ADC Assignment 6

Aditya Shekhar Camarushy, Bhavik Kollipara, Sai Prajwal Reddy, Srikar Devulapalli

Q2)

skill text	count_rec bigint	total_rec numeric	prob numeric
Databases	4	10	0.40
Networks	3	10	0.30
Programmi...	1	10	0.10
AI	2	10	0.20

Here we can see that as per the probability distribution defined initially in the Probability mass function (pmfSkill)

Q3) (a)

We can notice that as the size increases the amount of time taken for sorting and scanning also increases.

We can also see that as the memory size is increased the amount of time taken for sorting decreases. The reason that the time taken is high when the memory is less is because postgresql uses merge sort when the data does not fit in the memory. However, when the data fits in the memory postgresql uses quick sort and hence less time taken.

(b)

We can see that the average time that is required to sort is quite more than the average time that is required for sorting.

We can also see that as we are increasing the memory, the amount of time that is required for sorting is becoming less as we increase the memory until a certain point after which the time remains almost same.

However, the time required for scanning is almost the same even after increasing the memory.

In the case of 32mb and 256mb work memory, the time taken is almost same as the data fits in the memory in both the cases and hence quick sort is performed in both the cases and hence the almost same time.

(c) We can see from the tables that the amount of time taken to create indexes is higher than

the amount of time taken to perform sorting. We can also see that as the sizes increases, the time taken is also higher for both creating and sorting.
However, when we compare the times for sorting using indexes to the times for sorting without indexes, we can see that the time taken for sorting using indexes is very less. This is because sorting by index using B tree is faster compared to merge sort or quick sort that was used by postgresql before indexing.

Q4) (c) We can see that initially for records below 100000, the amount of time taken to remove duplicates using distinct and group by are almost the same. However, as the number of records increase above that, we can see that the amount of time taken is less when distinct is used compared to group by.
When we use explain analyze on both of them, we can see that the query plan that is used by distinct and group by is essentially the same. We can however see that the time taken for execution in this case is basically less in the case of group by than distinct. However, this difference is very less.
Distinct is basically Group by that is performed without aggregation to remove duplicates and hence in most cases, the time taken is approximately the same.

Q7)

(a) Given Parameters –

block size = 8192 bytes
block-address size = 10 bytes
block access time (I/O operation) = 15 ms (micro seconds)
record size = 200 bytes
record primary key size = 8 bytes
N = 10^{10}

Let us calculate the number of keys in a node (n) –

$$n \leq \frac{blocksize - |blockaddress|}{|blockaddress| + |key|}$$

$$n \leq \frac{8192 - 10}{8 + 10}$$

$$n \leq 454$$

In the best case scenario, the key would be the first element of the leaf node (hence the +1).

∴ **Minimum time** = ($\lceil \log_{454} 10^{10} \rceil + 1$) $* 15$ = **71.45 ms**

(b) For maximum insertion time we assume that the height of the tree is maximum, this means that the branching factor is minimum which is 2.

At the root : 2
At non-leaf nodes : n/2 = $\lfloor 454/2 \rfloor = 227$

⇨ **Maximum time** = $\left\lceil \log_{n/2}(N/2) \right\rceil$

∴ **Maximum time** = $\left\lceil \log_{227/2}(10^{10}/2) \right\rceil$ * 15 = 5*15 = **75ms**

(c) To hold the first two levels of B tree the main memory

We assume that the first 2 levels consists of the root node and the first level blocks

= 1 + 454 = **455 blocks**

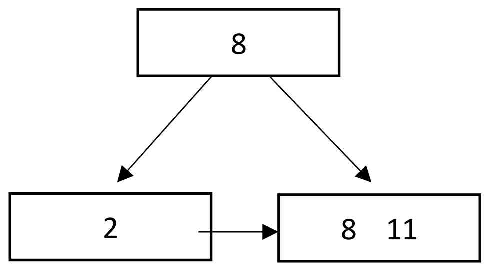
∴ The memory required to hold **first 2 levels** = 455 * 8192 = **3,727,360 bytes**

Similarly, To hold the first 3 levels = 1 + 454 + 454 * 454 = **206,571 blocks**

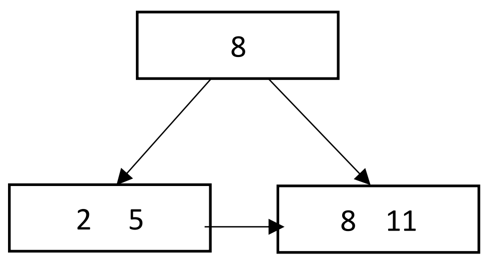
∴ The memory required to hold **first 3 levels** = 206571 * 8192 = **1,692,229,632 bytes**

Q8)

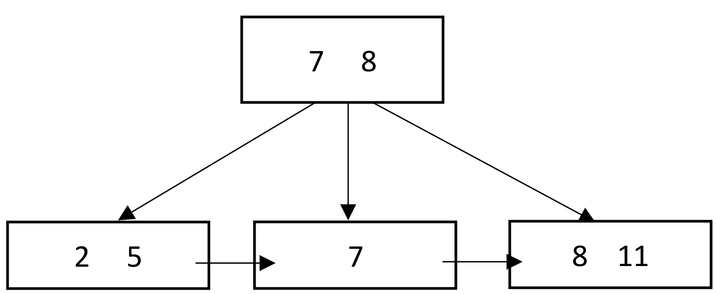
(a) Consider the following B+-tree of order 2 that holds records with keys 2, 8, and 11.



