Sketchbook: Efficient Indexing and Sorting via Transformation — B-Trees and Linear Sort

Santhoshini Goskula
801420688

Part-1: Dataset Transformation

What could be the result of the transformation code? The answer template is as follows:

The result of the transformation code would be
(23,{ID:1034, Name:"Alice", Age:23})
(34,{ID:1012, Name:"Bob", Age:34})
(27, {ID:1089, Name:"Carol", Age:27})
(23,{ID:1005, Name:"David", Age:23})

Part-2: Linear Sorting

A) With the results as input of a counting-based linear sort (code), what are element values in the count array at the end of the second iteration of the fifth for loop? The answer template is as follows :

The count array at the end of the second iteration of the fifth for loop would be: { 1, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 4 }

B) Please also discuss why linear sort can outperform comparison-based sorting (for example, insertion sort , selection sort, and merge sort) in certain cases. The answer template is as follows:

- Comparison-based sorting has a lower bound of **O(n log n)** comparisons (hence **O(n log n)** time under unit-cost comparisons) in the average and worst cases.
- Linear sorting algorithms avoid comparisons between elements, which allows them to achieve **O(n + k)** or **O(n)** performance, where k is the range of possible key values.

Part-3: Building a B-Tree Index

1. What is the key-value pair in the first child at the level 1 of the BTree?

The key-value pair in the first child at the level 1 of the BTree is **(23, {ID:1034, Name:"Alice", Age:23})**

2. What are the key-value pairs in the second child at the level 1 of the BTree?

The key-value pairs in the second child at the level 1 of the BTree are
(27, {ID:1089, Name:"Carol", Age:27}) and **(34, {ID:1012, Name:"Bob", Age:34}).**

3. When searching for the key 27, how many node(s) would be visited?

When searching for the key 27 in the BTree, **2** node(s) are visited.

Part-4: Comparative Analysis

1. How does the complexity of B-Tree insertion compare with linear sort?

Each insertion takes **O(log_t n)** node visits (≈ O(log n) time with binary search in-node), so inserting n items costs **O(n log_t n)**.

2. In what situations is building a B-Tree more appropriate than applying a sort?

Linear sort: Runs in **O(n + k)** when key range k is small.
Linear sort is faster when the dataset is static and key range is limited; B-Trees are slower but support efficient dynamic operations (insert/search/delete).

3. What happens if data arrives as a continuous stream — can linear sort still be used?

B-Trees are better for dynamic, searchable datasets, while sorting suits **static** datasets.
For streaming data, we can not use **linear sort** since it requires access to the entire dataset at once, but we can use incremental structures like **B-Trees** or heaps, which can maintain sorted order dynamically as new data arrives.