Calculations on tables.

Goods Table

We have a goods table with 24,00,000 records
size of 1 goods entry=54 bytes.
Blocking factor=floor(512/54)=9. => 9 records per block.

So no. of blocks required=ceiling(24,00,000/9)=266667 blocks.

Case 1:
No ordering based on any field.
Linear search will take o(n/2) block accesses,
so approximately 266667/2 = 133334 block accesses.

Case 2:
Ordering based on goodsid,
so binary search would require ceiling(log(266667) to the base 2)=19 block accesses.

Case 3:
Index based on goodsid,
Size of goodsid=4 bytes, block ptr size = 6 bytes.
So size of index entry=4+6=10 bytes.
So no of index entries per block=floor(512/10)=51 entries.

Total no. of index entries = total no. of blocks in data file = 266667

so index blocking factor(no. of blocks required to store index entries,bfri)=
ceiling(266667/51)=5229 blocks

so now search will take ceiling(log(5229) to base 2)+1=13+1=14 block accesses

Case 4:
Multilevel index on goodsid,
1^{st} level index(bfri) = 5229 entries
2^{nd} level index = ceiling(5229/51) = 103 entries
3^{rd} level index = ceiling(103/51)=3 entries
4^{th} level index = ceiling(3/51)=1 entry.

So no. of block accesses = 4(level of indexing) + 1=5 block accesses.

Case 5: (for search based on brand)
Clustered index on brand, ordered by brand
We have 24,000 brands each having 100 goods.
Size of brand=10 bytes
Size of block ptr = 6 bytes.
Size of index entry = 10+6=16 bytes.

So no. of index entries per block=floor(512/16)=32 entries.
no. of index entries = no. of brands = 24,000
so no. of blocks=ceiling(24,000/32)=750 blocks.

So no. of block accesses by binary search = ceiling(log(750) to base 2)+1=11 block accesses

Case 6:
Multilevel clustered index on brand, assuming ordered by brand
Size of brand=10 bytes
Size of block ptr = 6 bytes.
Size of index entry = 10+6=16 bytes.

So no. of index entries per block=floor(512/16)=32 entries.

no. of index entries = no. of brands = 24,000

so no. of blocks=ceiling(24,000/32)=750 blocks.

=>
1^{st} level index = 750
2^{nd} level index = ceiling(750/32)=24
3^{rd} level index = ceiling(24/32)=1

so no. of block accesses = 3(level of indexing) + 1=4 block accesses.

Case 7:
B-Tree Index on goodsid

Block pointer size P = 6 bytes
Record pointer size R = 7 bytes
Size of search key field = 4 bytes
Block size B = 512 bytes

Number of goods = 24 lakh
A node in a B-Tree can have at most p pointers, p-1 entries and p-1 data pointers. Since all these need to fit within a block, we have :

6p + 4(p-1) +7(p-1) <= 512
p approximately equal to 29

Assuming that 69 percent of the nodes are full when the number of values in the tree stabilizes, we have p = 0.69p = 20

No. of levels = log(2400000)/log(20) = 5 levels
If we don't consider the root we have levels = 4

Level	Nodes	Entries	Pointers
Root	1	19	20
Level 1	20	380	400
Level 2	400	7600	8000
Level 3	8000	1,52,000	1,60,000
Level 4	1,60,000	30,40,000	

No. of entries and pointers barring the last level = 328419
Remaining entries = 2400000-159999 = 2071581

No. of nodes in the last level = Remaining entries+pointers/no. of entries+pointers in one node =53118
Since size of one node <= size of a block, we will have to fit one node per block

Therefore number of blocks = 53118
Number of block accesses = level +1= 5 (not including root)

Summarizing.