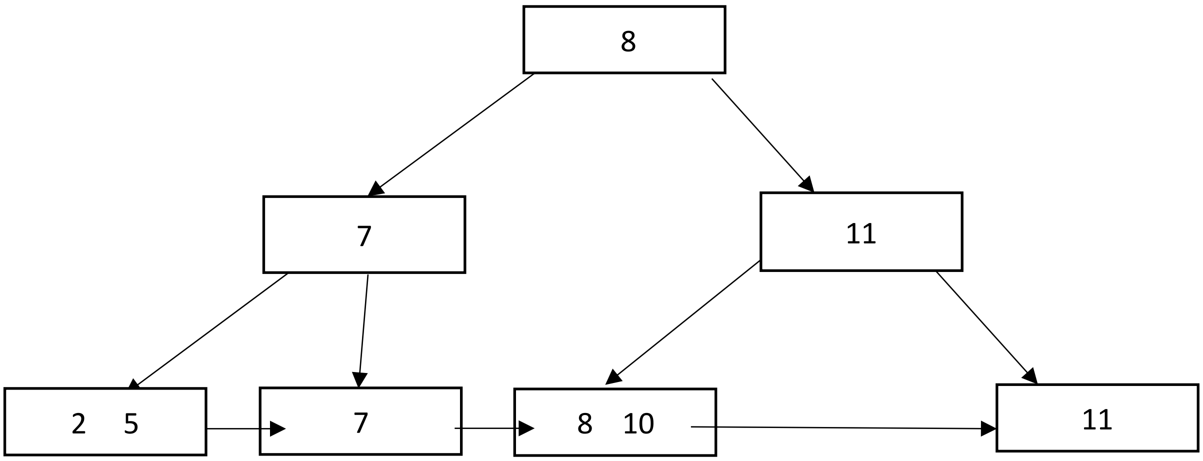
Insert 5



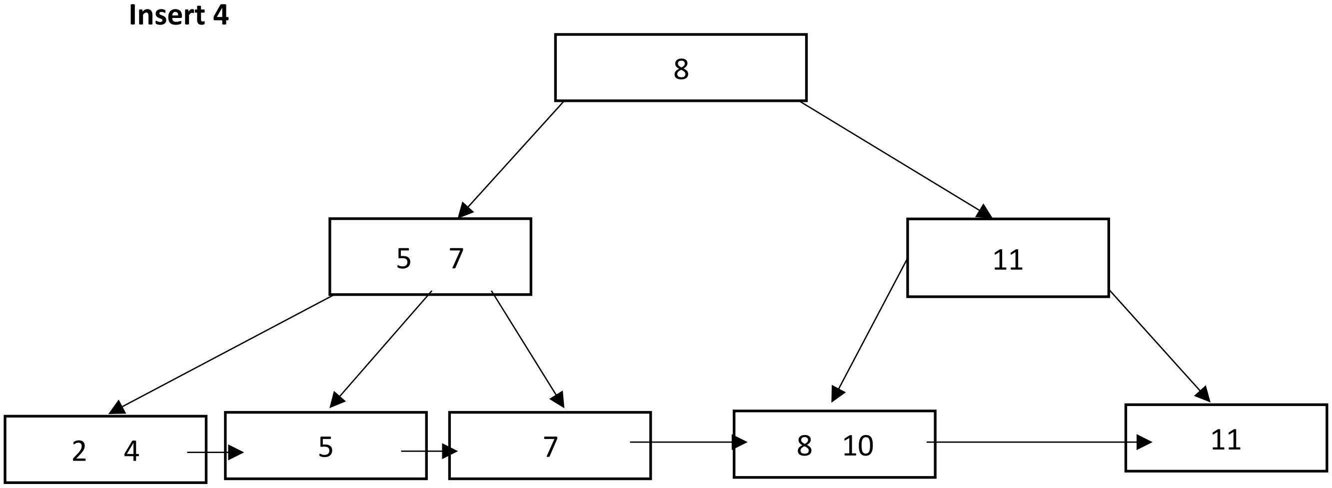
Insert 7



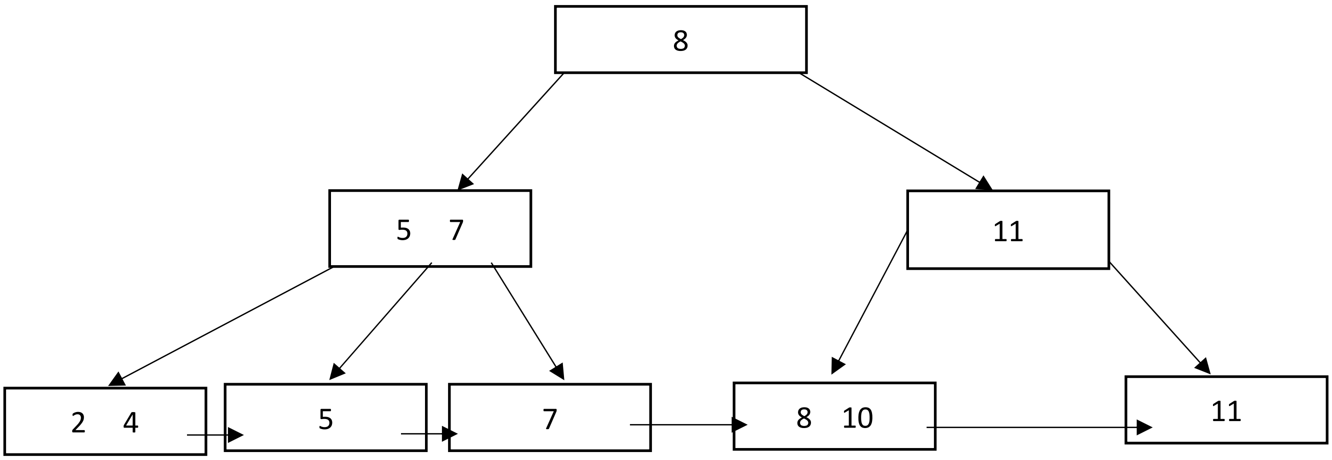
Insert 10



