Algorithms and Data Structures

**FORMATIVE ASSESSMENT**

Adam Benjelloun

A report submitted in part fulfilment of the degree of

**MSc Computer Science with Artificial Intelligence**

A black and white logo

Description automatically generated

November 26, 2023

**Table of Contents**

Contents

[**Introduction** 3](#_Toc151921333)

[**Pseudocode Algorithms** 4](#_Toc151921334)

[**Algorithm 1 – calculateDepartmentTotalSalesOfADeptInAQuarter** 4](#_Toc151921335)

[**Algorithm 2 - calculateTotalSalesForQuarterWithTax** 5](#_Toc151921336)

[**Algorithm 3 – findMostEffectiveSalesMonthForEachDeptPerQuarter** 6](#_Toc151921337)

[**Bibliography** 8](#_Toc151921338)

**Pseudocode Algorithms**

**Algorithm 1 – calculateDepartmentTotalSalesOfADeptInAQuarter**

This algorithm leverages the fundamental principles of Java’s loop constructs for iterating over multidimensional arrays in order to calculate the total sales of a particular department in a specific quarter. It utilises the principles of encapsulation in object-oriented programming, a concept whereby“attributes [are] only accessible to the methods of the same class” [1, p.196].

CLASS SalesData

calculateDepartmentTotalSalesOfADeptInAQuarter (department, quarter)

INITIALIZE total = 0 // Set initial total sales to zero

FOR month = 1 to 3 // Iterate through each month in the quarter

total += salesData[department][quarter][month] // Sum sales for the month

RETURN total // Return the total sales for the department in the quarter

1. *Pseudocode Algorithm 1: Calculate the total sales of a specific department in a selected quarter:*

The algorithm starts by accepting that ‘department’ and ‘quarter’ are parameters which are key to accessing the relevant data within the sales data array. It also implies the use of these parameters as indices which is a fundamental concept in data retrieval from arrays [1, pp.209-210].

The algorithm then initialises a 'total' variable to zero, serving as an accumulator for sales figures. This ensures a clean starting point for the aggregation process, preventing any interference from previously stored values.

The algorithm employs a 'for' loop to iterate through the sales data array, covering each month from 1 to 3 within the specified quarter, thereby ensuring a comprehensive summation of the sales for that period [2, pp.45-46] In this and subsequent pseudocode examples, one-indexing is employed to enhance the readability of the program. This approach aligns with common numerical conventions, making it more intuitive and accessible, especially for individuals who may not have a background in programming logic.

Within the 'for' loop, the algorithm performs the key calculation using the ‘+=’ operator, which incrementally adds each month's sales figures from the sales data array to the total. This process efficiently aggregates the sales data for the specified department and quarter [3, p.105].

The algorithm concludes with a return statement, which outputs the final accumulated total of sales. This return value represents the comprehensive sum of sales for the specified department and quarter, marking the completion of the calculation process.

**Algorithm 2 - calculateTotalSalesForQuarterWithTax**

This algorithm exemplifies the application of Java's arithmetic operations, particularly in calculating total sales and applying tax. It showcases Java's capability in efficiently handling computations over multidimensional arrays, a fundamental aspect in financial data processing.

calculateTotalSalesForQuarterWithTax(quarter)

INITIALIZE totalSales = 0 // Initialize total sales for the quarter

INITIALIZE taxRate = 0.17 // Set tax rate to 17%

FOR department= 1 to NUM\_DEPARTMENTS // Iterate over all departments

totalSales += calculateDepartmentTotalSalesInAQuarter (department, quarter) // Add department sales to total

totalSales = totalSales + (totalSales \* taxRate) // Apply tax to total sales

RETURN totalSales // Return total sales including tax for the quarter

1. *Pseudocode Algorithm 2: Calculate the total sales for a quarter, and add tax:*

The algorithm begins by defining 'quarter' as the sole parameter, focusing the calculation exclusively on the sales data within that specific quarter. In contrast to the previous algorithm, the individual department is not a consideration here, as the objective is to calculate total sales across all departments [1, pp.209-210]. It is important to note that in this pseudocode representation, the number of departments is dynamically assigned to a variable rather than being hardcoded [2, pp.131-133]. This design choice provides flexibility and scalability to the algorithm, allowing for easy adaption to different organisational structures or changes in departments.

