**Artifact Two Narrative**

Victor Landi

September 2023

**Artifact Description:**

Artifact Type: Source code from CS 260 Algorithms and Data Structures

Language: C++ Bid management program with selection sort and quick sort, bidirectional selection sort and user-friendly features added as my enhancement.

Creation Date: The C++ version's copyright notice indicates it was created in 2017. The course CS 260 which the artifact is from was taken in February of 2022. The enhancements include a prompt for name and birthyear, prompt to load the bids and a faster selection sort method added in September of 2023.

**Justification for Inclusion in ePortfolio:**

This artifact demonstrates proficiency in software development by showcasing my ability to design and implement a bid management program improved by increasing its selection sort efficiency, adding user friendly prompts and a prompt to load bid data from a csv file before attempting to display bid data.

It highlights my ability to work with user input, data structures, different sorting methods, and menu-driven user interfaces.

For these reasons I chose this artifact for the category of Data Structures and Algorithms.

**Specific Components Demonstrating Skills:**

In both versions, the loading and sorting of CSV files containing Bid data showcase my ability to load and sort data from a CSV file and calculate and display the time taken to perform each operation.

Input handling for user data (title, fund, vehicle, amount) demonstrates skills in user interaction and data validation. The menu-driven interface showcases control flow and conditional logic skills. String manipulation for formatting and cleaning the bid amount demonstrates data manipulation abilities.

**Key Differences in the two Versions:**

The two programs, both designed for sorting bid data, exhibit several notable differences in functionality and user interaction. These differences are particularly pronounced in the handling of user input, the introduction of a new sorting algorithm, and the addition of safeguards for proper program execution.

\*\*User Interaction:\*\*

\*Program 1:\* In Program 1, the user interaction is relatively straightforward. It prompts the user to input bid data, including an identifier, title, fund, and amount. However, it lacks a personalization element, such as collecting the user's name or birth year.

\*Program 2:\* Program 2 introduces a more personalized user experience. It starts by prompting the user to enter their name and birth year. This information is then used to calculate the user's age and display a greeting message, such as "Hello [Name], you are [Age] years old." This personalization adds a human touch to the program's interface.

\*\*Sorting Algorithms:\*\*

\*Program 1:\* Program 1 employs the conventional "Selection Sort" algorithm to alphabetize the list of bids based on their titles. Selection sort is characterized by iteratively finding the minimum element and swapping it with the element in the current position.

\*Program 2:\* In contrast, Program 2 introduces an innovative sorting algorithm called "Bidirectional Selection Sort." This new algorithm aims to accelerate the sorting process by identifying both the minimum and maximum elements in each pass. It maintains two pointers, one for the left side of the unsorted portion and one for the right side, and simultaneously places the minimum and maximum elements in their correct positions. This bidirectional approach potentially enhances the sorting speed compared to traditional selection sort.

\*\*User Guidance:\*\*

\*Program 1:\* Program 1 doesn't include specific safeguards to ensure that the user doesn't attempt to display or sort bids before they have been loaded. If the user tries to do so, the program may behave unpredictably.

\*Program 2:\* To enhance user experience and program integrity, Program 2 introduces a boolean flag named `bidsLoaded.` This flag tracks whether bids have been successfully loaded from a CSV file. If the user attempts to display or sort bids before loading them, the program provides clear guidance by displaying a message like "Please load the bids first." This ensures that the user follows the proper program flow, avoiding errors and confusion.

\*\*Conclusion:\*\*

In summary, the two programs exhibit distinct differences in user interaction, sorting algorithms, and user guidance. Program 2 goes beyond the basic functionality of Program 1 by offering a more personalized experience, introducing the novel "Bidirectional Selection Sort" algorithm for potentially faster sorting, and implementing safeguards to guide users in proper program usage. These enhancements make Program 2 a more user-friendly and efficient tool for managing bid data. Pseudocode for the original and enhanced program appears at the very end of the presentation.

**Meeting Course Objectives:**

The enhancement of Artifact Two meets the course objectives related to demonstrating proficiency in programming languages, data structures, and user interaction. By enhancing the functionality and user experience in the program I demonstrate by ability to design and evaluate computing solutions that solve a given problem using algorithmic principles and computer science practices and standards appropriate to its solution. This is accomplished while managing the trade-offs involved in design choices. Carefully calculating and comparing the time differentials of each sort parameter and weighing it against performance was factored into my choice to adapt the program to utilize a bidirectional sort method. By doing so I employed strategies for building collaborative environments that enable diverse audiences to support organizational decision making in the field of computer science. By making the code enhancements for this artifact, I am also demonstrating an ability to use well-founded and innovative techniques, skills, and tools in computing practices for the purpose of implementing computer solutions that deliver value and accomplish industry-specific goals.

**Reflection on Enhancement:**

While creating and improving this artifact, I learned:

How to enhance and improve on a sort method while not compromising accuracy but reduced the search time almost by half.

The importance of user-friendly input handling and data validation for a smooth user experience.

Challenges I faced:

I had to make several modifications to arrive at a different sort method which would not return errors and would decrease the time taken to perform a selection sort. Adding the name and birthyear prompt to include all the characters entered and calculate the users present age also required time to insure 1oo accuracy and a swift response.

