`SearchList` vs `AdvancedSearchList`

📘 1. `SearchList`: Basic Search using `.filter()` + `.includes()`

✅ Features:

- Filters list of items based on user input
- Case-insensitive match
- Uses `Array.prototype.filter()` and `String.prototype.includes()`

🔧 Code Summary:

```
const filteredItems = items.filter(item =>
  item.toLowerCase().includes(query.toLowerCase())
);
```

🧠 Step-by-Step Execution:

1. User types in the input.
2. `query` state updates via `setQuery()`.
3. `items.filter()` runs on every render.
4. Each item is converted to lowercase.
5. `includes()` checks if item contains the query.
6. Matching items are displayed in a `<ul>` list.

⏱ Time Complexity:

- `.filter()` → O(n)
- `.includes()` → O(m) per item

- **Total**: O(n × m)

🔁 Workflow Diagram:

[User Input]
↓
Update query via setQuery()
↓
Run items.filter()
↓
Run includes() on each item
↓
Render matching list

✅ When to Use:

- Small or static datasets
- Quick prototyping

❓ Sample Interview Q&A:

- **Q: How does this work?** A: It scans every item and checks if the search query is a substring.
- **Q: What is its complexity?** A: O(n × m), where n = number of items, m = average item length.

📘 2. `AdvancedSearchList`: Efficient Search using Trie

✅ Features:

- Performs prefix-based search
- Case-insensitive match
- Uses a Trie for fast lookup
- Uses `useMemo()` to avoid rebuilding on every render

🧠 Key Functions Explained:

🔹 `insert(word)`

- Adds a word to the Trie, character by character.
- Marks the last character as `isEndOfWord = true`.

🔹 `searchPrefix(prefix)`

- Navigates to the node that matches the prefix.
- Calls `collectAllWords()` to get full matches below.

🔹 `collectAllWords(node, prefix)`

- Recursive function that collects all words starting from a node.

⏱ Time Complexity:

- **Insert:** O(k) per word (k = word length)
- **Search:** O(k) to traverse prefix
- **Collect:** O(r), where r = results returned
- **Total:** O(k + r)

🔁 Workflow Diagram:

[User Input]
 ↓
Update query via setQuery()
 ↓
trie.searchPrefix(query)
 ↓
Go to prefix node in Trie
 ↓
collectAllWords() to get matches
 ↓
Render matching list

✅ **When to Use:**

- Large datasets
- Prefix search/autocomplete
- Performance-critical apps

❓ **Sample Interview Q&A:**

- **Q: Why use a Trie here?** A: It optimizes prefix search to O(k), faster than linear search.
- **Q: What's the role of useMemo()?** A: Prevents rebuilding the Trie on each render.
- **Q: What if I want to use fuzzy or partial matching?** A: Then you might integrate `Fuse.js` or revert to `.includes()` logic.

🧠 Summary Table:

Feature	SearchList (Filter)	AdvancedSearchList (Trie)
Matching Type	Substring	Prefix
Time Complexity	O(n × m)	O(k + r)
Suitable For	Small lists	Large lists, real-time UX
Custom Matching	Easy (includes/starts)	Needs extensions
Memory Usage	Low	Higher (more nodes)
Performance	Slower on scale	Fast + Scalable

Let me know if you'd like this visualized as a diagram or exported as a PDF!