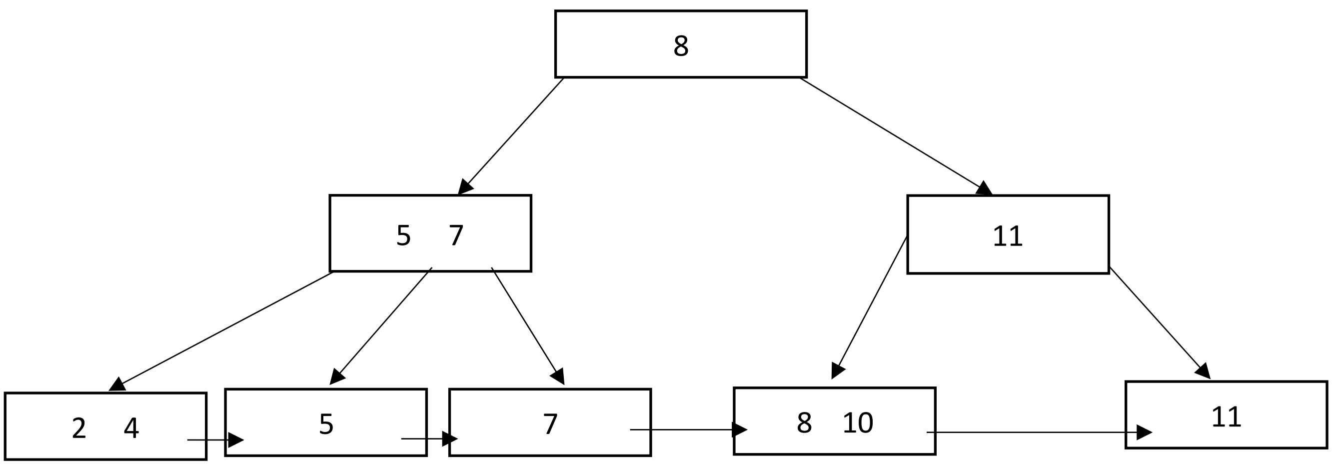
Insert 4



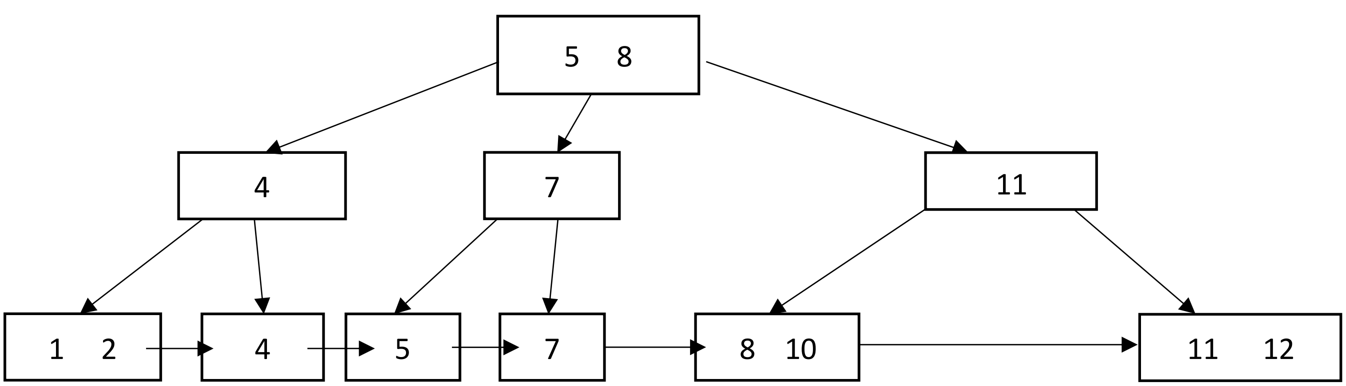
Insert 10 (No change in the B + - tree as 10 already exists record goes to datafile)



