing the gap between the upper and the lower bound.

For salary range 10000 and 30000

Records	Execution time without index	Execution time with index
100	0.250 ms	0.045 ms
1000	0.944 ms	0.165 ms
10000	1.549 ms	1.448 ms

Question:6-

```
select pid, pname
from Person
where pid in (select pid from worksFor where cname = c);
```

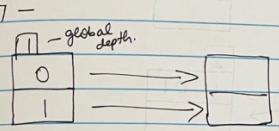
For c= ‘Salesforce’

Records	Execution time without index	Execution time with index
100	0.084 ms	0.083 ms
1000	0.415 ms	0.422 ms
10000	3.410 ms	3.337 ms

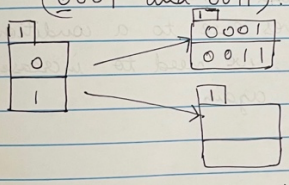
For cname=’Salesforce’, we can observe that there is not a significant change in the execution time of the above query when computed with and without index.

QUESTION:7-
SOLUTION:

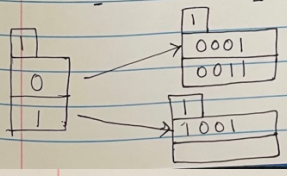
Ques: 7 –



a) Inserting records with keys 1 and 3.
(0001 and 0011).

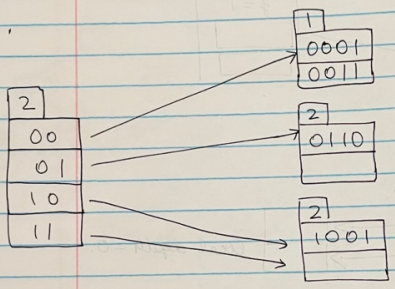


b) Inserting records with keys 6 and 9.
0110 and 1001



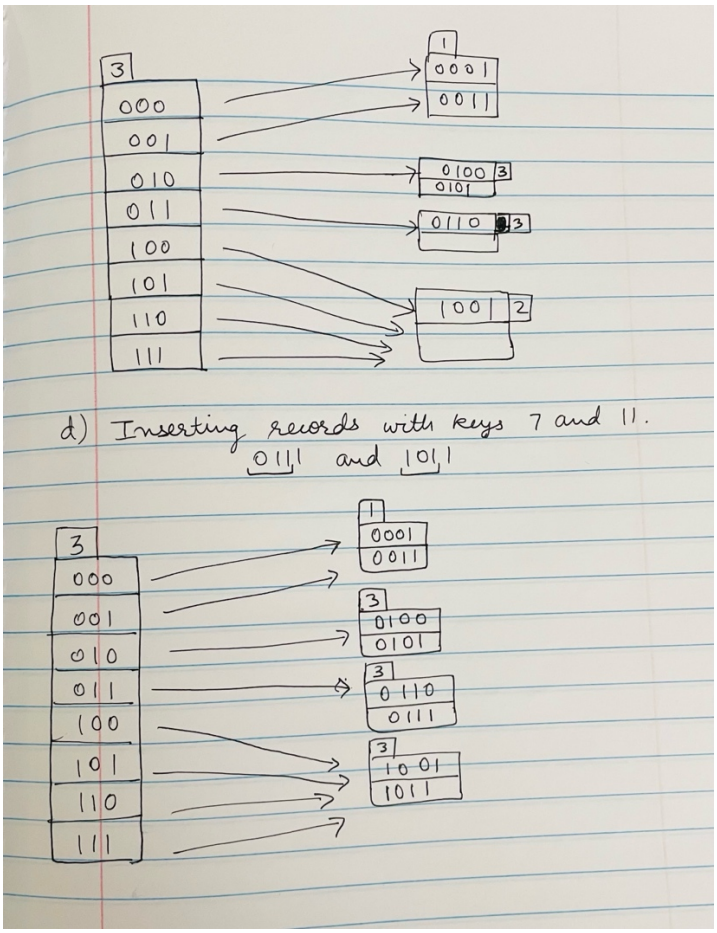
Now, there occurs a situation of overflow when we insert 0110.

So, we increase the global depth.



c) Inserting records with keys 4 and 5.
0100 and 0101

→ This again gives rise to a condition of overflow. So, we need to increase the global depth again.



QUESTION:8-
SOLUTION:
The major problem here is clustering. Many consecutive elements form groups. This means that if many collisions occur at the same hash value, a number of surrounding slots will be filled by linear probing resolution.

Given, linear hash function, $h(i) = i^2 \bmod(B)$
Here I describe the significant number of bits in the hashing function output. Only the I significant bits are considered while storing the record in a bucket.

a) For B=10, computing h(i):

i	$h(i) = i^2 \bmod(B)$
1	1
2	4
3	9
4	6
5	5
6	6

7	9
8	4

B) For B=16, computing h(i):

i	$h(i) = i^2 \bmod(B)$
1	1
2	4
3	9
4	0
5	9
6	4
7	1
8	0
:	
:	
:	

c) As the number of buckets increase, the number of overflows must decrease as a bucket would have sufficient accommodation available and the number of collisions will be reduced as well.