Linear Search
- Items can be in any order,
- Have to examine first record, then second record, then third record, etc., until item is found or all items have been examined,
- Worst case search time (item not found) is $O(N)$ for $N$ items,
- Search time grows linearly as a function of $N$.

Binary Search
- Items must be sorted on search field,
- Examine middle record, stop if found, but if not found discard half of list known to not contain item, repeat until found or list empty.
- Worst case search time (item not found) is $O(\log_2(N))$ for $N$ items,
- Search time grows logarithmically as a function of $N$.

Sorting
- Very time expensive,
- Worst case sort time is $O(N\times\log_2(N))$ for $N$ items for good sorting algorithms or $O(N^2)$ for bad sorting algorithms,
- Worth it to sort once if binary search can be used many times.
PART #2:
INDEXING vs. SORTING

SORTING

- Physically rearranges table to be in the desired order.
- As we’ve seen, sorting takes a long time: \(O(N\times\log_2(N))\) for \(N\) records.

INDEXING

- Indexes are additional hidden data structures associated with individual fields in a table.
- Indexes are maintained automatically by the database package.
- An index makes a table “look sorted” on the indexed field.
- Every field may have its own index.
- Indexes speed up both searches and joins.

What if we need to see Table in several different orders?

- Option #1: Re-sort table each time a new view is needed.
  - Only one table, but...
  - ...takes lots of unnecessary time.
- Option #2: Make several copies of table, each sorted on a different field.
  - Many copies means lots of disk space used, and...
  - Adding/Deleting/Changing record in one means same change must be made to all (data consistency).
- Option #3: Use Indexes.

A Table without Indexing

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12345 Fred</td>
</tr>
<tr>
<td>2</td>
<td>72401 Joe</td>
</tr>
<tr>
<td>3</td>
<td>22222 Mary</td>
</tr>
<tr>
<td>4</td>
<td>54321 Sam</td>
</tr>
<tr>
<td>5</td>
<td>20202 Martha</td>
</tr>
<tr>
<td>6</td>
<td>11111 Bob</td>
</tr>
<tr>
<td>7</td>
<td>47904 Tom</td>
</tr>
</tbody>
</table>

Tables without Indexes

- In a table without any indexes, data records may appear in any random order.
- A table **might** have its records in a sorted order without an index.
- Searching an unindexed table for matching records requires examining every record in the table. This is a linear search, which runs in \(O(N)\) time for \(N\) records.
The People table apparently is indexed on the ID field.
People might be unindexed, but if it is indexed it would have to be on the ID field.

Typically, indexed fields contain unique information (no duplicates).
Such fields can be searched with binary search in $O(\log_2(N))$ time. (Check middle item, discard half not containing search term, repeat until found or list becomes empty).

JOINs

Something spreadsheets can’t do.
Synthesize new tables from two or more source tables.
Source tables must have a field in common (doesn’t have to have the same name, but must have compatible data types).
Result may contain more records than either source table, or may be empty.
Inner Join on ID Field (Intersection)

Inner Join
- Contains only records where join-field in one table matches join-field in the other table.
- In this case, there are only three matches. Those three results will be filled in with data from both tables.

Left Outer Join on ID Field

Left Outer Join
- Contains all data from the left table.
- The matches with the right table will have their records completely filled in.
- Answer records from the left with no match in the right will have "holes".
Right Outer Join
- Contains all data from the right table.
- The matches with the left table will have their records completely filled in.
- Answer records from the right with no match in the left will have “holes”.

Full Outer Join on ID Field (Union)

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bob</td>
<td>$40,000</td>
</tr>
<tr>
<td>2</td>
<td>Fred</td>
<td>$45,000</td>
</tr>
<tr>
<td>3</td>
<td>Martha</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Mary</td>
<td>$10,000</td>
</tr>
<tr>
<td>5</td>
<td>Sam</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Tom</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Joe</td>
<td>$35,000</td>
</tr>
<tr>
<td>8</td>
<td>John</td>
<td>$25,000</td>
</tr>
<tr>
<td>9</td>
<td>Jane</td>
<td>$30,000</td>
</tr>
</tbody>
</table>

For our example
- There are 7 records in the left table.
- There are 6 records in the right table.
- There are 3 matches (the inner join).
- The left outer join will contain 7 records, 3 filled in and 4 with holes.
- The right outer join will contain 6 records, 3 filled in and 3 with holes.
- The full outer join contains 4 (from left) + 3 (inner) + 3 (from right) = 10 records.

Records in Result of Joins

Part 3B: Relationships
- Many:Many
  - Neither table indexed on join field
- 1:Many or Many:1
  - One table indexed on join field
- 1:1
  - Both tables indexed on join field
Many:Many Relationships

- Neither table has an index on the join-field.
- Any record in either table may have multiple matches in the other table.
- Every record in one table is compared against all records in the other table.
- For M records in one table and N records in the other, there will be O(M \times N) comparisons.
- It is possible to have M \times N records in the answer (all match), or none (no match).

1:Many or Many:1 Relationships

- One table (M records) is without an index, the other (N records) is indexed on the join-field.
- The indexed table typically has unique information in the indexed field.
- Every record in the unindexed table does a binary search into the indexed table.
- A record in the unindexed table has at most one match in the indexed table.
- A record in the indexed table may have multiple matches in the unindexed table.
- There are O(M \times \log_2(N)) comparisons.
1:1 Relationships

- Both tables have an index on the join-field.
- Joining the tables requires only one pass each over the two tables.
- A record in either table has at most one match in the other table.
- There are at most $O(M+N)$ comparisons.

Suppose $M = N = 1024$ Records

- Many:Many
  - $M \times N = 1,048,576$ comparisons
  - Possibly $1,048,576$ records in answer
- 1:Many or Many:1
  - $M \times \log_2(N) = 1024 \times 10 = 10,240$ comparisons
- 1:1
  - $M+N = 2048$ comparisons (maximum)

1:1 Join

Both Tables have Indexes

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
<th>PEOPLE</th>
<th>PAYMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11111 Bob</td>
<td>11111</td>
<td>540,000</td>
</tr>
<tr>
<td>2</td>
<td>12345 Fred</td>
<td>13333</td>
<td>545,000</td>
</tr>
<tr>
<td>3</td>
<td>20202 Martha</td>
<td>54321</td>
<td>520,000</td>
</tr>
<tr>
<td>4</td>
<td>22222 Mary</td>
<td>60233</td>
<td>530,000</td>
</tr>
<tr>
<td>5</td>
<td>47904 Tom</td>
<td>72401</td>
<td>535,000</td>
</tr>
<tr>
<td>6</td>
<td>54321 Sam</td>
<td>97330</td>
<td>550,000</td>
</tr>
<tr>
<td>7</td>
<td>72401 Joe</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Conclusion

- Indexing speeds up both queries and joins.
- In a join, having one index is faster than having none, having two indexes is even faster.
- For many:many relationships, neither table will have an index so joins will be slow (oh, well), and may generate many more records than are in either source table.