Question 1 (5 marks)
Consider two relations A and B. A has 80,000 tuples, and B has 100,000 tuples. Both relations store 100 tuples per page. Consider the following SQL statement:

```
SELECT *
FROM A INNER JOIN B
ON A.a = B.a;
```

We wish to evaluate an equijoin between A and B, with an equality condition A.a = B.a. There are 102 buffer pages available for this operation. Both relations are stored as (unsorted) heap files. Neither relation has any indexes built on it.
Consider the alternative join strategies described below and calculate the cost of each alternative. Evaluate the algorithms using the number of disk I/O's (i.e., pages) as the cost. For each strategy, provide the formulae you use to calculate your cost estimates.

a) Page-oriented Nested Loops Join. Consider A as the outer relation. (1 mark)

Page-oriented nested loops join is to do a page by page scan of the outer relation and for each outer page, do a page-by-page scan of the inner relation.
In this algorithm we don't use multiple buffers at a time, so the minimum requirement is one input buffer to page through each relation and one output buffer to store the output. Hence in total 3 buffer pages are required.

NPages (A) = 80,000 / 100
= 800
NPages (B) = 100,000 / 100
= 1000

Total cost = (Number of pages in relation A) + (Number of pages in relation A * Number of pages in relation B)
= NPages (A) + (NPages (A) * NPages (B))
= 800 + (800 * 1000)
= 800,800 I/O

b) Block-oriented Nested Loops Join. Consider A as the outer relation. (1 mark)

In block nested loops join, the outer relation is read in blocks (groups of pages that will fit into whatever buffer pages are available), and, for each block, do a page-by-page scan of the inner relation. The outer relation is scanned once, and the inner relation is scanned only once for the outer block.

If we have fewer buffers available, the cost will increase as the number of blocks will vary. The minimum number of buffer pages is 102 for this cost.

NPages (A) = 80,000 / 100
= 800

NPages (B) = 100,000 / 100
= 1000

Number of blocks = (Number of pages in relation A) / (Number of buffer pages – 2)
= Npages (A) / (Number of buffer pages – 2)
= 800 / (102 – 2)
= 8

Total cost = (Number of pages in relation A) + (Number of blocks in relation A * Number of pages in B)
= 800 + (8 * 1000)
= 8,800 I/O

c) Sort-Merge Join. Assume that Sort-Merge Join can be done in 2 passes. (1 mark)

We will use external merge sort for this case. The initial sorting pass will split B into 10 runs of 100 buffer pages. After that, these pages will be written to the disk. These sorted pages will be read again to be compared across runs and combine the results. The resulting sorted pages will be written to disk. Similarly, A will split into 8 runs of approximately 100 buffer pages and follow the same process as B.

Cost of sorting A = 2 * Number of passes * Number of pages of A
= 2 * Number of passes * NPages (A)
= 2 * 2 * 800
= 3,200

Cost of sorting B = 2 * Number of passes * Number of pages of B
= 2 * Number of passes * NPages (B)
= 2 * 2 * 1000
= 4,000

Cost of merging A and B = Number of pages read of A + Number of pages read of B
= NPages (A) + NPages (B)
= 800 + 1,000

= 1,800

Total cost = Cost of sorting A + Cost of sorting B + Cost of merging A and B
= 3200 + 4000 + 1800
= 9000 I/O

d) Hash Join (1 mark)

In hash join, each relation is partitioned and then the join is performed by "matching" elements from corresponding partitions.

Total cost = 2 * Number of pages of relation A + 2 * Number of pages of relation B + Number of pages of relation A + Number of pages of relation B
= 2 * NPages (A) + 2 * NPages (B) + NPages (A) + NPages (B)
= 3 * (NPages (A) + NPages (B))
= 3 * (800 + 1,000)
= 5,400 I/O

e) What would be the lowest possible cost to perform this query, assuming that no indexes are built on any of the two relations, and assuming that sufficient buffer space is available? What would be the minimum buffer size required to achieve this cost? Explain briefly. (1 mark)

The optimal cost would be achieved if each relation was read only once. We could do such a join by storing the entire smaller relation in memory, reading in the larger relation page by page and for each tuple in the larger relation we search the smaller relation (which exists entirely in memory) for matching tuples. The buffer pool would have to hold the entire smaller relation, one page for reading in the larger relation and one page to serve as an output buffer.