Insert 12

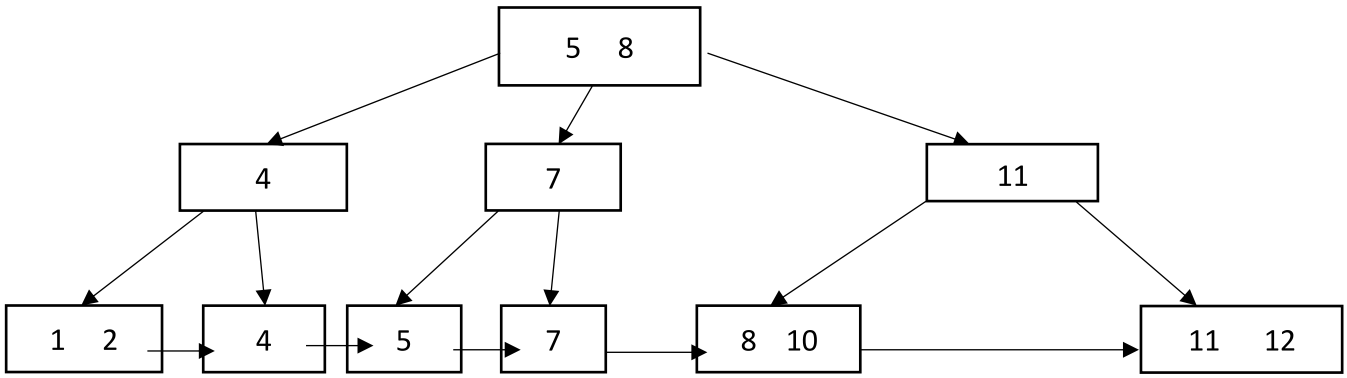


Insert 1

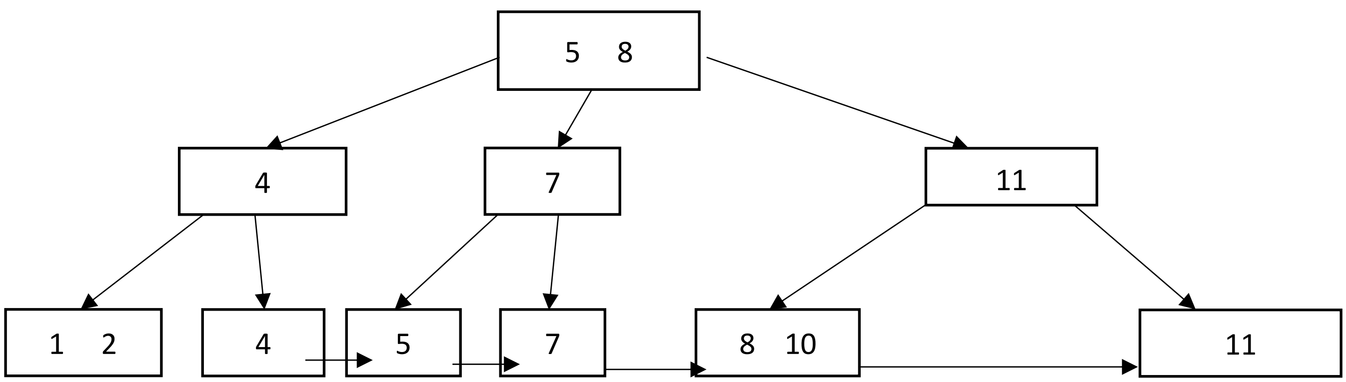


(b)

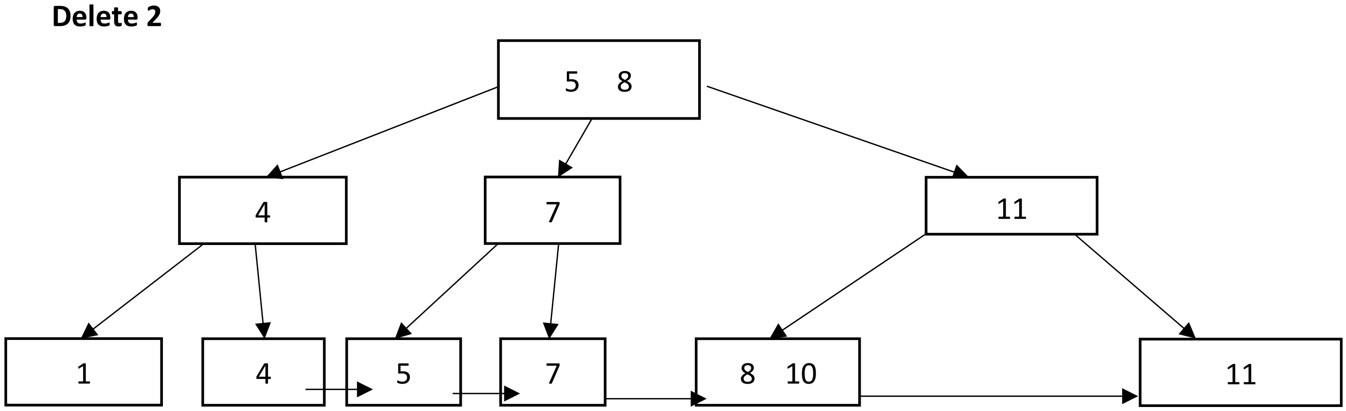
The B + tree from part (a)



