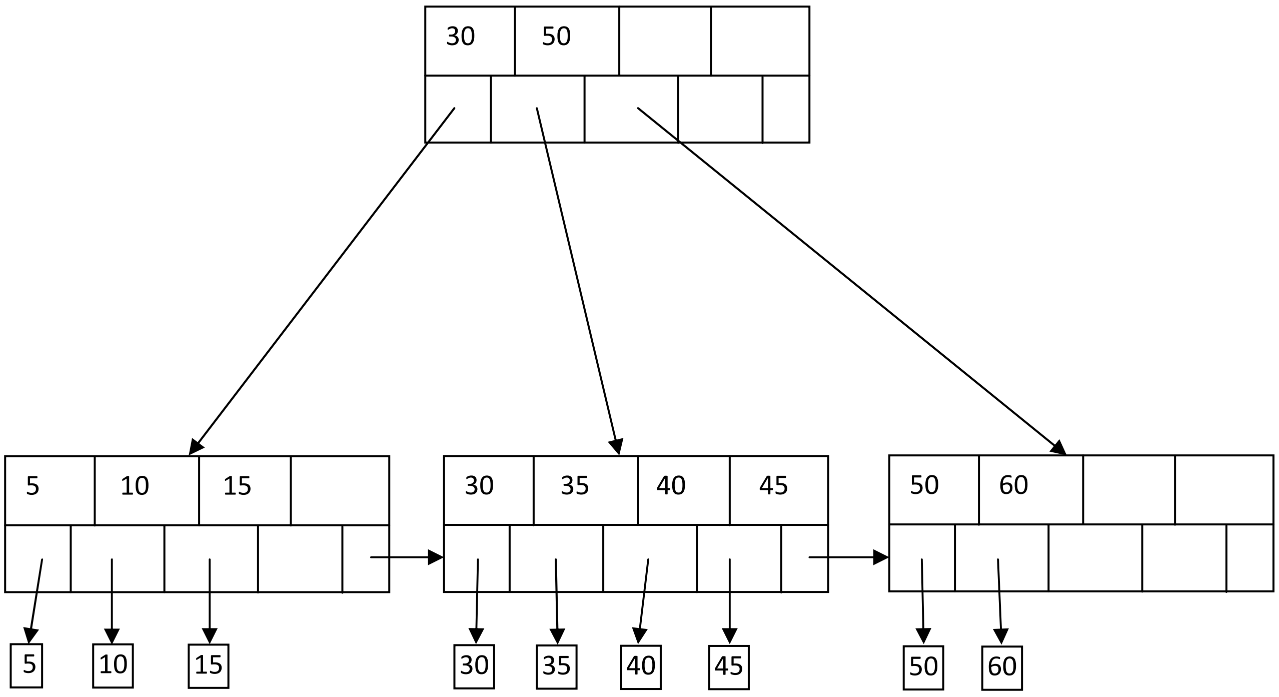
CSE 444 Final Exam

August 21, 2009

Name ________________________________

Question 1	/ 15
Question 2	/ 25
Question 3	/ 25
Question 4	/ 15
Question 5	/ 20
Total	/ 100

Question 1. B+ trees (15 points) Suppose we have the following B+ tree, where each index node has a maximum of 4 children.



Show how this B+ tree changes if the values 65 and 42 are inserted into the tree one after the other.

Question 2. Logical query plans (25 points) This problem and the next concern two relations R(a, b, c) and S(x, y, z) that have the following characteristics:

$B(R) = 600$	$B(S) = 800$
$T(R) = 3000$	$T(S) = 4000$
$V(R, a) = 300$	$V(S, x) = 100$
$V(R, b) = 100$	$V(S, y) = 400$
$V(R, c) = 50$	$V(S, z) = 40$

We also have M = 1000 (number of memory blocks).

Relation R has a clustered index on attribute a and an unclustered index on attribute c. Relation S has a clustered index on attribute x and an unclustered index on attribute z. All indices are B+ trees.

Now consider the following logical query plan for a query involving these two relations.

$$\begin{tikzcd}
& \Pi_{R.c,\ ans} \arrow[d, dash] & \\
& \gamma_{R.c,\ sum(S.x) \rightarrow ans} \arrow[d, dash] & \\
& \sigma_{R.c\ =\ 10\ and\ S.z\ =\ 17} \arrow[d, dash] & \\
& \underset{R.b\ =\ S.y}{\bowtie} \arrow[dl, dash] \arrow[dr, dash] & \\
R & & S
\end{tikzcd}$$

(Answer the questions about this query on the following page)

Question 2. (cont.) (a) Write a SQL query that is equivalent to the logical query plan on the previous page.

(b) Change or rearrange the original logical query plan to produce one that is equivalent (has the same final results), but which is estimated to be significantly faster, if possible. Recall that logical query optimization does not consider the final physical operators used to execute the query, but only things at the logical level, such as the sizes of relations and estimated sizes of intermediate results.

You should include a brief but specific explanation of how much you expect your changes to improve the speed of the query and why.

