ISLAMIC UNIVERSITY OF TECHNOLOGY (IUT)
ORGANISATION OF ISLAMIC COOPERATION (OIC)
DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

NAME: Mustafid Bin Mostafa **STUDENT ID:** 200021141

DEPARTMENT: EEE **SECTION:** A/A1

COURSE NO. EEE 4616

COURSE TITLE: Introduction to IBM PC Assembly Language

Overview:

In this assignment, the aim is to solve a problem related to input processing, sorting, and binary search. The problem involves taking input for item names and their corresponding prices, sorting the items based on their prices using the Bubble Sort algorithm, and performing a binary search to find items with a specific price.

Output:

The output of the problem will be a sorted list of items based on the price along with the corresponding item names. Then all the items having a price of more than 20$ will be displayed along with their name and price.

Solution Approach:

Input Processing: The program prompts the user to enter the total number of items and then repeatedly accepts input for item names and their prices until the maximum number of items is reached.

Sorting: After collecting the input data, the program sorts the items based on their prices using the Bubble Sort algorithm, which is a simple sorting technique suitable for the limited resources of the 8086 microprocessor.

Binary Search: Once the items are sorted, the program performs a binary search to find items with a specific price ($20 in this case) and displays the results.

Questions:

1) Between Quick Sort and Bubble Sort, Bubble Sort is generally more suitable for the architecture of the 8086 microprocessor.

Complexity: Bubble Sort has a time complexity of O(n^2), while Quick Sort has an average-case time complexity of O(n log n). While Quick Sort is more efficient on average, its recursive nature and additional overhead make it less suitable for the limited resources and stack depth of the 8086 microprocessor.

Memory Usage: Bubble Sort is an in-place sorting algorithm, meaning it operates directly on the input array without requiring additional memory. Quick Sort, on the other hand, typically requires additional memory for its recursive calls and partitioning steps, which can be a concern on a system with limited memory like the 8086.

Implementation Complexity: Bubble Sort is simpler to implement and understand than Quick Sort, especially in assembly language programming. Quick Sort involves recursive function calls and complex partitioning logic, which may be more challenging to implement efficiently on the 8086 architecture.

Overall, for the 8086 microprocessor with its limited computational power, memory, and stack depth, Bubble Sort is generally the better choice between the two algorithms. However, if efficiency is critical and the input size is relatively large, other sorting algorithms optimized for small memory footprints, such as Insertion Sort or Shell Sort, may be preferable.

2) Here's a list of different types of operations executed in the provided assembly code:

Memory Operations:

(i) Loading data from memory into registers (e.g., mov ax, @data, mov ds, ax, mov itm[di], al).

(ii) Storing data from registers into memory (e.g., mov price[si], ax).

(iii) Accessing data from arrays stored in memory (e.g., mov ax, price[si], mov bx, item_indice_2[si]).

Arithmetic Operations:

(i) Incrementing and decrementing register values (e.g., inc di, dec bx).

(ii) Performing addition and subtraction (e.g., add si, 2, sub bx, 1).

Logical Operations:

(i) Performing logical AND operations (e.g., and ax, 00FH).

(ii) Performing logical OR operations (e.g., or ax, ax).

Control Flow Operations:

(i) Unconditional jump instructions (e.g., jmp begin, jmp sort). (ii) Conditional jump instructions (e.g., je itm_inc, jne p1). (iii) Procedure calls and returns (e.g., call indec, ret).

Input/Output Operations:

(i) Reading character input from standard input (e.g., int 21h with ah=1). (ii) Writing strings to standard output (e.g., int 21h with ah=9). (iii) Printing characters (e.g., int 21h with ah=2).

Data Conversion Operations:

(i) Converting characters to integers (e.g., converting ASCII characters to decimal values in indec procedure). (ii) Converting integers to characters (e.g., converting integers to ASCII characters in outdec procedure).

Sorting Operations:

(i) Sorting the input data using the Bubble Sort algorithm (e.g., bubble procedure).

Search Operations:

(i) Searching for an item with a specific price using binary search (e.g., binary procedure).

Stack Operations:

Pushing and popping data onto/from the stack (e.g., push ax, pop bx).

3) For a sorted dataset with unique values, binary search is highly efficient with a time complexity of O(log n), where n is the number of elements in the dataset. However, if the dataset is not sorted, binary search cannot be directly applied, and sorting it first would add time complexity of O(n log n) using an efficient sorting algorithm like Quick Sort or Bubble Sort.

Regarding the specific question about finding all item names with a price of $20 using binary search:

If the dataset is sorted by price, a binary search can efficiently locate the first occurrence of an item with a price of $20. However, binary search alone cannot find all occurrences without additional modifications. Once the first occurrence is found, you would typically need to perform a linear search to find all subsequent occurrences, which would not be as efficient as a binary search for this specific task. If the dataset is not sorted by price, you would need to sort it first, which adds complexity to the process mentioned earlier.

Regarding the impact of the 8086 architecture on the efficiency of the binary search algorithm:

The 8086 architecture itself does not have a significant impact on the efficiency of the binary search algorithm, as binary search primarily relies on comparisons and arithmetic operations, which are fundamental instructions supported by the processor.

However, the limitations of the 8086 architecture, such as limited memory and processing power, can indirectly affect the efficiency of binary search. For example, if the dataset is very large and cannot fit entirely in memory, the overhead of disk I/O operations or limited memory space for sorting may impact the overall efficiency of the search algorithm. Additionally, if the dataset is too large, the binary search algorithm's recursive implementation may run into stack overflow issues due to limited stack space on the 8086 architecture.

Additional Comments:

The code demonstrates basic input/output operations, memory manipulation, and algorithm implementation in assembly language.

While the provided code accomplishes the specified task, further optimization and error handling could enhance its robustness and efficiency.

Future improvements could include implementing more efficient sorting and searching algorithms, optimizing memory usage, and adding error-checking mechanisms.