The answer is contributed by Yili Wang (with small modifications)

4.1 Statistics of the tables

1) Actual number of tuples: 10000 ; estimated number by the optimizer: 10064 ;
How do you find this information?

```
hw4=# explain analyze select * from partsupp;
                                    QUERY PLAN
--------------------------------------------------------------------------------------------------------
 Seq Scan on partsupp (cost=0.00..174.64 rows=10064 width=24) (actual
time=0.14..97.84 rows=10000 loops=1)
 Total runtime: 116.78 msec
(2 rows)
```

2)

Attribute Name	# of Distinct Values
ps_pskey	-1
ps_partkey	250
ps_suppkey	125
ps_availqty	250
ps_placed	365
ps_ship	367

```
SELECT attname, n_distinct
FROM pg_stats
WHERE tablename = 'partsupp';

or
SELECT attname, stadistinct
FROM pg_statistic, pg_attribute, pg_class
WHERE attnum=staattnum AND attrelid=oid AND
starelid=oid AND relname='orders';
```

-1 means all values are distinct

4.2 Index on perfect match query

1) Estimated total cost is 199.00 ;

The cost of a plan means

For each node in the tree, we must estimate the page I/O cost of performing the corresponding operation. Costs are also affected by whether pipelining is used or temporary relations are created to pass the output of an operator to its parent. Moreover, CPU power is considered in this estimation as well.

2) The estimated result cardinality is 39 ;

How does the query optimizer get this value?

The size of the result can be estimated by the maximum size allowed multiplied by the products of the reduction factors in the where clause. Since there is only one term in the where clause, column = value, the reduction factor for the term is $\frac{1}{250}$. In this case, the maximum size is 10000 tuples. Therefore, 10000/250 yields about 39 (assuming a uniform distribution).

Is it a reasonable value? YES

3) The access method is Sequential scan .

4) Order of the tuples returned by the plan:

The tuples would be returned by the row order in which they are in the table.

Create index.

5) The access method now is Index Scan .

6) Explanation

With sequential scan, we need to go through all the tuples in the table (since it is not sorted according to the column ps_availqty). This is very time-consuming because there are 10000 tuples in the table. On the other hand, with an index on the column ps_availqty, we can simply use the index to find the specific tuples that qualifies for the query. If the index is clustered, then this operation would take a very short time. Even if the index is not clustered, since the query is very highly selective (very few tuples qualify), using the index is still a better plan than sequential scan.

4.3 Index on range select

1) 5916 tuples will be returned by this plan; the total cost is 199.80;

2) Explanation

The reduction factor is approximated by $\frac{value(I) - Low(I)}{High(I) - Low(I)}$ where I is the index on ps_availqty. Low(I) is the lowest value in the index. High(I) is the highest value in the index. Value is equal to 150. The cardinality of the results is again calculated by multiplying reduction factor with the number of tuples in the table (again, assume uniform distribution.)
Explanation for cost is the same as the answer to "question 1" in 4.2.

3) The access method is Sequential Scan;

Disable the access method in 3)

4) The total cost now is 396.23;
Order:

After we turn off sequential scan access method, the optimizer is forced to use index scan for this query. Since the index was built on the column ps_availqty, the returned tuples would be order by ps_availqty. In other words, it will return the tuples that have ps_availqty = 0, then 1, then 2 and so on up to 149. Sequential scan would be returned in order of the tuples' primary key.

Is it the same as step 1)? NO

5) Explanation:

In this case, sequential scan is actually cheaper than index scan because the index is unclustered. This means that every tuple that we find according to the index requires fetching a new page where that tuple is located (almost). The unclustered index scanning performs a random-like access pattern which makes data locality very bad and makes the prefetch almost useless. This method is actually much slower than simply scan all the tuples. At least with sequential scan, we will never need to fetch a page again after it is done.

4.4 Join algorithm

1) Estimated total cost is ______211.44______;
Plan Tree:

```
                    π Distinct(supplier.s_name)
                    |
                   Sort
       (sort key = supplier.s_name)
                    |
                    ⋈ Hash Join
         (ps_suppkey = s_suppkey)
              /                 \
         Seq Scan               Hash
      (S ps_availqty < 4)         |
             |                Seq Scan
             |                    |
         partsupp                 |
                               supplier
```

2) Use ________Hash_______ join algorithm;

3) Number of tuples that will be retrieved from partsupp is ________120_______;

Disable the join algorithm in 2)
4) Now the join algorithm is _____Merge Join_____; the total cost is now _218.81 (estimated), 57.14 (actual)______.

Disable the join algorithm in 2) and 4)
5) Now the join algorithm is ______Nested Loop_____; the total cost is now __________772.53 (estimated), 243.85 (actual)____.