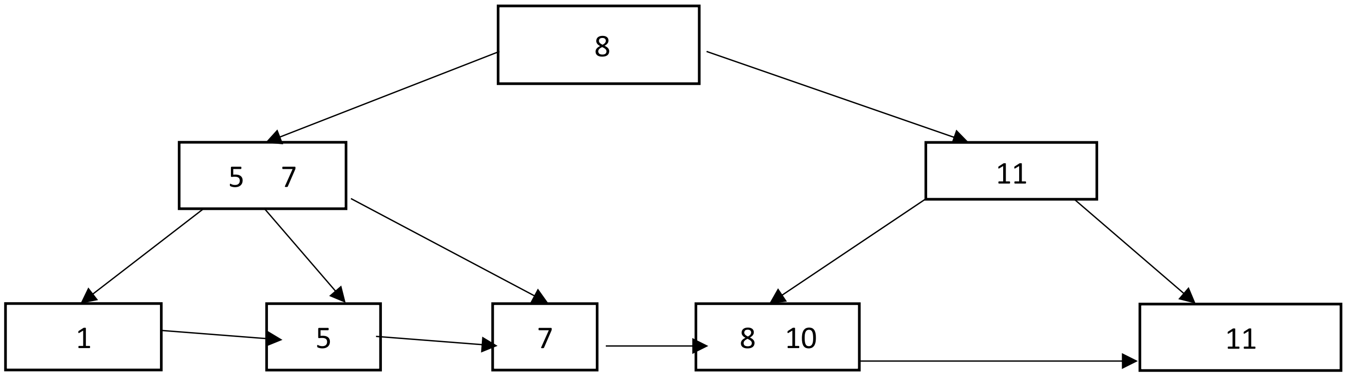
Delete 12



Delete 2

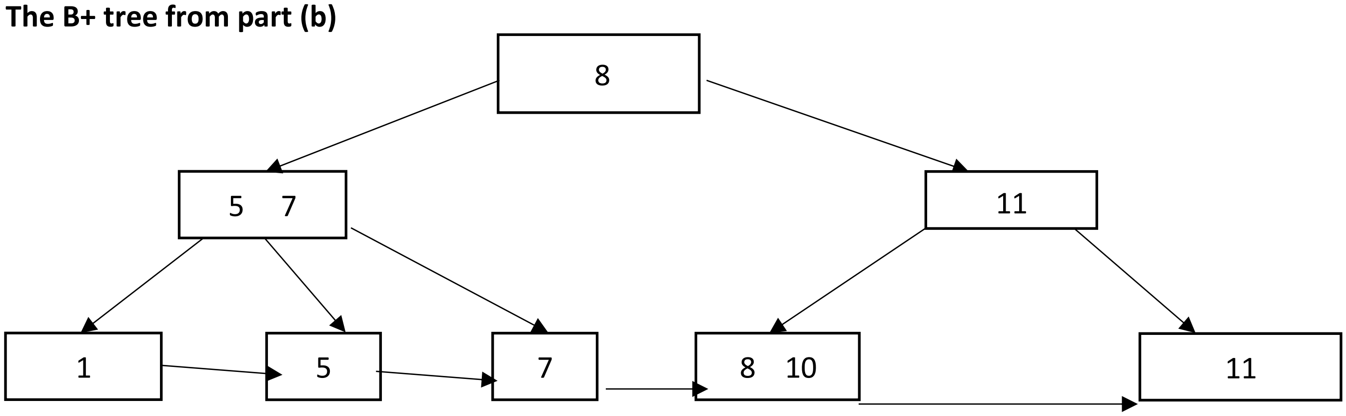


Delete 4



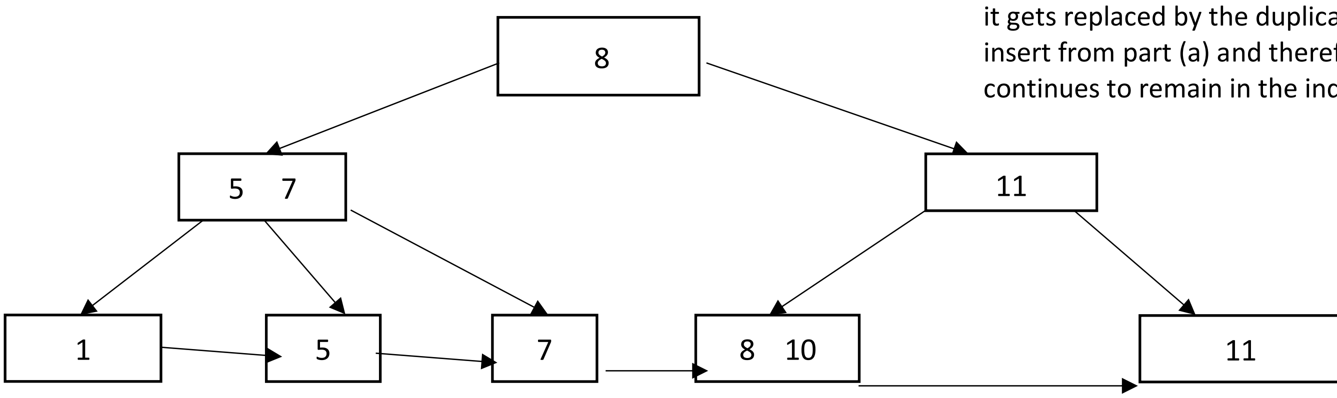
(c)

The B+ tree from part (b)

