Exercise 1

T(R) = 10000, B(R) = 500, V(R,A) =30. Calculate
a) the number of tuples expected in the result, and
b) the IOs it will take to retrieve the result, for the query $\sigma_{A=10}(R)$ in the following cases :

1. R has a clustering index on A.
2. R has a non-clustering index on A
3. R has no index on A but is a clustered/contiguous relation.
4. R is non-contiguous.

Exercise 1a

T(R) = 50,000, B(R) = 2,500, V(R, B) = 50. Calculate
a) the number of tuples expected in the result, and
b) the I/Os required to retrieve the result for the query **σB=20(R)** in the following cases:

1. R has a primary **B+ tree index** on B.
2. R has a **secondary index** (non-clustering) on B.
3. R has **no index** on B but is stored **sequentially**.
4. R is **randomly scattered** across disk pages.

Exercise 1b

T(R) = 20,000, B(R) = 1,000, V(R, C) = 25. Calculate
a) the number of tuples expected in the result, and
b) the I/Os required to retrieve the result for the query **σC=5(R)** in the following cases:

1. R has a **clustering index** on C.
2. R has a **non-clustering index** on C.
3. R has **no index** on C but is **sorted** on another attribute.
4. R is stored **in an unordered fashion**.

Exercise 1c

T(R) = 200,000, B(R) = 8,000, V(R, D) = 40. Calculate
a) the number of tuples expected in the result, and
b) the I/Os required to retrieve the result for the query **σD=15(R)** in the following cases:

1. R has a **clustered B-tree index** on D.
2. R has a **hash-based index** on D.
3. R has **no index** on D but is **clustered** in storage.
4. R is **fragmented across multiple disks**.

Exercise 2

$T(R) = 10000$, $B(R) = 400$, $V(R,A) = 50$, $V(R,B)=30$, $V(R,C)=100$. R has a clustering index on A and non-clustering index on B.

Calculate the cost in terms of a) the number of tuples expected in the result and b) the IOs it will take to retrieve the result in the following cases (assume index to be in memory) :

1. $\sigma_{A<10}(R)$
2. $\sigma_{B=20}(R)$
3. $\sigma_{C=40}(R)$

What would be a cost effective plan to execute this query:

4. $\sigma_{A=10 \text{ AND } B=4 \text{ AND } C=6}(R)$

Exercise 2a

T(R) = 30,000, B(R) = 600, V(R, X) = 60, V(R, Y) = 40, V(R, Z) = 80.
R has a **clustering index on X** and a **non-clustering index on Y**.

Calculate the cost in terms of
a) the number of tuples expected in the result, and
b) the I/Os it will take to retrieve the result in the following cases (assume index is in memory):

1. **σX>15(R)**
2. **σY=25(R)**
3. **σZ=50(R)**

What would be a cost-effective plan to execute this query:
4. **σX=10 AND Y=8 AND Z=20(R)**

Exercise 2b

T(R) = 50,000, B(R) = 1,000, V(R, P) = 100, V(R, Q) = 50, V(R, R) = 70.
R has a **B+ tree index on P** and a **hash index on Q**.

Calculate the cost in terms of
a) the number of tuples expected in the result, and
b) the I/Os it will take to retrieve the result in the following cases (assume index is in memory):

1. **σP<30(R)**
2. **σQ=12(R)**
3. **σR=40(R)**

What would be a cost-effective plan to execute this query:
4. **σP=5 AND Q=9 AND R=25(R)**

Exercise 2c

T(R) = 20,000, B(R) = 500, V(R, M) = 90, V(R, N) = 60, V(R, O) = 120.
R has a **clustering index on M** and a **secondary index on N**.

Calculate the cost in terms of
a) the number of tuples expected in the result, and
b) the I/Os it will take to retrieve the result in the following cases (assume index is in memory):

1. **$\sigma M>50(R)$**
2. **$\sigma N=30(R)$**
3. **$\sigma O=80(R)$**

What would be a cost-effective plan to execute this query:
4. **$\sigma M=20$ AND $N=10$ AND $O=50(R)$**

Exercise 3

This question is created from the DSCB example mentioned earlier.

Suppose B(R) = 1000, and T(R) = 20,000.

R is clustered and A is one of the attributes of R, and there is a non-clustering-index on A. What would be a good way to execute the query $\sigma_{A=0}(R)$.

1. Index probe costs 2 IO per probe
2. Index is in memory

Exercise 3a

This question is based on a DSCB example.
Suppose **B(R) = 800** and **T(R) = 25,000**.
R is clustered, and **B is one of the attributes of R**. There is a **non-clustering index on B**.
What would be a good way to execute the query **$\sigma B=15(R)$**?

1. Index probe costs **2 IO per probe**.
2. Index is **in memory**.

Exercise 3b

This question is based on a DSCB example.
Suppose **B(R) = 500** and **T(R) = 30,000**.
R is **not clustered**, and **C is one of the attributes of R**. There is a **secondary index on C**.
What would be a good way to execute the query **$\sigma C>20(R)$**?

1. Index probe costs **3 IO per probe**.
2. Index is **in memory**.

Exercise 3c

This question is based on a DSCB example.
Suppose **B(R) = 1,200** and **T(R) = 15,000**.
R is **clustered**, and **D is one of the attributes of R**. There is a **non-clustering index on D**.
What would be a good way to execute the query **σD=5(R)**?

1. Index probe costs **1 IO per probe**.
2. Index is **in memory**.

Exercise 4

Relation R : B(R) = 10,000
Relation S : B(S) = 4000

Given that R and S are sorted and contain no duplicates for
join attribute, what Join would you suggest. What would be the

---

Question 4a: Join Selection for Sorted Relations

Relation R: B(R)=15,000 *B*(*R*)=15,000. Relation S: B(S)=5,000 *B*(*S*)=5,000
Given that R and S are sorted and contain no duplicates for the join attribute, what join algorithm would you suggest? What would be the total number of I/O operations required?

Question 4b: Join Selection for Large Relations

Relation R: B(R)=50,000 *B*(*R*)=50,000. Relation S: B(S)=20,000 *B*(*S*)=20,000
Given that R and S are sorted and contain no duplicates for the join attribute, what join algorithm would you suggest? What would be the total number of I/O operations required?

Question 4c: Join Selection for Small Relations

Relation R: B(R)=1,000*B*(*R*)=1,000
Relation S: B(S)=500*B*(*S*)=500
Given that R and S are sorted and contain no duplicates for the join attribute, what join algorithm would you suggest? What would be the total number of I/O operations required?

Explanation of Concepts in the Questions

- **Sorted Relations**: When both relations are sorted on the join attribute, a **sort-merge join** is often the most efficient choice.
- **No Duplicates**: The absence of duplicates simplifies the join process, as there is no need to handle multiple matches.
- **I/O Operations**: The total number of I/O operations depends on the size of the relations and the join algorithm used.