CS386D Problem Set #3

1. Justify the statement: “as long as the amount of memory available is between the square root of the data file size and the size of the entire data file, the file can always be sorted into two passes”.

2. Given the following SQL select statement:

```
select *
from R
where A.exceeds(20) and B.includes(40) and C.overlaps(40,20)
```

   (a) Suppose the **exceeds** predicate has an execution cost of 5/tuple and a selectivity of .3, the **includes** predicate has an execution cost of 20/tuple with a selectivity of .1; and the **overlaps** predicate has an execution cost of 7/tuple and with a selectivity of .2. What order of evaluation minimizes query evaluation costs if R is to be scanned?

   (b) Now suppose the costs for all of these predicates are 0. What order of predicate evaluation would minimize query evaluation costs? (Hint: order actually does make a difference).

4. What is the ‘plumbing diagram’ that would process the following MDX query efficiently? The cube is the SalesCube covered in the course notes.

```
SELECT
{USA_North.*, USA_South, Japan}        on COLUMNS
{Time.Q1.*,Time.Q2,Time.Q3,Time.Q4.*} on ROWS
FROM SalesCube
WHERE ( Sales.Jones, Sales.Rao, Time.2011, Measures.sales )
```

(yes, I know #3 is missing, but I can’t figure out how to make ‘4’ to be ‘3’ in MS Word. I hate Word).

Solutions

1. Justify the statement: "as long as the amount of memory available is between the square root of the data file size and the size of the entire data file, the file can always be sorted into two passes".

Answer: Assume a file has n^2 blocks. On the first pass, read in n blocks at a time, sort them in memory, and output n blocks (whose tuples are now ordered). This sequence of n blocks is called a **run**. In the first pass, n runs are produced. In the second pass, you merge n sorted runs. Thus, if you have internal memory of n blocks, you can sort a file of n^2 blocks in 2 passes. If you have less than n blocks in memory, you can't sort the file in 2 passes.

2. Given the following SQL select statement:

```
select *
from R
where A.exceeds(20) and B.includes(40) and C.overlaps(40,20)
```

4. Suppose the **exceeds** predicate has an execution cost of 5/tuple and a selectivity of .3, the **includes** predicate has an execution cost of 20/tuple with a selectivity of .1; and the **overlaps** predicate has an execution cost of 7/tuple and with a selectivity of .2. What order of evaluation minimizes query evaluation costs if R is to be scanned?
5. Now suppose the costs for all of these predicates are 0. What order of predicate evaluation would minimize query evaluation costs? (Hint: order actually does make a difference).

Answer: (a) This was a no-brainer problem. The rank of **exceeds** is 5/.7 = 7.14, the rank of **includes** is 20/.9=22.2, and the rank of **overlaps** is 7/.8=8.75. Process **exceeds**, then **includes**, then **overlaps** in that order.

(b) Here's an odd corner case: when the cost is "0" (really, something very small), the ordering is based on selectivities: the most selective predicate goes first. The ordering for this problem would be **includes** first, then **overlaps**, then finally **exceeds**.

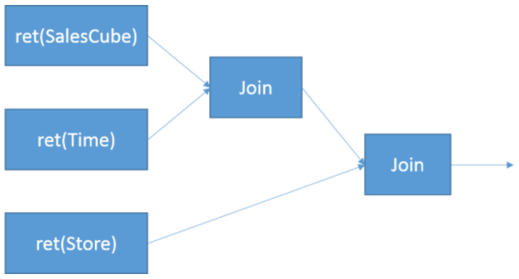
3. What is the 'plumbing diagram' that would process the following MDX query efficiently?

```
SELECT
{USA_North.*, USA_South, Japan}        on COLUMNS
{Time.Q1.*,Time.Q2,Time.Q3,Time.Q4.*} on ROWS
FROM SalesCube
WHERE ( Sales.Jones, Sales.Rao, Time.2011 )
```