Property	Block Accesses	No. of extra blocks required
No ordering	133334	0
Ordering on goods id	20	0
Index on goods name	14	5229
Multilevel index on goods id	5	5229+103+3+1=5363
Clustered Index on brand	11	750
Multilevel Clustered Index on brand	4	750+24+1=775
B-tree index on goods id	5	53118+21+400+8000 = 61538

Looking at these figures, we conclude that using a Multilevel Index on goods id is better than B-Tree Index on goods id.
We will also include Multilevel clustered index on brand to optimize our search queries.

Seller Table

We have a seller table with 10,000 records.

Seller record size=48 bytes.

Blocking factor=floor(512/48)=10 entries per block

No. of blocks required = ceiling(10,000/10)=1000

Case 1:
No ordering based on any field.
Linear search will take o(n/2) block accesses,
so approximately 1000/2 = 500 block accesses.

Case 2:
Ordering based on sid,
so binary search would require ceiling(log(1000) to the base 2)=10 block accesses.

Case 3:
Index based on sid,
Size of sid=20 bytes, block ptr size = 6 bytes.
So size of index entry=20+6=26 bytes.
So no. of index entries per block=floor(512/26)=19 entries.

Total no. of index entries = total no. of blocks in data file = 1000

so index blocking factor(no. of blocks required to store index entries,bfri)=
ceiling(1000/19)=53 entries

so now search will take ceiling(log(53) to base 2)=6 block accesses

Case 4:
Multilevel index on sid,
1^{st} level index(bfri) = 53 entries
2^{nd} level index = ceiling(53/19) = 3 entries
3^{rd} level index = ceiling(3/19)=1 entry

So no. of block accesses = 3(level of indexing) + 1=4 block accesses.

Case 5:
Btree : Index on Seller id
Block pointer size P = 6 bytes
Record pointer size R = 7 bytes
Size of search key field = 20 bytes
Block size B = 512 bytes

Number of goods = 10000

A node in a B-Tree can have at most p pointers, p-1 entries and p-1 data pointers. Since all these need to fit within a block, we have :

6p + 20(p-1) +7(p-1) <= 512
p approximately equal to 15

Assuming that 69 percent of the nodes are full when the number of values in the tree stabilizes, we have p = 0.69p = 11

No. of levels = log(10000)/log(11) = 4 levels
If we don't consider root levels = 3

Level	Nodes	Entries	Pointers
Root	1	10	11
Level 1	11	110	121
Level 2	121	1210	1331
Level 3	1331	13310	14641

No. of entries and pointers barring the last level = 2793
Remaining entries = 10000-2793= 7207

No. of nodes in the last level = Remaining entries+pointers/no. of entries+pointers in one node = 7207/21 = 343
Since size of one node <= size of a block, we will have to fit one node per block

Therefore number of blocks = 343
Number of block accesses = level +1 = 4

Summarizing.

Property	Block accesses	No. of extra blocks required
No ordering	500	0
Ordering on seller id	11	0
Index on seller id	7	53
Multilevel index on seller id	4	53+3+1=57
B-tree index on seller id	4	343+121+11+1=476

We have chose Multilevel index on Seller Id

Customer Table

We have a customer table with 24,000 records.

Customer record size=154 bytes.

Blocking factor=floor(512/154)= 3 entries per block.

No. of blocks required = ceiling(24,000/3)= 8000.

Case 1:
No ordering based on any field.
Linear search will take o(n/2) block accesses,
so approximately 8000/2 = 4000 block accesses.

Case 2:
Ordering based on custid,
so binary search would require ceiling(log(8000) to the base 2)= 13 block accesses.

Case 3:
Index based on custid,
Size of custid=20 bytes, block ptr size = 6 bytes.
So size of index entry=20+6=26 bytes.
So no. of index entries per block=floor(512/26)=19 entries.

Total no. of index entries = total no. of blocks in data file = 8000

so index blocking factor(no. of blocks required to store index entries,bfri)=
ceiling(8000/19)=422 entries

so now search will take ceiling(log(422) to base 2)+1=9+1=10 block accesses.