Let M be the number of pages of relation A
M = 80,000 / 100
= 800

Let N be the number of pages of relation B
N = 100,000 / 100
= 1000

Total cost = Number of pages of relation A + Number of pages of relation B
= 800 + 1000
= 1800 I/O

The minimum number of buffer pages for this cost is min{M, N} + 1 + 1 = 800 + 1 + 1
= 802

Question 2 (5 marks)
Consider a relation with the following schema:
JobSeekers(id, firstname, lastname, city, soughtsalary)

The JobSeekers relation consists of 10,000 pages. Each page stores 100 tuples. The online software works in the 8 largest Australian cities and soughtsalary can have values between 60,000 and 160,000 (i.e., [60,000 – 160,000].)

Suppose that the following SQL query is executed frequently using the given relation:

SELECT * FROM JobSeekers WHERE city = 'Melbourne' AND soughtsalary > 80,000;

Your job is to:
a) Compute the reduction factors and the estimated result size in number of tuples. (1 mark)
NTuples (JobSeekers) = 10,000 * 100
= 1,000,000 tuples

RF (city) = 1/ (Number of cities)
= 1/8
= 0.125

RF (soughtsalary > 80,000) = (High(I) – value) / (High(I) − Low(I))
= (160,000 – 80, 000) / (160, 000 – 60, 000)
= 4/5
=0.8

Result size = NTuples (JobSeekers) * RF (city) * RF (soughtsalary > 80,000)
= 1,000,000 * 0.125 * 0.8
= 100,000 tuples

b) Compute the estimated cost in number of disk I/O's of the best plan if a clustered B+ tree index on (city, soughtsalary) is the only index available. Suppose there are 2,000 index pages. Discuss and calculate alternative plans. (1 mark)

Cost = (Number of pages of the index + Number of pages of the relation) * Reduction factors
= (NPages (I) + NPages (R)) * RF (city) * RF (soughtsalary > 80,000)
= (2,000 + 10,000) * 0.125 * 0.8
= 1,200 I/O

We can also perform a full table scan, (doing a heap scan) as an alternative plan.
Cost = Number of pages of the relation
= NPages (JobSeekers)
= 10,000 I/O

Hence the clustered B+ tree index with cost 1,200 I/O is the cheapest access path.

c) Compute the estimated cost in number of disk I/O's of the best plan if an unclustered B+ tree index on (soughtsalary) is the only index available. Suppose there are 2,000 index pages. Discuss and calculate alternative plans. (1 mark)

Cost = (Number of pages of the index + Number of tuples of the relation) * Reduction factor
= (NPages (I) + NTuples (R)) * RF (soughtsalary > 80,000)
= (2,000 + 1,000,000) * 0.8
= 801,600 I/O

We can also perform a full table scan, (doing a heap scan) as an alternative plan.
Cost = Number of pages of the relation
= NPages (JobSeekers)
= 10,000 I/O

Hence the cheapest access path is the full table scan, with cost 10,000 I/O.

d) Compute the estimated cost in number of disk I/O's of the best plan if an unclustered Hash index on (city) is the only index available. Discuss and calculate alternative plans. (1 mark)
Cost = Number of tuples of the relation * Reduction factor * 2.2
= NTuples (R) * RF (city) * 2.2
= 1,000,000 * 0.125 * 2.2
= 275,000 I/O

We can also perform a full table scan, (doing a heap scan) as an alternative plan.
Cost = Number of pages of the relation
= NPages (JobSeekers)
= 10,000 I/O

Hence the cheapest access path is the full table scan, with cost 10,000 I/O.

e) Compute the estimated cost in number of disk I/O's of the best plan if an unclustered Hash index on (soughtsalary) is the only index available. Discuss and calculate alternative plans. (1 mark)

Cost = Number of tuples of the relation * Reduction factor * 2.2
= NTuples (R) * RF (soughtsalary > 80,000) * 2.2
= 1,000,000 * 0.8 * 2.2
= 1,760,000 I/O

We can also perform a full table scan, (doing a heap scan) as an alternative plan.
Cost = Number of pages of the relation
= NPages (JobSeekers)
= 10,000 I/O

Hence the cheapest access path is the full table scan, with cost 10,000 I/O.

Question 3 (10 marks)
Consider the following relational schema and SQL query. The schema captures information about employees, departments, and company finances (organized on a per department basis).