The next stage is the initialisation of the ‘totalSales’ and ‘taxRate’ variables. As with the previous pseudocode, ‘totalSales’ is set to zero, which establishes a baseline for accumulating the sales figures. The taxRate, is set to ‘0.17’ as it is a requirement of the application to add 17% tax to the total sales of the specific quarter.

A 'for' loop iterates over each department (from 1 to the specified number of departments), aggregating the total sales for the quarter. This iteration demonstrates the use of loop constructs in Java for going through arrays and collecting data, aligning with standard practices in data analysis and aggregation [2, p.84].

The algorithm then applies the tax rate to the total sales using the ‘+=’ addition assignment, illustrating the implementation of basic arithmetic operations in Java [3, p.105]. The use of a fixed tax rate highlights the adaptability of the algorithm to different financial scenarios, where such parameters may vary. The design choice to initialise the taxRate within the program as opposed to having an undefined variable demonstrates the different approaches to variable conventions in pseudocode [4, pp.20-22].

Finally, the algorithm returns the total sales amount inclusive of tax, this return statement signifies the completion of the financial calculation.

**Algorithm 3 – findMostEffectiveSalesMonthForEachDeptPerQuarter**

This comparative algorithm employs Java’s array structures for data storage and retrieval. The iterative comparison to identify peak sales periods demonstrates Java’s real-world capabilities in data analysis. This algorithm serves as an example of practical data analysis applications in retail.

findMostEffectiveSalesMonthForEachDeptPerQuarter(quarter)

FOR department = 1 to NUM\_DEPARTMENTS // Iterate through each department

INITIALIZE highestSales = salesData[department][quarter][1] // Start with sales of the first month

INITIALIZE mostEffectiveMonth = 1 // Start with the first month as most effective

FOR month = 1 to 3 // Iterate through each month in the quarter

currentMonthSales = salesData[department][quarter][month] // Get sales for the current month

IF currentMonthSales > highestSales THEN // Compare with highest sales so far

highestSales = currentMonthSales // Update highest sales

mostEffectiveMonth = month // Update most effective month

PRINT mostEffectiveMonth // Print the most effective month for this department

1. *Pseudocode Algorithm 3: Find most effective month of sales for each department in a selected quarter*

The algorithm initiates with a ‘for’ loop that iterates through each department from 1 to the specified total number of departments as defined by ‘NUM\_DEPARTMENTS’. During this iteration, the variables ‘highestSales’ and ‘mostEffectiveMonth’ are initialised [1, pp.124-128].

The ‘highestSales’ variable is initially set to the sales figure of the first month of the quarter for each department, establishing the baseline for the comparison in the subsequent ‘for’ loop which is nested within the first [1, p.73].

This nested ‘for’ loop iterates through the three months of the specified quarter [1, p.73] For each month, it retrieves the sales figures and compares them to the ‘highestSales’ variable. If the sales for a given month exceed the current ‘highestSales’ value, the algorithm updates ‘highestSales’ with this new value and records the month as the ‘mostEffectiveMonth’.

The crucial role of the ‘if’’ statement in this algorithm is to continuously evaluate each month’s sales figures against the current highest record. This mechanism illustrates a fundamental concept in programming where conditional statements are used to make decisions and update variables on specific criteria [1, pp.43-47].

Finally, the algorithm outputs the ‘mostEffectiveMonth’ for each department, providing valuable insights into the best-performing periods. The program ends by displaying the most effective month for each department sequentially. The choice of using a 'print' statement, as opposed to a 'return' statement, facilitates the display of each department's highest earning month one after the other. This approach ensures that the algorithm reveals the peak sales period for all departments, rather than limiting the output to just one department's best month, offering a comprehensive view that is essential for effective retail analysis [1, pp.14-15].

**References**

[1] Q. Charatan and A. Kans, Java in Two Semesters: Featuring JavaFX. in Texts in Computer Science. Cham: Springer International Publishing, 2019.

[2] A Concise and Practical Introduction to Programming Algorithms in Java. in Undergraduate Topics in Computer Science. London: Springer London, 2009. doi: 10.1007/978-1-84882-339-6.

[3] H. Schildt, "Java: The Complete Reference," McGraw-Hill Education Group, 2014.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to algorithms, Third edition. Cambridge, Massachusetts London, England: MIT Press, 2009.