Case 4:
Multilevel index on goodsid,
1^{st} level index(bfri) = 422 entries
2^{nd} level index = ceiling(422/19) = 23 entries
3^{rd} level index = ceiling(23/19)=2 entries
4^{th} level index = ceiling(2/19)=1 entry.

So no. of block accesses = 4(level of indexing) + 1=5 block accesses.

Case 5:
Btree : Index on Customer id
Block pointer size P = 6 bytes
Record pointer size R = 7 bytes
Size of search key field = 20 bytes
Block size B = 512 bytes

Number of goods = 24000

A node in a B-Tree can have at most p pointers, p-1 entries and p-1 data pointers. Since all these need to fit within a block, we have :

6p + 20(p-1) +7(p-1) <= 512
p approximately equal to 15

Assuming that 69 percent of the nodes are full when the number of values in the tree stabilizes, we have p = 0.69p = 11

No. of levels = log(24000)/log(11) = 5 levels
If we don't consider the root level = 4

Level	Nodes	Entries	Pointers
Root	1	10	11
Level 1	11	110	121
Level 2	121	1210	1331
Level 3	1331	13310	14641
Level 4	14641	146410	161051

No. of entries and pointers barring the last level = 2793
Remaining entries = 24000-2793 = 21207

No. of nodes in the last level = Remaining entries and pointers/no. of entries and pointers in one node = 1009
Since size of one node <= size of a block, we will have to fit one node per block

Therefore number of blocks = 1009
Number of block accesses = level + 1 = 5
Summarizing.

Property	Block accesses	No. of extra blocks required
No ordering	4000	0
Ordering on customer id	14	0
Index on customer id	10	422
Multilevel index on customer id	5	422+23+2+1=448
B-Tree Index on customer id	5	1009+1+11+121=1142

We have decided to use Multilevel on Customer Id

Products Table

We have a products table with 36,00,000 records
Customer record size=154 bytes.116

Blocking factor=floor(512/116)= 4 entries per block.

No. of blocks required = ceiling(3600000/4)= 900000.

Case 1:
No ordering based on any field.
Linear search will take o(n/2) block accesses,
so approximately 900000/2 = 450000 block accesses.

Case 2:
Ordering based on pid,
so binary search would require ceiling(log(450000) to the base 2) =23 block accesses.

Case 3:
Index based on pid,
Size of pid=20 bytes, block ptr size = 6 bytes.
So size of index entry=20+6=26 bytes.
So no. of index entries per block=floor(512/26)=19 entries.

Total no. of index entries = total no. of blocks in data file = 3600000

so index blocking factor(no. of blocks required to store index entries,bfri)=
ceiling(3600000/19)=189474 entries

so now search will take ceiling(log(189474) to base 2)+1=18+1=19 block accesses.

Case 4:
Multilevel index on pid,
1^{st} level index(bfri) = 189474 entries
2^{nd} level index = ceiling(189474/19) = 9973entries
3^{rd} level index = ceiling(9973/19)=525 entries
4^{th} level index = ceiling(525/19)=28 entries.
5^{th} level index = ceiling(28/19)=2 entries
6^{th} level index = ceiling(2/19)=1 entry

So no. of block accesses = 6(level of indexing) + 1=7 block accesses.

Case 5:
Btree : Index on Product id
Block pointer size P = 6 bytes
Record pointer size R = 7 bytes
Size of search key field = 20 bytes
Block size B = 512 bytes

Number of goods = 3600000
A node in a B-Tree can have at most p pointers, p-1 entries and p-1 data pointers. Since all these need to fit within a block, we have :

6p + 20(p-1) +7(p-1) <= 512
p approximately equal to 15

Assuming that 69 percent of the nodes are full when the number of values in the tree stabilizes, we have p = 0.69p = 11

No. of levels = log(3600000)/log(11) = 7 levels
If we don't consider the root then levels = 6

Level	Nodes	Entries	Pointers
Root	1	10	11
Level 1	11	110	121
Level 2	121	1210	1331
Level 3	1331	13310	14641
Level 4	14641	146410	161051
Level 5	161051	1610510	1771561
Level 6	1771561	17715610	

No. of entries and pointers barring the last level = 338205
Remaining entries = 3600000-338205 = 3261795

No. of nodes in the last level = Remaining entries+pointers/no. of entries+pointers in one node = 155324
Since size of one node <= size of a block, we will have to fit one node per block

Therefore number of blocks = 155324
Number of block accesses = level + 1 = 7

Summarizing.

