# Algorithms, Data Structures, and Programming Languages

|  |  |
| --- | --- |
| Type: | Advanced Workbook |
| Course of Study: | B.Sc.Computer Science |
| Author: | Leo Wolf |
| Matriculations Number: | 9210825 |
| Tutor’s Names: | Musharaf Doger & Mr. Libbrecht |
| Date: | 05.12.2024 |

**Task 1: Structure analysis for football problem**

|  |  |
| --- | --- |
| 1. **Sorted Array:**   Description – An array where elements (IDs) are always stored in sorted order  Advantages:   1. Efficient Searching – Binary search can be used to find an ID in O(log n) time, making searching very fast. 2. Compact Storage – Minimal overhead as only the IDs are stored (no additional pointers or structures).   Disadvantages:   1. Deletion/Insertion Overhead – Adding and deleting an ID requires shifting elements to maintain the sorted order which takes O(n) in the worst case 2. Dynamic Resizing – if implemented as a static array, resizing may be required when the array reaches capacity, adding complexity. | 1. **Linked List:**   Description – A linear collection of nodes where each node contains an Id and a pointer to the next node  Advantages:   1. Easy Insertion and Deletion – Adding or removing an ID can be done in O(1) time if the position is known. 2. Dynamic size – No need for resizing as memory is allocated as needed.   Disadvantages:   1. Inefficient searching – Seaching requires traversing the list, taking O(n) in worst case. 2. Memory overhead – Additional memory is required for linking nodes 3. Sequential access – Cannot use binary search or direct indexing, limiting performance for searching |
| 1. **Chain (Hash Table with chaining):**   Description – A hash table in which each slot stores a linked list (similar layout) containing IDs that generate the same hash value.  Advantages:   1. Efficient searching – Depending on the hash function, a search for an ID should take O(1) time on average. 2. Insertion/Deletion – These operations can be executed in O(1) average time 3. Dynamic size – slots can grow as needed   Disadvantages:  Complexity – Hashing can lead to clustering, reducing the efficiency.  Memory overhead – Slots and chains require more memory compared to other structures. | 1. **Heap:**   Description – A binary tree-based structure  Advantages:   1. Access to extremes – Accessing smallest or largest ID is O(1), depending on the heap type. 2. Dynamic size – increases or decreases as IDs join or exit   Disadvantages:   1. Inefficient searching – as heaps are not optimized for random searches. 2. Adding/Deleting – Both these operations are time consuming taking O(log n), making it less efficient than other structures. 3. Ordering – IDs wont be stored in order. |

**Task 2: Creating a Utility for restaurant owner**

1. The utility will assist restaurant owners in calculating the selling price of menu items based on ingredient costs, generating a formatted menu card in tabular form. The application will include functionalities for entering ingredient data, defining recipes, calculating selling prices with tax, and displaying a finalized menu.

We can break this down into 3 objects. The utility will have an ingredients object a recipes object, and a menu object

Ingredients object:

Functionality - Inputting and storing new ingredients as well as updated existing ingredients created by having three attributes name, unit type, and cost per unit.

Functions for this object will include:

1. addIngredient(name, unitType, cost) – here the users (owners) will input new ingredients to add to their collection of ingredients.
2. updateIngredient(name, unitType, cost) – this will search for the name of the ingredient and then update that ingredient with new unit type and cost (in case supplier changed or price increased)

Recipes object:

Functionality – This object will allow the user to define recipes with the required ingredients and unit amount required for the recipe.

Functions for this object will include:

1. addRecipe(recipeName, ingredients, unitAmount) – This will add recipe with the ingredients and amounts for those ingredients (inputting of ingredients can be done via a drop box and text box for values)
2. DeleteRecipe(recipeName) – this will remove the recipe

Menu object:

Functionality – This will take all the recipes, calculate the cost for the recipes, and then display them in a table format for the customers to see.

