# COMP 3311: Database Management Systems

## Query Processing Cost Estimates (page I/Os)

**Algorithm A1 (*linear search*)**

$b_r$ the number of pages that contain records of relation $r$

***or***

$b_r/2$ if selection is on a key attribute, since we stop when we find the record (as there is at most only one with the specified value).

**Algorithm A2 (*binary search*)**

$\lceil \log_2(b_r) \rceil$ the cost of locating the first tuple by a binary search on the pages ***plus*** the number of pages that contain records that satisfy the selection condition

**Algorithm A3 (*primary index on key, equality search*)**

For a tree index: $HT_i + 1$ where $HT_i$ is the height of the tree index

For a hash index: Normally a hash index would not be used in this case as the file would use a hash file organization.

**Algorithm A4 (*primary index on nonkey, equality search*)**

$HT_i$ + the number of pages that contain records that satisfy the selection condition

**Algorithm A5 (*secondary index, equality search*)**

*search key is a candidate key*

For a tree index: $HT_i + 1$ where $HT_i$ is the height of the tree index

For a hash index: 1 + 1 or 1.2 + 1 if there exist overflow buckets

*search key is <u>not</u> a candidate key*

For a tree index: $HT_i$ + *cost to retrieve indirection pointers* + number of records retrieved (assumes each record requires a page access)

For a hash index: 1 + cost to retrieve indirection pointers + number of records retrieved or 1.2 + cost to retrieve indirection pointers + number of records retrieved if there exist overflow buckets

**Algorithm A6 (*primary index, comparison*)**

$\sigma_{A \geq v}(r)$ use the index to find the first tuple where $A \geq v$ and scan the relation sequentially from there.

$\sigma_{A \leq v}(r)$ do not use the index; instead scan the relation sequentially until you find the first tuple where $A > v$.

**Algorithm A7 (*secondary index, comparison*)**

$\sigma_{A \geq v}(r)$ use the index to find first index entry where $A \geq v$ and scan the index sequentially from there to find pointers to the tuples.

$\sigma_{A \leq v}(r)$ scan the leaf pages of the index finding pointers to records, until the first entry where $A > v$.

**Algorithm A8 (*conjunctive selection using one index*)**

Select a combination of $\theta_i$ and algorithms A1 through A7 that results in the least cost for $\sigma_{\theta i}(r)$.

Check the remaining conditions on the tuple after retrieving it into memory.

**Algorithm A9 (*conjunctive selection using composite index*)**

If available, use an appropriate composite (multiple-attribute) index.

The type of index determines which of the algorithms A2, A3 or A4 will be used.

**Algorithm A10 (*conjunctive selection by intersection of identifiers*)**

Using indexes: sum of the costs of the individual index scans ***plus*** the cost of retrieving the tuples

**Algorithm A11 (*disjunctive selection by union of identifiers*)**

Using indexes: sum of the costs of the individual index scans ***plus*** the cost of retrieving the tuples

**Negation (NOT): $\sigma_{\neg\theta}(r)$**

Use linear search on the file.

If very few records satisfy $\neg\theta$ and an index is applicable for $\theta$ then find satisfying records using the index and retrieve the records from the file.

**Block nested-loop join**

Worst case: $b_r * b_s + b_r$

Best case: $b_r + b_s$

If neither relation fits in memory, use the smaller relation as the outer relation *r*. If a relation fits in memory, use it as the inner relation *s*.

**Indexed Nested-Loop Join**

$b_r + n_r * c$ where c is the cost of traversing the index and fetching all matching s tuples for one tuple or r.

**Merge Join**

$b_r + b_s$ + cost of sorting if relations are unsorted

**Hash Join**

3 * ($b_r + b_s$)

**Projection Operation: Using Sorting**

Pass 0: the original relation is read and the same number of smaller tuples are written out.

Merge passes: fewer tuples are written out in each pass.

**Projection OPERATION: USING HASHING**

For partitioning: Read *r* and write out each tuple, but with fewer attributes.

For duplicate elimination: Read the result of the partitioning phase.

**Aggregate Operations: Without Grouping**

In general, requires scanning the relation.

If we have a tree index whose search key includes *all attributes* in the **select** or **where** clauses, we can do an index-only scan.

**Aggregate Operations: With Grouping**

Can use the same techniques as for duplicate elimination (i.e., sorting or hashing).

Instead of eliminating duplicates, gather them into groups and apply aggregate operations on each group.

If we have a tree index whose search key includes *all attributes* in **select**, **where** and **group by** clauses, we can do an index-only scan.