Emp(eid: integer, did: integer, sal: integer, hobby: char(20))
Dept(did: integer, dname: char(20), floor: integer, phone: char(10))
Finance(did: integer, budget: real, sales: real, expenses: real)

Consider the following query:

```
SELECT D.dname, F.budget
FROM Emp E, Dept D, Finance F
WHERE E.did = D.did
    AND D.did = F.did
    AND E.sal > 100,000
    AND E.hobby IN ('diving', 'soccer');
```

The system's statistics indicate that employee salaries range from 50,000 to 150,000, and employees enjoy 50 different hobbies. There is a total of 25,000 employees and 1,200 departments (each with corresponding financial record in the Finance relation) in the database. Each relation fits 100 tuples in a page. Suppose there exists a clustered B+ tree index on (Dept.did) and a clustered B+ tree index on (Emp.salary), both of size 50 pages.

a) Compute the reduction factors and the estimated result size in number of tuples. (2 marks)

The number tuples in Finance relation = Number of financial records
= Number of departments
= 1,200
Maximum number of tuples in the result = Number of employees * Number of departments * Number tuples in Finance relation
= 25,000 * 1,200 * 1,200
= 36,000,000,000

RF (hobby) = 2 / (Total number of hobbies)
= 2/50
= 0.04

RF (sal > 100,000) = (High(I) – value) / (High(I) − Low(I))
= (150,000 – 100, 000) / (150, 000 – 50, 000)
= 1/2
=0.5

NTuples (employees) = 25,000
NTuples (departments) = 1,200
NTuples (finance record) = 1,200

Since department ID is a primary key in department relation,
Max(NKeys(did)) = NTuples (departments)
= 1,200

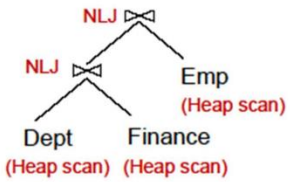
RF (E.did = D.did) = 1/Max(NKeys(did))
= 1/1,200

RF (D.did = F.did) = 1/Max(NKeys(did))
= 1/1,200

Result size = Maximum number of tuples in the result * RF (hobby) * RF (sal)
= 36,000,000,000 * 0.04 * 0.5 * 1/1,200 * 1/1,200
= 500 tuples

b) Compute the cost in number of disk I/O's of the plans shown below. Assume that sorting of any relation (if required) can be done in 2 passes. NLJ is a Page-oriented Nested Loops Join. Assume that did is the candidate key, and that 50 tuples of a resulting join between Emp and Dept fit in a page. Similarly, 50 tuples of a resulting join between Finance and Dept fit in a page. Any selections/projections not indicated on the plan are performed "on the fly" after all joins have been completed. (8 marks, 2 marks per plan)

1)



1) This left-deep plan is joining Dept with Finance using Page-oriented Nested Loop Join and then joining the results with Emp also using Page-oriented Nested Loop Join.
The cost analysis is shown below:

Cost to join Dept with Finance
= NPages(Dept) + NPages(Dept) * NPages(Finance)
= 1,200/100 + 1,200/100 * 1,200/100
= 12 + 12 * 12
= 156 I/O

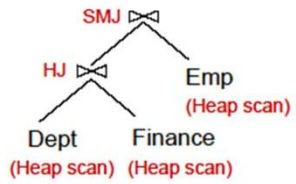
Number of resulting tuples for Dept JOIN Finance
= 1/ NKeys(did) * NTuples(Dept) * NTuples(Finance)
= 1 / 1,200 * 1,200 * 1,200
= 1,200 tuples

Number of pages for Dept JOIN Finance
= 1,200 / 50
= 24 pages

Cost to join with Emp = NPages(Dept JOIN Finance) * NPages(Emp)
= 24 * 25,000/100
= 24 * 250
= 6,000 I/O

Total cost = Cost to join Dept with Finance + Cost to join with Emp
=156 + 6,000
= 6,156 I/O

2)



2) This left-deep plan is joining Dept with Finance using Hash Join and then joining the results with Emp using Sort-Merge Join. The number of passes of Sort-Merge Join is 2.
The cost analysis is shown below:

Cost to join Dept with Finance
= 3 * NPages(Dept) + 3 * NPages(Finance)
= 3 * (1,200/100) + 3 * (1,200/100)
= 3* 12 + 3 * 12
= 72 I/O

Number of resulting tuples for Dept JOIN Finance
= 1/ NKeys(did) * NTuples(Dept) * NTuples(Finance)

= 1 / 1,200 * 1,200 * 1,200
= 1,200 tuples

Number of pages for Dept JOIN Finance
= 1,200 / 50
= 24 pages

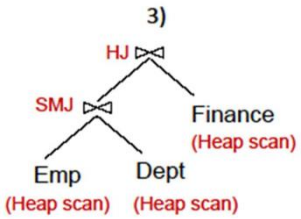
Cost of sorting Dept JOIN Finance = 2 * NPasses * NPages(Dept JOIN Finance)
= 2 * 2 * 24
= 96 I/O