Property	Block Accesses	No. of extra blocks required
No ordering	450000	0
Ordering on pid	24	0
Index on pid	19	189474
Multilevel index on pid	7	189474+525+28+2+1=190030
Btree index on pid	7	155324+1+11+121+1331+14641=171429

Since the number of block accesses are lesser in the case of BTree index on pid as opposed to Multilevel index on pid, we have chosen this.

Books Table

We have 600000 records
Index entry = 4bytes + 6 bytes [from goodsid + blockptr]

Bfri = 512/10 = 51
So we need [600000/51] = 11765 blocks
So block access = log(11765)/log(2) + 1 = 15

Media Table

We have 600000 records
Index entry = 4bytes + 6 bytes [from goodsid + blockptr]

Bfri = 512/10 = 51
So we need [600000/51] = 11765 blocks
So block access = log(11765)/log(2) + 1 = 15

Fashion Table

We have 600000 records
Index entry = 4bytes + 6 bytes [from goodsid + blockptr]

Bfri = 512/10 = 51
So we need [600000/51] = 11765 blocks
So block access = log(11765)/log(2) + 1 = 15

TV Table

We have 600000 records
Index entry = 4bytes + 6 bytes [from goodsid + blockptr]

Bfri = 512/10 = 51
So we need [600000/51] = 11765 blocks
So block access = log(11765)/log(2) + 1 = 15

Laptop Table

We have 600000 records
Index entry = 4bytes + 6 bytes [from goodsid + blockptr]

Bfri = 512/10 = 51
So we need [600000/51] = 11765 blocks
So block access = log(11765)/log(2) + 1 = 15

Mobiles Table

We have 600000 records
Index entry = 4bytes + 6 bytes [from goodsid + blockptr]

Bfri = 512/10 = 51
So we need [600000/51] = 11765 blocks
So block access = log(11765)/log(2) + 1 = 15

QUERY EXECUTION

Assuming 1 Block access time=24.11ms

1)select * from goods where goodsid="x"or brand="x"

This is equivalent to saying the no. of block accesses for

(select * from goods where goodsid="x")

+

(select * from goods where brand="x")

- since we have a multilevel index on goodsid on goodsid in goods,first query will take 5 block accesses
- since we have a multilevel clustered index on brand in goodsid ,second query will take 4 block accesses

so the whole query will take 9 block accesses.

So avg. query execution time=9*24.11=217ms

2)select * from products join goods on products.goodsid=goods.goodsid where goodsid="x";

- since we have a multilevel index on goodsid in goods table(5),
- since we have a btree index on pid in products table(7),

the query will take 5*7 block accesses=35 block accesses

So avg. query execution time=35*24.11=843.85ms

3)select * from seller join products on seller.sid=products.sid where sid=x;

- since we have a multilevel index on sid in seller table(4 block accesses),
- since we have no index on sid in products table(log(3600000) to base 2 block accesses, as we have 3600000 records in products table),

the query will take 22*4 block accesses=88 block accesses

So avg. query execution time=88*24.11=2121ms

4)select * from products where pid="x";

- since we have a btree index on pid in products table we'll need 7 block accesses.

So avg. query execution time=7*24.11=169ms

5)select * from customers where customer="x"

- since we have a multilevel index on custid in customers table we'll need 5 block accesses.

So avg. query execution time=5*24.11=121ms