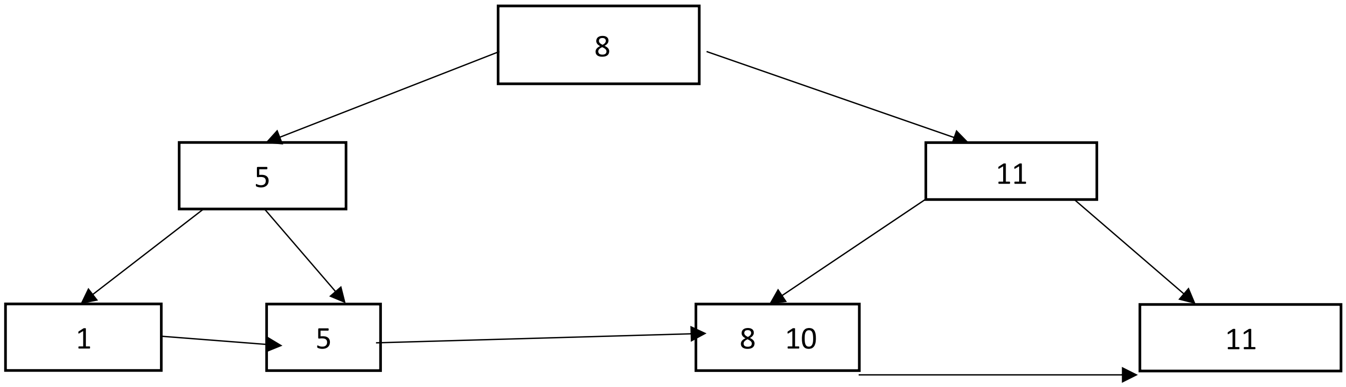


Delete 10

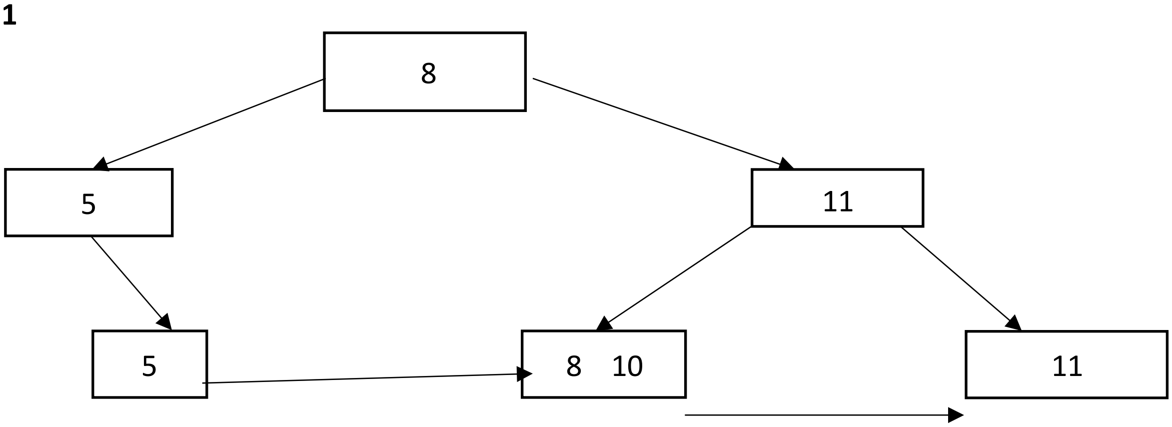
NOTE : We delete 10 here, however it gets replaced by the duplicate 10 insert from part (a) and therefore 10 continues to remain in the index



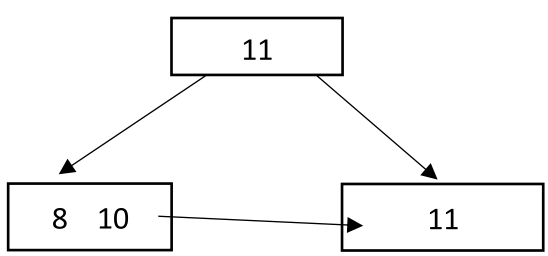
Delete 7



Delete 1

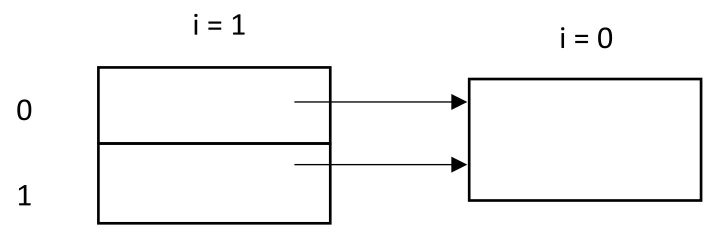


Delete 5

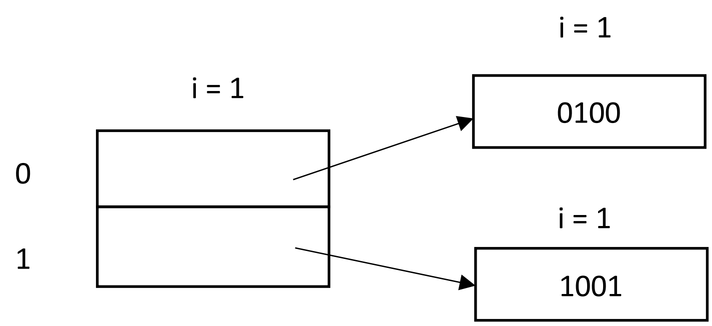


Q9)

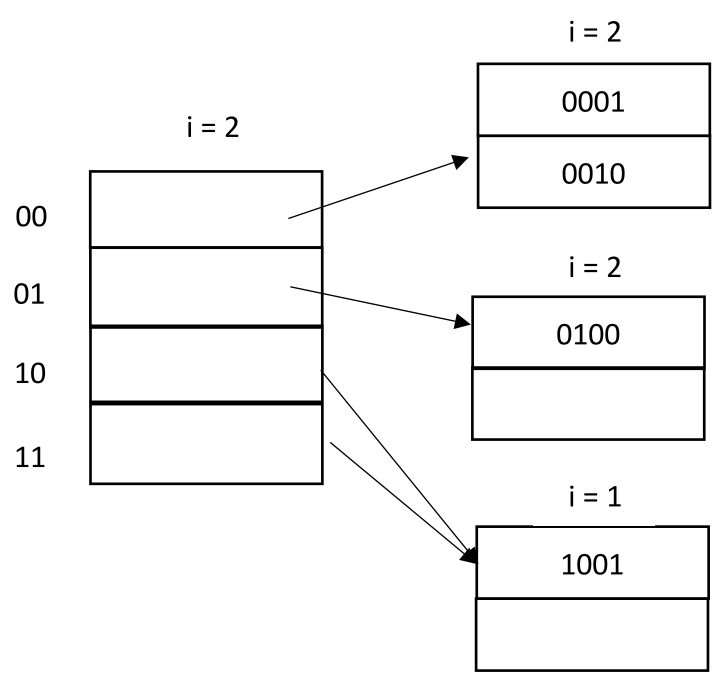
(a)