Answer: first, look at the SQL query that would merge all of these retrieval requests together:

> **Select ***
> **from SalesCube NJ Time NJ Store**
> **where RepName in (Jones, Rao) and Year=2011 and**
> **(Quarter=Q1 or Quarter=Q3 or Quarter=Q3 or Quarter=Q4) and**
> **(Region=USA_North or Region=USA_South or Country=Japan)**

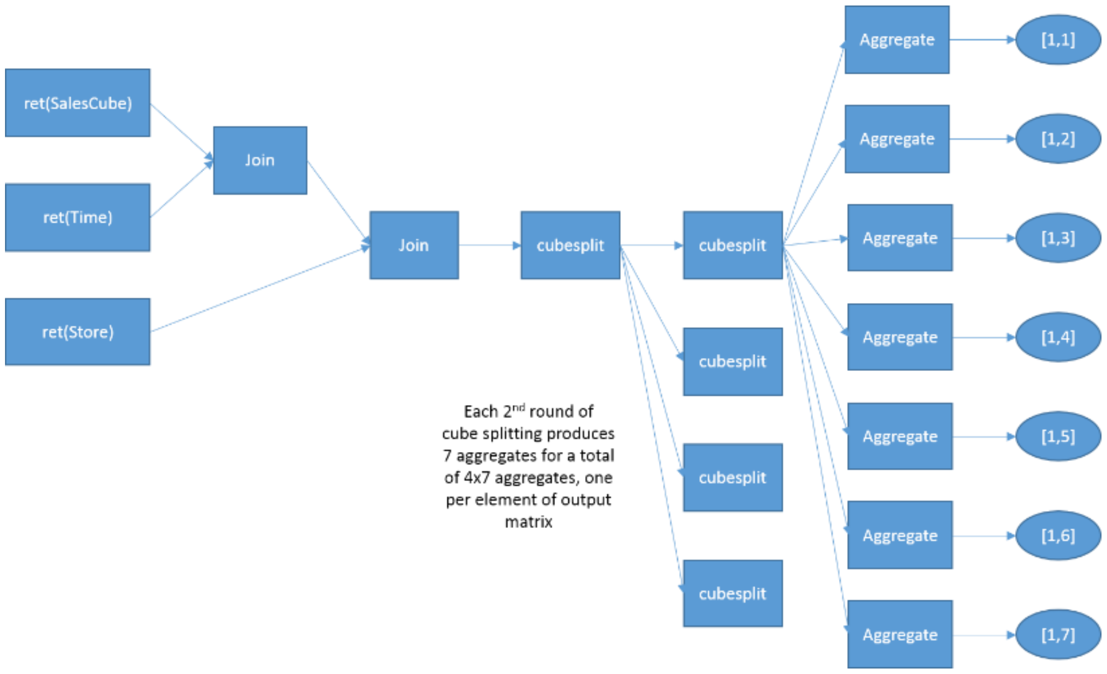
The plumbing diagram of this query is, where Store could be joined with SalesCube first. (It doesn't matter for this problem):



The output of this query is then partitioned into 4x7=28 streams, one stream per element of the matrix. Each stream is aggregated into a single number, which is an element of the matrix. 28 streams, 28 elements computed.

Cubesplit(State=Vermont, State=Maine, Region=USA_South, Country=Japan)
followed by
Cubesplit(Month=Jan, Month=Feb, Month=Mar, Quarter=Q2, Month=Oct, Month=Nov, Month=Dec)

The plumbing diagram looks like:


ret(SalesCube)
ret(Time)
ret(Store)
Join
Join
cubesplit
cubesplit
cubesplit
cubesplit
cubesplit
Each 2nd round of cube splitting produces 7 aggregates for a total of 4x7 aggregates, one per element of output matrix
Aggregate
Aggregate
Aggregate
Aggregate
Aggregate
Aggregate
Aggregate
[1,1]
[1,2]
[1,3]
[1,4]
[1,5]
[1,6]
[1,7]