1. Consider two relations R and S with the following information about them:
   Relation R consists of 2,000 pages with 10 tuples per page, and relation S consists of 500 pages with 50 tuples per page. If S has a B+ tree index on the join attribute, what is the I/O cost of performing an index nested loops join? Explain your answer.

R (join) S
In index nested loop join, inner table should have an index. So the S table has to be the inner table.
So the I/O cost = no pages in R + (B+ tree's top to bottom travel cost) * no of tuples in R
= 2000 + 2 * (2000 * 10) I/O
Here we assume the height of the B+ tree is 2.

2. Consider the join of two relations R and S on attributes R.a and S.b. R contains 10,000 tuples and has 10 tuples per page. S contains 2,000 tuples also with 10 tuples per page. There are 52 buffer pages available. Estimate the minimum number of page I/Os for performing a Block Nested Loop join of R and S. Ignore the I/O cost of writing out the result. Explain your steps clearly.

Minimum I/O comes when we choose the table which has the minimum no of pages as the outer table.

So S table is the outer table.

I/O cost = no of pages in S + no of blocks in S * no of pages in R table
= 2000/10 + ((2000/10) / (52-2)) * 2000/10 I/0

3. Consider the join of two relations R and S on attributes R.a and S.b. R contains 20,000 tuples and has 10 tuples per page. S contains 1,000 tuples also with 10 tuples per page. Assume that there are clustered B+ tree indexes on R.a and S.b. Estimate the minimum number of page I/Os for performing a Sort-Merge join of R and S. Ignore the I/O cost of writing out the result. Explain your steps clearly.

Here the R and S table has clustered B+ indexes. Which means they are already sorted.
So I/O cost = cost to sort R + cost to sort S + cost to merge R and S
= 0 + 0 + 20000/10 + 1000/10

4. The relation Executives has attributes ename, title, dname, and address; all are string fields of the same length. The ename attribute is a candidate key. The relation contains 10,000 pages. There are 10 buffer pages. Consider the query:

```
SELECT E.Ename
FROM Executives E
WHERE E.title = 'CFO' and E.dname = 'Toy';
```

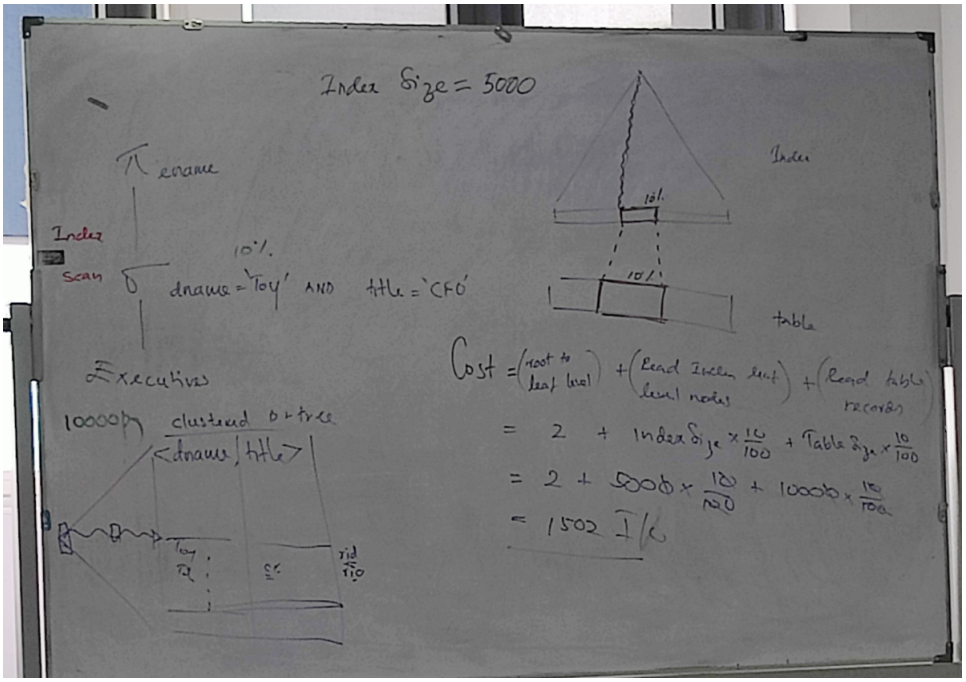
   Assume that only 10% of Executives tuples meet the selection conditions. If a clustered B+ tree index on <dname, title> is (the only index) available, what is the cost of the best plan for this query? Clearly describe how you arrived at the answer.

Here selection values and index have the same columns. So there is no need to access the Table to do the selection, instead we can directly get the selection records from the B+ index. We know values in the B+ tree, reside at the child level. So we have to go from top to bottom in B+ tree. Assuming the height of B+ tree is 2, the cost to travel from top to bottom is 2. Also

we know exactly where dname and title values are in the child node. So we only have to retrieve those from the child node. We know only 10% records are available in the given selection condition. So only have to retrieve 10% of records in the index.we know this table is 10000 pages in size with ename, title, and address columns. But in B+ tree we only have ename and title which means 10000/2 = 5000 pages is the size of the B+ tree. We only retrieve 10% from the B+ tree which means 5000 * 10/100 = 500 pages. From these 500 pages we can get the recordId, dname, title. This query needs to print Ename. But we don't have it in the index. So now we will have to go to the table and get it. But with the recordIds we retrieved from the index we can find where the values are in the table. It says after the selection only 10% records meet the condition. We have 10000 pages in the table. So we only retrieve 1000 pages. (Here we assume this is an alternative 2 B+ tree. That is why we retrieve from Table.)