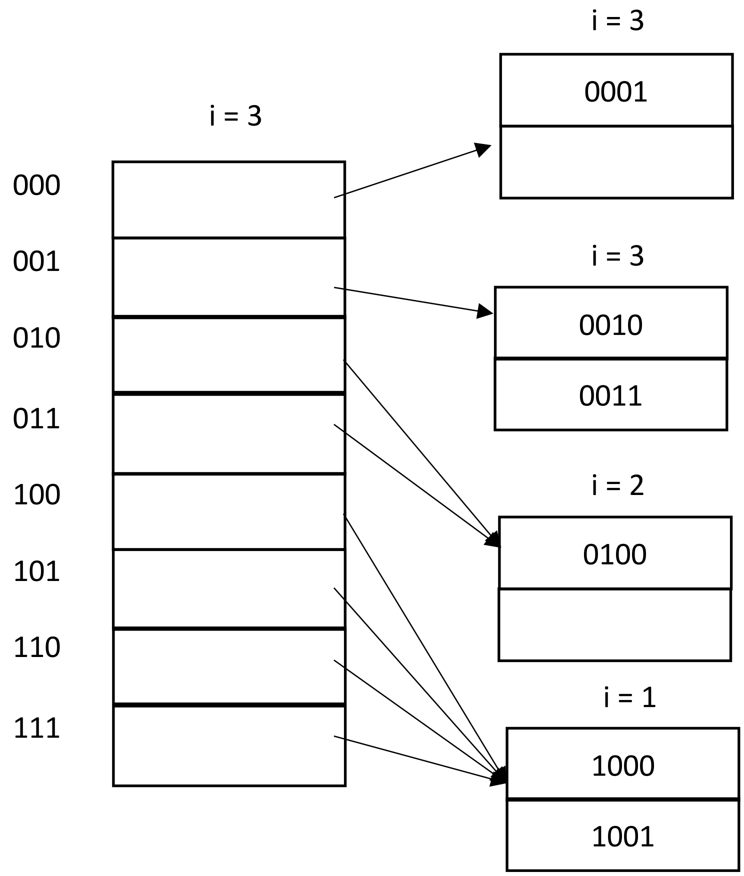
(i) Insert 9 (1001) and 4 (0100)



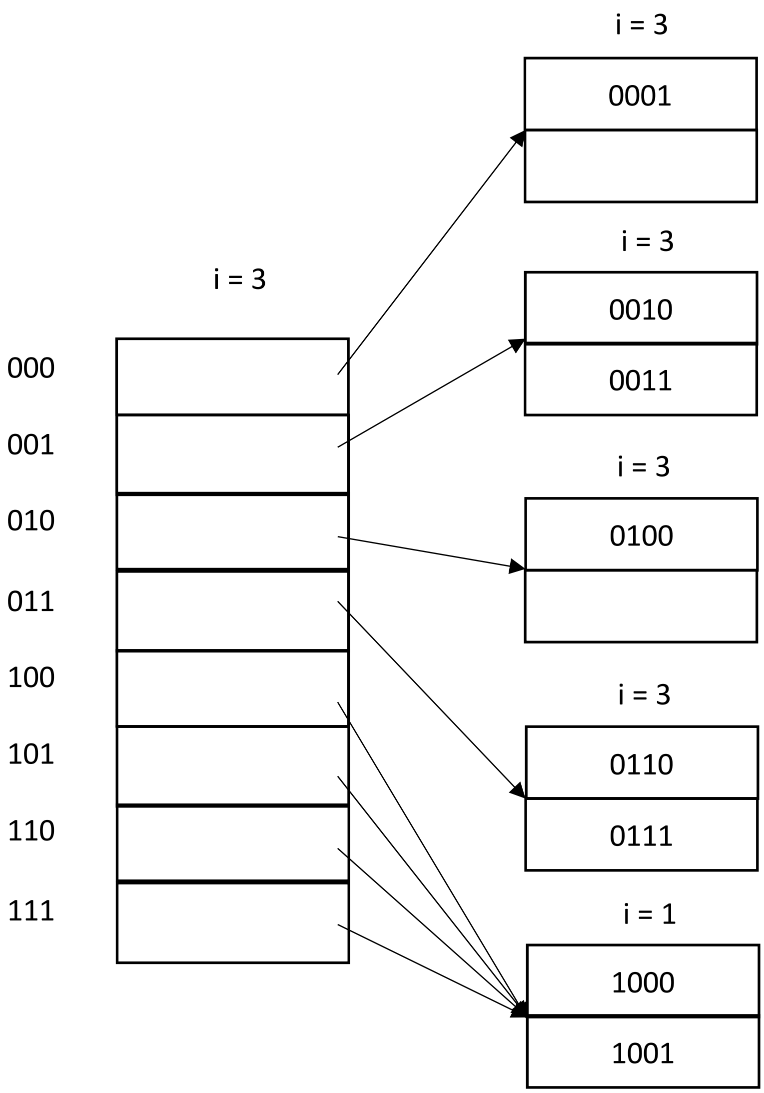
(ii) Insert 1 (0001) and 2 (0010)



(iii) Insert 8 (1000) and 3 (0011)

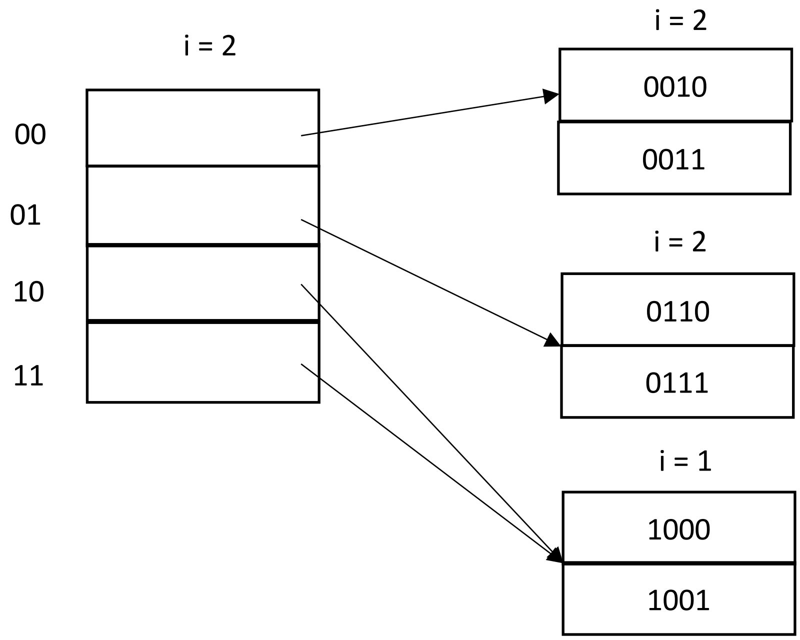


(iv) Insert 6 (0110) and 7 (0111)