Functions for this object will include:

1. calculatePrice(recipe) – this function will traverse through the recipe’s ingredients calculating the total price from the amount need and the cost per unit price in the ingredient object. At the end it will add a 25% profit margin and a 10% tax
2. displayMenu(recipes) – will take an array like structure of all the recipes and then display them in a table format with prices next to it.
3. JavaScript suitability:

Pros:

1. Client-Side Application - JavaScript can efficiently handle user interactions, such as ingredient entry, recipe input, and menu generation, directly in the browser without needing server-side processing.
2. Dynamic Updates - JavaScript excels at dynamically updating and rendering content, making it easy to display the calculated menu in real time as the user enters data.
3. Ease of Development - JavaScript, along with libraries like React or frameworks like Angular, simplifies the development of user-friendly interfaces and responsive designs.
4. Tabular Display - JavaScript, combined with HTML and CSS, allows for easy creation of a visually appealing menu in tabular format.
5. No Need for Complex Back-End - For a single-user, standalone tool like this, a JavaScript-based front-end application is sufficient without requiring server-side technologies.

Cons:

1. Large-Scale Data Handling:
   * JavaScript alone may not handle large datasets efficiently compared to a back-end solution.
2. Persistence:
   * If the application requires persistent storage of ingredients, recipes, or menus, additional tools (e.g., IndexedDB or localStorage) or server-side integration may be required.

Conclusion:

JavaScript is capably of delivering a utility like this, with its ease of use, dynamic rendering, and wide range of libraries. The only time it might be a bit of a change is when persistent storage or multi-user access is required other frameworks might be needed.

**Task 3: Stock price data**

1. Iterative Algorithm:

To find the high of the stock value in a 52-week period using a while loop, the following will have to be done:

* Traversing through the linked list.
* Set a variable value (which will be the stock high at the end) to a very low number.
* Compare that variable to the current stock price if our current max is lower making the update to set the max stock to that week’s value.
* Once at the end of the loop return the max value.

Example in python:

A screen shot of a computer code

Description automatically generated

1. Recursive Algorithm:

To modify the above into a recursive algorithm, you will need to do the following:

* Take the current value and the current maximum as inputs.
* You then compare the current nodes value with the maximum value if update need then update.
* It will then move to the next node by calling itself recursively.
* Once the end of the list is reached the maximum value will be returned.

Example in python:

A screen shot of a computer program

Description automatically generated

1. Pros and Cons:

**Iterative**

Pros Cons

1. Simpler and easier to use for this problem. 1. Slightly longer code area.
2. Efficient in terms of memory.
3. Less risk of stack overflow.
4. Faster in terms of execution as there is no  
   overhead from function calls.

**Recursive**

Pros Cons

1. Clearer and more concise code. 1. Can lead to stack overflow for large lists.
2. Natural fit for the problem in finding max value. 2. Slightly less efficient due to overhead from function calls.
3. Easy to read for developers comfortable with   
   recursions.

Looking at the Pros and Cons of the two methods Iterative would be safe for this task as it is more ideal for larger data sets. Calculating stock count does stop and so in the future when the weeks are more a recursive program might have a few more challenges even though the code looks cleaner and more concise.

**Task 4: Merging algorithms**

So, after careful analysis I would choose the Pairwise Merging algorithm, as I find it would be more efficient than the Sequential Merging algorithm. Here is why:

Pairwise merging:

This approach reduces the number of sequences by half in each step. It will continue until only one sequence remains. There are log2(k) steps and each step processes all n elements, the time complexity will be O(n.log2(k)), this will be more efficient when k’s value increases.

Sequential merging:

Using this approach, merging will take place by merging the first two sequences, then taking the result and merging it with the next sequence. Each merger processes all n elements and is repeated k – 1 times. Therefore, the time complexity will be O(n.k), growing for the number of k sequences.

Findings:

When looking at the graph flow of the time complexities you will see that the Sequential merge algorithm grows with a steep slope as the number of sequences rises, whereas the Pairwise merge algorithm shows a flatter curvature as the sequences increase. This making a Pairwise merge algorithm more efficient and better suited for scalability, hence why the Pairwise merge algorithm is the better choice in this scenario especially when the k sequences go up.

**Task 5:**