Draw your new query plan and write your explanation below. *If* you can clearly and easily show the changes on the original diagram, you may do so, otherwise draw a new diagram here.

Question 3. Physical query plans (25 points) This problem uses the same two relations and statistics as the previous one, but for a different operation not related to the previous query.

As before, the relations are R(a, b, c) and S(x, y, z). Here are the statistics again:

$B(R) = 600$	$B(S) = 800$
$T(R) = 3000$	$T(S) = 4000$
$V(R, a) = 300$	$V(S, x) = 100$
$V(R, b) = 100$	$V(S, y) = 400$
$V(R, c) = 50$	$V(S, z) = 40$

M = 1000 (number of memory blocks)

Also as before, relation R has a clustered index on attribute a and an unclustered index on attribute c. Relation S has a clustered index on attribute x and an unclustered index on attribute z. All indices are B+ trees.

Your job for this problem is to specify and justify a good physical plan for performing the join

$R \bowtie_{a=z} S$ (i.e., join R and S using the condition R.a = S.z)

Your answer should specify the physical join operator used (hash, nested loop, sort-merge, or other) and the access methods used to read relations R and S (sequential scan, index, etc.). Be sure to give essential details: i.e., if you use a hash join, which relations(s) are included in hash tables; if you use nested loops, which relation(s) are accessed in the inner and outer loops, etc.

Give the estimated cost of your solution in terms of number of disk I/O operations needed (you should ignore CPU time and the cost to read any index blocks).

You should give a brief justification why this is the best (cheapest) way to implement the join. You do not need to exhaustively analyze all the other possible join implementations and access methods, but you should give a brief discussion of why your solution is the preferred one compared to the other possibilities.

(Write your answer on the next page. You may remove this page for reference if you wish.)

Question 3. (cont) Give your solution and explanation here.

Question 4. XML (15 points) It's never too early to think about the holidays – at least not if you are in the toy business. Santa's elves are using XML to keep track of the toys in Santa's sack this year. The DTD for their data is the following:

```
<!DOCTYPE sack [
<!ELEMENT sack (toy)* >
<!ELEMENT toy (name, color*, age?) >
<!ELEMENT name (#PCDATA) >
<!ELEMENT color (#PCDATA) >
<!ELEMENT age (#PCDATA) >
]>
```

Although it is not specified in the DTD itself, the values for age will be one of "baby", "child", "teen", "adult", or "everyone", if age is present in an entry. If a toy is available in multiple colors, it will have multiple color attributes in its entry. Here is a short example of the kind of data that we might find in Santa's database:

```
<sack>
    <toy>
        <name>firetruck</name>
        <color>red</color>
        <age>child</age>
    </toy>
    <toy>
        <name>rattle</name>
        <color>pink</color>
        <color>blue</color>
        <age>baby</age>
    </toy>
</sack>
```

On the next page, write XPath or XQuery expressions as requested to perform the desired tasks.

(You may detach this page for reference if you wish.)

Question 4. (cont). Use the XML schema on the previous page to answer these questions. If it matters, you can assume that the data is stored in a file named "sack.xml".

(a) Write an XPath expression (not a more general XQuery) that returns the names of all toys that are available in the color purple (i.e., have at least one color entity with the value "purple").

(b) Write an XQuery expression that reformats a valid XML document specified by the original DTD to give a list of toy names grouped by age. The resulting document should include all of the toys in the original document and should match this DTD:

```
<!DOCTYPE agelist [
<!ELEMENT agelist (group)* >
<!ELEMENT group (age, name*) >
<!ELEMENT name (#PCDATA) >
<!ELEMENT age (#PCDATA) >
]>
```

Question 5. Map-Reduce (20 points) For each of the following problems describe how you would solve it using map-reduce. You should explain how the input is mapped into (key, value) pairs by the map stage, i.e., specify what is the key and what is the associated value in each pair, and, if needed, how the key(s) and value(s) are computed. Then you should explain how the (key, value) pairs produced by the map stage are processed by the reduce stage to get the final answer(s). If the job cannot be done in a single map-reduce pass, describe how it would be structured into two or more map-reduce jobs with the output of the first job becoming input to the next one(s).

You should just describe your solution algorithm. You should not translate the solution into Pig Latin or any other detailed programming language.

(a) The input is a list of housing data where each input record contains information about a single house: (address, city, state, zip, value). The output should be the average house value in each zip code.

Question 5. (cont.) (b) [Same as the last part of project 3, only this time using map-reduce instead of SQL] The input contains two lists. One list gives voter information for every registered voter: (voter-id, name, age, zip). The other list gives disease information: (zip, age, disease). For each unique pair of age and zip values, the output should give a list of names and a list of diseases for people in that zip code with that age. If a particular age/zip pair appears in one input list but not the other, then that age/zip pair can appear in the output with an empty list of names or diseases, or you can omit it from the output entirely, depending on which is easier.

(Hint: the keys in a map/reduce step do not need to be single atomic values.)