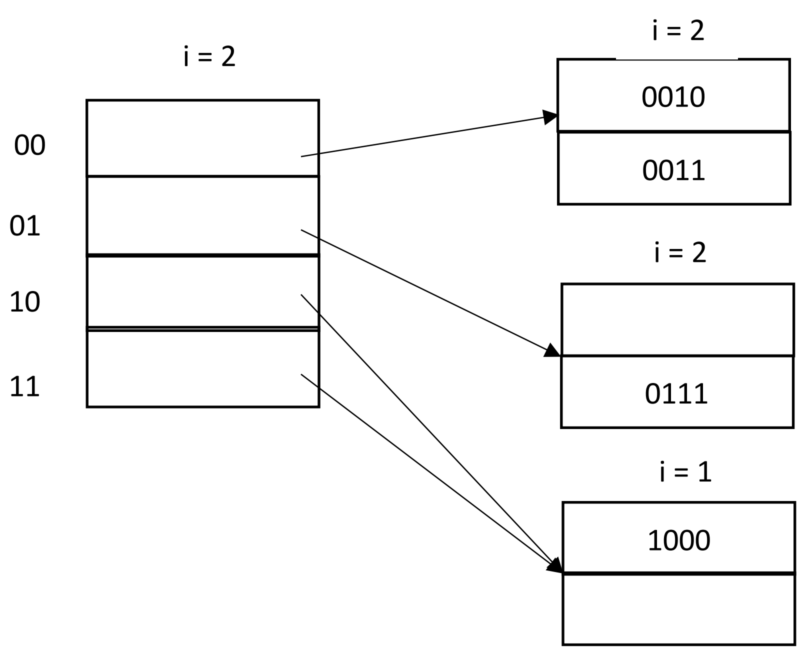


(b)

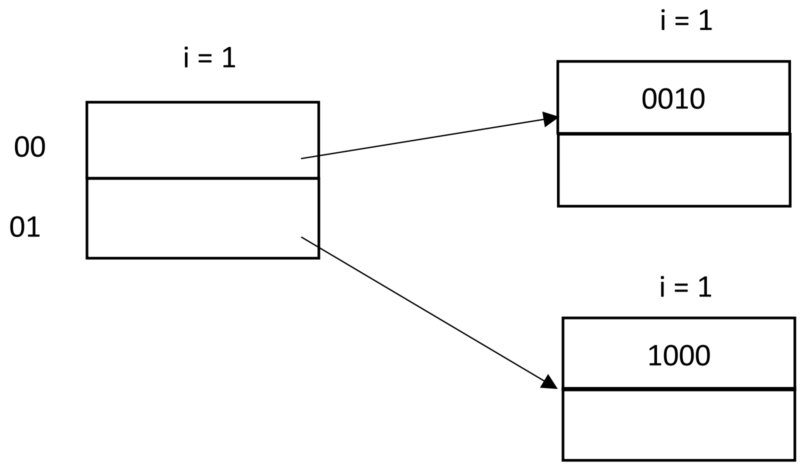
(i) Delete records with keys 4 (0100) and 1 (0001)



(ii) Delete Records with keys 9 (1001) and 6 (0110)



(iii) Delete Records with keys 7 (0111) and 3 (0110)



Q 10)

Number of Total Records	Avg Exec. Time without Index:	Avg. Exec. Time with Index:
10000	0.651	0.223
100000	5.450	2.301
1000000	50.126	24.527

Q 11)

a. Small Range Execution:

Number of Total Records	Avg Exec. Time without Index:	Avg. Exec. Time with Index:
100000	23.297	1.409
1000000	193.031	13.125
10000000	1141.897	198.567

b. Intermediate Range Execution:
We can see a small discrepancy in the following where in when the number of total records is 1000000, the average time taken without index is less than the average time taken with index.

Number of Total Records	Avg Exec. Time without Index:	Avg. Exec. Time with Index:
100000	32.706	24.716
1000000	263.306	266.025
10000000	2361.667	2323.625

c. Maximum Range Execution:

Number of Total Records	Avg Exec. Time without Index:	Avg. Exec. Time with Index:
100000	27.060	9.210
1000000	246.500	93.820
10000000	1757.513	1207.124

Q 12)

Number of Total Records	Avg Exec. Time without Index:	Avg. Exec. Time with Index:
100000	4.011	0.033
1000000	72.061	0.033
10000000	214.805	0.059

Q 13)

Number of Total Records	Avg Exec. Time without Index:	Avg. Exec. Time with Index:
10000	2.108	1.799
100000	21.341	17.976
1000000	238.002	192.713

Q 14)

We can see a small discrepancy in the following where in when the number of total records is 10000, the average time taken without index is less than the average time taken with index and also when the number of records is 1000, the time is almost the same in both cases.

Number of Total Records	Avg Exec. Time without Index:	Avg. Exec. Time with Index:
10	0.337	0.053
100	0.500	0.159
1000	1.937	1.951
10000	19.058	23.449