**Problem 1**

**A1: 10 points**
**5 points for (i)**
We aim to find suppliers who are the only suppliers of some part - in other words, they provide at least one part that no other supplier provides. One approach was to execute a self-join on the part number, ensuring a different supplier.

```
SELECT SupplierNo
FROM warehouse AS w1
WHERE NOT EXISTS
    (SELECT * -- a second SupplierNo for w1.PartNo
     FROM warehouse w2
     WHERE w1.PartNo = w2.PartNo
       AND w1.SupplierNo <> w2.SupplierNo)
```

Another potential solution for (i) was grouping by PartNo and counting:

```
SELECT SupplierNo
FROM warehouse w,
  (SELECT PartNo
   FROM warehouse
   GROUP BY PartNo HAVING COUNT(*) = 1) AS oneSupplierPart
WHERE ON oneSupplierPart.PartNo = w.PartNo
```

Other similar solutions used EXCEPT instead of NOT EXISTS, PartNo IN <subquery> instead of a join on PartNo, etc.
Finally, some students opted to negate condition (i) and INTERSECT it with the negated condition (ii) - this is correct (and thus acceptable) for (i).

**5 points for (ii)**
We aim to identify suppliers that sell at least two parts at the minimum price (for those parts). One solution is as follows:

```
SELECT SupplierNo
FROM
  (SELECT PartNo, min(Price) AS mPrice
   FROM warehouse
   GROUP BY PartNo) AS partMinPrice
JOIN warehouse AS w ON w.PartNo = partMinPrice.PartNo
AND w.Price = partMinPrice.mPrice
GROUP BY w.SupplierNo HAVING COUNT(*) >= 2
```

Another similar solution is to replace the join above with a subquery in the WHERE clause:

```
SELECT SupplierNo
FROM warehouse w1
```

```
WHERE w1.Price =
     (SELECT MIN(Price)
      FROM warehouse w2
      WHERE w1.partNo = w2.partNo)
GROUP BY w1.SupplierNo HAVING COUNT(*) >= 2
```

**A2: 10 points**
**5 points for (i)**
This is identical to A1

**5 points for (ii)**
The strict minimum check can be accomplished a few ways. One is to ensure that only one tuple exists for the given part number and price, eg adding the following to the WHERE clause of the previous A1(ii) solutions:

```
[SELECT … FROM warehouse w1 WHERE … AND]
1 = (SELECT COUNT(*)
FROM warehouse AS w2
WHERE w1.PartNo = w2.PartNo
  AND w1.Price = w2.Price)
```

Similarly, you could use not exists and exclude the current supplier:

```
NOT EXISTS
  (SELECT *
   FROM warehouse AS w2
   WHERE w1.PartNo = w2.PartNo
     AND w1.Price = w2.Price
     AND w1.SupplierNo <> w2.PartNo)
 w1.Price = w2.Price AND w.SupplierNo <> w2.PartNo)
```

**A3: 10 points**

```
DELETE FROM warehouse WHERE SupplierNo IN (<query from A2>)
```

The above was by far the most common (and expected) solution. Other potential solutions do exist, eg:

```
DELETE FROM warehouse
WHERE EXISTS (<query from A2> AND A2.SupplierNo = SupplierNo)
```

An answer similar to “Replace A2 ‘SELECT SupplierNo’ with DELETE” was acceptable under certain restrictions. Primarily, A2 must not contain joins at the top-level and must query from the warehouse relation.

(No points taken off for referencing/reusing an incorrect answer to A2)

**A4: 10 points**
**2 points for "yes" or attempting to provide a RA query, 8 points for the RA query (4 points each for (i), (ii))**

Yes, the query is expressible. The below is one example of building the full query. Note that I abbreviate: $\rho_{W(SNo,Pno,Pr)}(warehouse)$

(i):

1. First we find parts that are supplied by more than one supplier:
   $\pi_{A.PNo}(\rho_A(W) \bowtie_{(A.PNo=B.PNo \wedge A.SNo<>B.SNo)} \rho_B(W))$
   I refer to this query as RA1.
2. Next we find parts with only one supplier. Note that parts with no suppliers are not represented in the warehouse table:
   $\pi_{PNo}(W) - RA1$
   I refer to this query as RA2
3. Now that we have all parts supplied by only one supplier, we can join against the warehouse table to find the corresponding supplier. This yields the final result for (i) (all suppliers that are the only supplier of some part):
   $\pi_{W.SNo}(RA2 \bowtie_{(W.PNo=RA2.PNo)} W)$
   I refer to this query as A4I.