Cost of sorting Emp = 2 * NPasses * NPages(Emp)
= 2 * 2 * 25,000/100
= 2 * 2 * 250
= 1,000 I/O

Cost of joining sorted Dept JOIN Finance and Emp
= NPages (Dept JOIN Finance) + NPages(Emp)
= 24 + 25,000/100
= 24 + 250
= 274 I/O

Cost to join with Emp = 96 + 1,000 + 274
= 1,370 I/O

Total cost = Cost to join Dept with Finance + Cost to join with Emp
= 72 + 1,370
= 1,442 I/O



3) This left-deep plan is joining Dept with Emp using Sort-Merge Join and then joining the results with Finance using Hash Join.

The number of passes of Sort-Merge Join is 2.
The cost analysis is shown below:

Number of resulting tuples for Dept JOIN Emp
= 1 / NKeys(did) * NTuples(Dept) * NTuples(Emp)
= 1 / 1,200 * 1,200 * 25,000
= 25,000 tuples

Number of pages for Dept JOIN Emp = 25,000 / 50
= 500 pages

Cost of sorting Dept = 2 * NPasses * NPages(Dept)
= 2 * 2 * 1,200/100
= 2 * 2 * 12
= 48 I/O

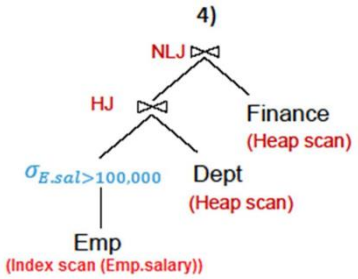
Cost of sorting Emp = 2 * NPasses * NPages(Emp)
= 2 * 2 * 25,000/100
= 2 * 2 * 250
= 1,000 I/O

Cost of joining sorted Dept and Emp = NPages(Dept) + NPages(Emp)
= 1,200/100 + 25,000/100
= 12 + 250
= 262 I/O

Total cost of SMJ between Dept and Emp = 48 + 1,000 + 262
= 1,310 I/O

Cost to join with Finance = 2 * NPages(Dept JOIN Emp) + 3 * NPages(Finance)
= 2 * 500 + 3 * (1,200 / 100)
= 1,000 + 36
= 1,036 I/O

Total cost = 1,310 + 1,036
= 2,346 I/O



4) This left-deep plan is joining Dept with Employee who has salary greater than 100,000 using Hash Join and then joining the results with Finance using Page-oriented Nested Loop Join. The cost analysis is shown below:

RF (salary > 100,000) = (High(I) – value) / (High(I) − Low(I))
= (150,000 – 100, 000) / (150, 000 – 50, 000)
= 0.5

Number of tuples selected = NTuples (employee) * RF (salary > 100,000)
= 25,000 * 0.5
= 12,500 tuples

Number of pages selected = 12,500 / 100
= 125 pages

Cost of selection = (NPages(index) + NPages(employee)) * RF
= (50 + 25,000/100) * 0.5
= (50 + 250) * 0.5
= 150 I/O

We replaced the reading of $\boldsymbol{\sigma}_{\text{Emp.sal} > 100,000}$ (Emp) with the Index scan of Emp, so we pipeline from the Index Scan to sorting.
Cost to join Dept with $\boldsymbol{\sigma}_{\text{Emp.sal} > 100,000}$ (Emp)
= 3 * NPages(Dept) + 2 * NPages($\boldsymbol{\sigma}_{\text{Emp.sal} > 100,000}$ (Emp))
= 3 * (1,200/100) + 2 * 125
= 3* 12 + 2 * 125
= 286 I/O

Number of resulting tuples for $\boldsymbol{\sigma}_{\text{Emp.sal} > 100,000}$ (Emp) JOIN Dept
= 1 / NKeys(did) * NTuples(Dept) * NTuples($\boldsymbol{\sigma}_{\text{Emp.sal} > 100,000}$ (Emp))
= 1 / 1,200 * 1,200 * 12,500
= 12,500 tuples

Number of pages for $\sigma_{Emp.sal > 100,000}$ (Emp) JOIN Dept = 12,500 / 50
= 250 pages

Cost to join with Finance = NPages($\sigma_{Emp.sal > 100,000}$ (Emp) JOIN Dept) * NPages(Finance)
= 250 * 1,200/100
= 250 * 12
= 3,000 I/O

Total cost = Cost of selection + Cost to join Dept with $_{Emp.sal > 100,000}$ (Emp) + Cost to join with Finance
= 150 + 286 + 3,000
= 3,436 I/O