Overall, this artifact demonstrates my ability to work with different algorithms and data structures showcasing these skills, making the enhanced program a valuable addition to my ePortfolio. From my experience being able to be versatile is vital in any career, especially in computer science. Being flexible and able to employ a variety of methods to perform the same task allows me to adjust based on the user / client needs. I can offer alternatives and perform tasks in more than one way providing flexibility and versatility which show in the finished product.

**Pseudocode for the original program:**

START

// Define a structure to hold bid information

STRUCT Bid

STRING bidId

STRING title

STRING fund

FLOAT amount

CONSTRUCTOR Bid()

amount = 0.0

// Define functions for displaying a bid and converting string to double

FUNCTION displayBid(Bid bid)

PRINT bid.bidId + ": " + bid.title + " | " + bid.amount + " | " + bid.fund

FUNCTION strToDouble(STRING str, CHAR ch)

// Remove the character 'ch' from 'str'

str = REMOVE\_CHAR(str, ch)

// Convert 'str' to a floating-point number and return it

RETURN TO\_FLOAT(str)

// Main function

FUNCTION main()

STRING csvPath

INTEGER choice

// Process command line arguments

SWITCH ARGUMENT\_COUNT

CASE 2:

csvPath = ARGUMENT[1]

DEFAULT:

csvPath = "eBid\_Monthly\_Sales.csv"

// Define a vector to hold all the bids

ARRAY OF Bid bids

// Display menu and get user's choice

REPEAT UNTIL choice = 9

DISPLAY "Menu:"

DISPLAY " 1. Load Bids"

DISPLAY " 2. Display All Bids"

DISPLAY " 3. Selection Sort All Bids"

DISPLAY " 4. Quick Sort All Bids"

DISPLAY " 9. Exit"

INPUT choice

SWITCH choice

CASE 1:

// Load bids from CSV file

bids = loadBids(csvPath)

DISPLAY LENGTH(bids) + " bids read"

CASE 2:

// Display all bids

FOR EACH bid IN bids

CALL displayBid(bid)

END FOR

CASE 3:

// Sort bids using selection sort

CALL selectionSort(bids)

CASE 4:

// Sort bids using quick sort

CALL quickSort(bids, 0, LENGTH(bids) - 1)

CASE 9:

// Exit the program

DISPLAY "Goodbye."

END

**Pseudocode for the enhanced program:**

START

// Define a structure to hold bid information

STRUCT Bid

STRING bidId

STRING title

STRING fund

FLOAT amount

CONSTRUCTOR Bid()

amount = 0.0

// Define functions for displaying a bid and converting string to double

FUNCTION displayBid(Bid bid)

PRINT bid.bidId + ": " + bid.title + " | " + bid.amount + " | " + bid.fund

FUNCTION strToDouble(STRING str, CHAR ch)

// Remove the character 'ch' from 'str'

str = REMOVE\_CHAR(str, ch)

// Convert 'str' to a floating-point number and return it

RETURN TO\_FLOAT(str)

// Function to calculate user's age

FUNCTION calculateAge(INTEGER birthYear)

INTEGER currentYear = 2023 // Assuming the current year is 2023

RETURN currentYear - birthYear

// Main function

FUNCTION main()

STRING csvPath

STRING name

INTEGER birthYear

INTEGER age

INTEGER choice

BOOLEAN bidsLoaded = FALSE

// Display prompts and get user input

INPUT "Enter your name: " INTO name

INPUT "Enter your birth year: " INTO birthYear

age = calculateAge(birthYear)

DISPLAY "Hello " + name + ", you are " + age + " years old."

// Process command line arguments

SWITCH ARGUMENT\_COUNT

CASE 2:

csvPath = ARGUMENT[1]

DEFAULT:

csvPath = "eBid\_Monthly\_Sales.csv"

// Define a vector to hold all the bids

ARRAY OF Bid bids

// Display menu and get user's choice

REPEAT UNTIL choice = 9

DISPLAY "Menu:"

DISPLAY " 1. Load Bids"

DISPLAY " 2. Display All Bids"

DISPLAY " 3. Bidirectional Selection Sort All Bids"

DISPLAY " 4. Quick Sort All Bids"

DISPLAY " 9. Exit"

INPUT choice

SWITCH choice

CASE 1:

// Load bids from CSV file

bids = loadBids(csvPath)

bidsLoaded = TRUE

DISPLAY LENGTH(bids) + " bids read"

CASE 2:

// Display all bids

IF bidsLoaded THEN

FOR EACH bid IN bids

CALL displayBid(bid)

END FOR

ELSE

DISPLAY "Please load the bids first."

END IF

CASE 3:

// Sort bids using bidirectional selection sort

IF bidsLoaded THEN

CALL bidirectionalSelectionSort(bids)

ELSE

DISPLAY "Please load the bids first."

END IF

CASE 4:

// Sort bids using quick sort

IF bidsLoaded THEN

CALL quickSort(bids, 0, LENGTH(bids) - 1)

ELSE

DISPLAY "Please load the bids first."

END IF

CASE 9:

// Exit the program

DISPLAY "Goodbye."

END