So the total cost = B+ tree travel from top to leaf nodes + retrieve records from index + retrieve records from table

= 2 + 500 + 1000 = 1502 I/O



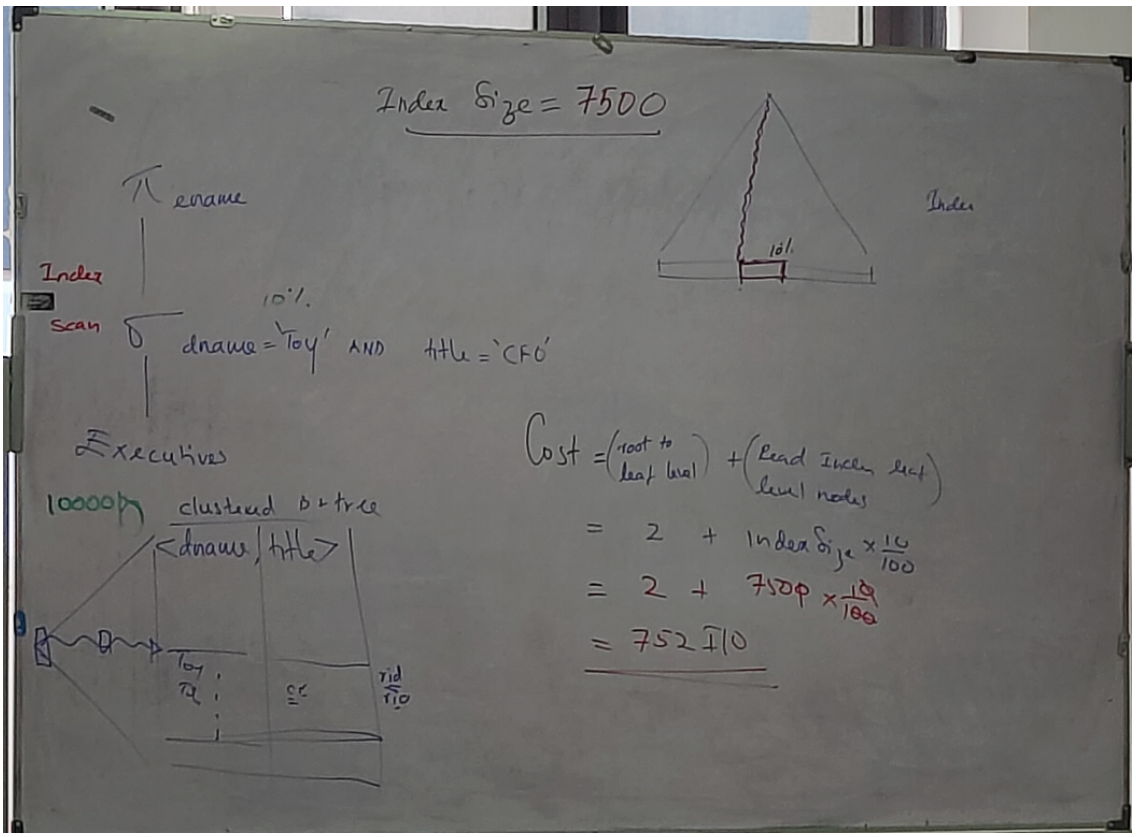
6. The relation Executives has attributes ename, title, dname, and address; all are string fields of the same length. The ename attribute is a candidate key. The relation contains 10,000 pages. There are 10 buffer pages. Consider the query:

```
SELECT E.Ename
FROM Executives E
WHERE E.title = 'CFO' and E.dname = 'Toy';
```

Assume that only 10% of Executives tuples meet the selection conditions. If a clustered B+ tree index on <dname, title, ename> is (the only index) available, what is the cost of the best plan for this query? Clearly describe how you arrived at the answer.

Same as above but, now the index size is different since we have 3 columns now. So the index size is 10000 * ¾ = 7500 pages. From these we only retrieve 10% pages as above which is 750 pages. But unlike above, we don't have to access tables here since the ename column is available in the index. So the total cost is,

2 + 750 = 752 I/O



7. The relation Emp(empno: string, name: string, age: integer, salary: real) contains 4,000 pages. There are 10 buffer pages. The attribute empno takes up 10 bytes, name 20 bytes, age 2 bytes and salary 8 bytes. A clustered B+ tree on <age,salary> is the only index available. In estimating the cost of query plans, ignore the cost of writing the final result. Consider the following query:

   Select age, avg(salary)
   From emp
   Where salary>40000
   Group by age;

   Assume that 10% of the tuples satisfy the selection condition.
   (a) Describe the best plan and show an estimation of its cost.
   (b) If this is the most important of all queries on the Emp table, what additional index if any would you implement to speed up this query? Justify your answer.

<age, salary> index size = 4000 pages * (2 bytes + 8 bytes) / 40 bytes = 1000 pages.

a. Here in the index first column is age. For example 14 y/o salary can be 8000. 15 y/o salary can be 5000, 16 y/o salary can be 3000. Likewise, age is sorted. But the salary is not. In the query where clause we have to identify salaries which are more than 40000. But index is not helpful because salaries are not sorted so we have to read all the pages in the index to find out salaries which are more than 40000. Now after getting those salaries we can group them by age. When grouping them by age, we have sort the age. But here age is already sorted. So no need to do that. AVG is happening inside memory so no need to worry.

So the total cost is 1000 pages. (which is for searching salaries in the index and that's all

b. We can make the index <salary, age>. Now we don't have to go through whole B+ tree since the salary is sorted. Now we can go directly from top to bottom in the B+ tree. Cost for that is 2 (assuming B+ tree's height is 2). It says only 10% of tuples will satisfy the condition. We know the index size is 1000 pages. 10% of that is 100 pages. We don't have to go to the able since the index has both salary and age.When grouping them by age, we have to sort the age. Now the age is not sorted so we have to sort it. We have 100 pages and 10 buffer pages to sort is,