(ii):

1. First, we find all supplier+part combinations that cannot be the strict minimum. This means all supplier+part combinations where another supplier exists for the same part at cheaper or equal price:
   $\pi_{A.Sno,A.Pno}(\rho_A(W) \bowtie_{(A.PNo=B.PNo \wedge A.SNo<>B.Sno \wedge A.Pr>=B.Pr)} \rho_B(W))$
   I refer to this query as RA4.
2. To find all supplier+part combinations that are guaranteed strict minimums, we can remove those that are not strict minimums:
   $\pi_{Sno,Pno}(W) - RA4$
   I refer to this query as RA5.
3. Now that we have supplier+part combinations corresponding strict minimum prices, we need to find suppliers with at least two such combinations. This requires a self-join on the result from above:
   $\pi_{A.SNo}(\rho_A(RA5) \bowtie_{(A.SNo=B.SNo \wedge A.Pno<>B.PNo)} \rho_B(RA5))$
   I refer to this query as A4II.

There are a few equivalent ways to combine the queries for (i) and (ii), eg:

- $\pi_{SNo}(W) - A4I - A4II$
- $\pi_{SNo}(W) - (A4I \cup A4II)$
- $(\pi_{SNo}(W) - A4I) \cap (\pi_{SNo}(W) - A4II)$

**Problem 2**

**A. (6')**
2048/50 = 40 records per block. (3')
10^6/40 = 25000 (3')

**B. (20')**
N=51 pointers (50 at the leaf level) Worst case: 25 at bottom level, 26 at other levels.

B+ tree, best case:
Leaf level: sparse index. One pointer per block: 25.000/50 = 500
Next Level: 10
The root:1

B+ tree, worst case:
Leaf nodes: 25.000/25 = 1000
Next Level: 1000/26 = 38.4 take the FLOOR (not the ceiling): 38
Next Level: 38/26=1.46 take the FLOOR: 1  Cannot split in two blocks. This is the root!

**C. (4')**
A sparse index can only be built on clustered data. Thus this is a primary index.

**D. (10')**
Non-Leaf: 1 + 1  (4')
Leaf: 2 (worst case)  (3')
Data Blocks: 21 (worst case)  (3')
Total: 25 (worst case)

**Problem 3**

**A. (10')**
We must count how many keys share the same prefix i, and then select the min i whereby all records with that prefix fit in one page.
In our case a page/bucket holds 3 records.
For i=1, six records starting with 0. They do not fit in one bucket.
For i=2, three records with prefix 01. They fit in one bucket. Also the three records with prefix 00 fit in one bucket. Then we have a record with prefix 10. Thus **3 buckets** are used. Moreover, since i=2. the hash table has **4 cells (directory/dictionary size)**.

**B. (10')**
Note we need to **distinguish between bucket split and overflow**. Overflow means the issue cannot be solved no matter how much times we do the bucket splitting. For example, two new records with duplicate key value 148 will end up in the same bucket, but a third such record will cause an overflow bucket. In fact, any three record insertions with their hashed value equal to the same will satisfy (plus this hashed value should equal to any of 106, 115, 148, 126, 16, 15, 31).

**Extra Credit**

**A. (3’)**
0, when the B values in R are not matched by the B values in S.

**B. (3’)**
n × m. The case when there only on B value in R and one in B and the two are the same.

**C. (3’)**
Maximum number of tuples: max(n × m, n + m) **(2’)**
Minimum number of tuples: max(m, n) **(1’)**