**MINI PROJECT**

**On**

**(E-Commerce: Price & Popularity Sorting)**

**In**

**Data Structure & Algorithms**

**BACHELOROF TECHNOLOGY**

**IN**

**Artificial Intelligence and Machine Learning**

SUBMITTED BY

**Anshul Jain**

**(PRN - 22070126014)**

**A red logo with a globe and text

Description automatically generated**

**Under the Supervision of**

**Dr. Kailash Shaw,**

**Associate Professor**

**SYMBIOSIS INSTITUTE OF TECHNOLOGY, PUNE – 412115**

**(A CONSTITUENT OF SYMBIOSIS INTERNATIONAL (DEEMED UNIVERSITY))**

**2023-24**

**INDEX**

1. Problem Statement (Page Number: 3)
2. Motivation (Page Number: 4)
3. Objectives (Page Number: 5)
4. Methodology of Implementation (Page Numbers: 6-7)
5. Hardware and Software Used (Page Numbers 8)
6. Dataset description (Page Number: 9)
7. Executable Code (Page Numbers: 10-21)
8. Result Analysis with Output Screen Shot (Page Numbers: 22-35)
9. Result Analysis with Edge Case Screen Shot (Page Numbers: 36-41)
10. How can we implement Symbol Table and AVL Tree in the Project? (Page Number: 42)
11. Learning Outcomes (Page Number: 43)
12. References (Page Number: 44)

**Problem Statement**

E-Commerce, short for “Electronic Commerce” refers to the online system or platform where the products are digitally organized, displayed and sold. E-Commerce emphasizes the need to manage a large list of products efficiently, ensuring that users can easily view products in order of price, search for specific products by name or category, and receive product suggestions based on the same category products.

This project aims for organizing the digitally stored products, sort them on the basis of their prices (ascending or descending) and on their popularity index. It also aims to provide a product recommendation system for the user and provide an efficient search option to the user to search any product available on the platform.

**Motivation**

Imagine that you are visiting an online shopping platform and you feel lost amidst a sea of products available on the platform with no clear organization. At such a point the customer will get puzzled by seeing such an unorganized platform and it may turn away the potential customer. The motivation behind my project is to solve this challenge. I want that my e-commerce platform will display the products to the customer in the way the customer wants it, whether it is based on the price, whether the customer wants to check the most popular product or wants a recommendation on a product which will belong to the same product category.

**Objectives**

* **Dynamic Sorting:** To provide the user the ability to view the products sorted by price, either from low to high or high to low price.
* **Popular Products:** To allow the user to view the most or least popular products available on the platform.
* **Effective Search:** To allow the user to easily search for specified products from the platform.
* **Personalized Recommendations:** To offer the user relevant product suggestions based on the category of the products they show interest in.

**Methodology of Implementation**

1. **Data Structure and Product Representation**

* **Product Structure:** Each product is represented using a structure containing attributes name, price, popularity, category and pointers left and right to point towards the left and right childs.
* **AVL Tree:** The main data structure used in the program is and AVL Tree which is a self-balancing binary search tree.

1. **Efficient Product Sorting**

* **Price-Based Sorting:** We are inserting the products in the AVL Tree based on their prices. This allows for efficient sorting of products in ascending or descending orders of their prices using in-order and reverse in-order traversals.
* **Popularity-Based Sorting:** We also sort the products based on their popularity index by storing their popularity in a temporary array and then by using qsort function.

1. **Product Insertion and Balancing**

* **Insertion:** New products are inserted in the AVL Tree based on their prices.
* **Balancing:** After each insertion, we check whether the tree is balanced or not. If not then we will rotate the tree in the right or left direction to balance it.

1. **Product Search and Recommendation**

* **Substring Search:** Customers can search for products by writing the complete product’s name or a part of the product’s name. Then this input from the customer is checked with each and every product and will return the potential products to the user.
* **Recommendations:** The program can suggest recommendations to the customer where the suggested products will belong to the product’s same category.

1. **Loading Data from CSV File**

* **Load the File:** The program can populate its product database by reading from a CSV File. Each row in the file are the attributes of a product.

1. **Menu Drive**

* **User Friendly Menu:** The program offers an interactive menu to the user which includes displaying the products as it is from the csv file, sort and then display them on the basis of their price or popularity, will show recommendations to the user from the same category products and will allow the user to search from the list. The menu will run infinite times until the user wants to exit from the menu drive.

1. **Program Completion**

* **Exiting from the program:** The program concludes when the user exits from the program.

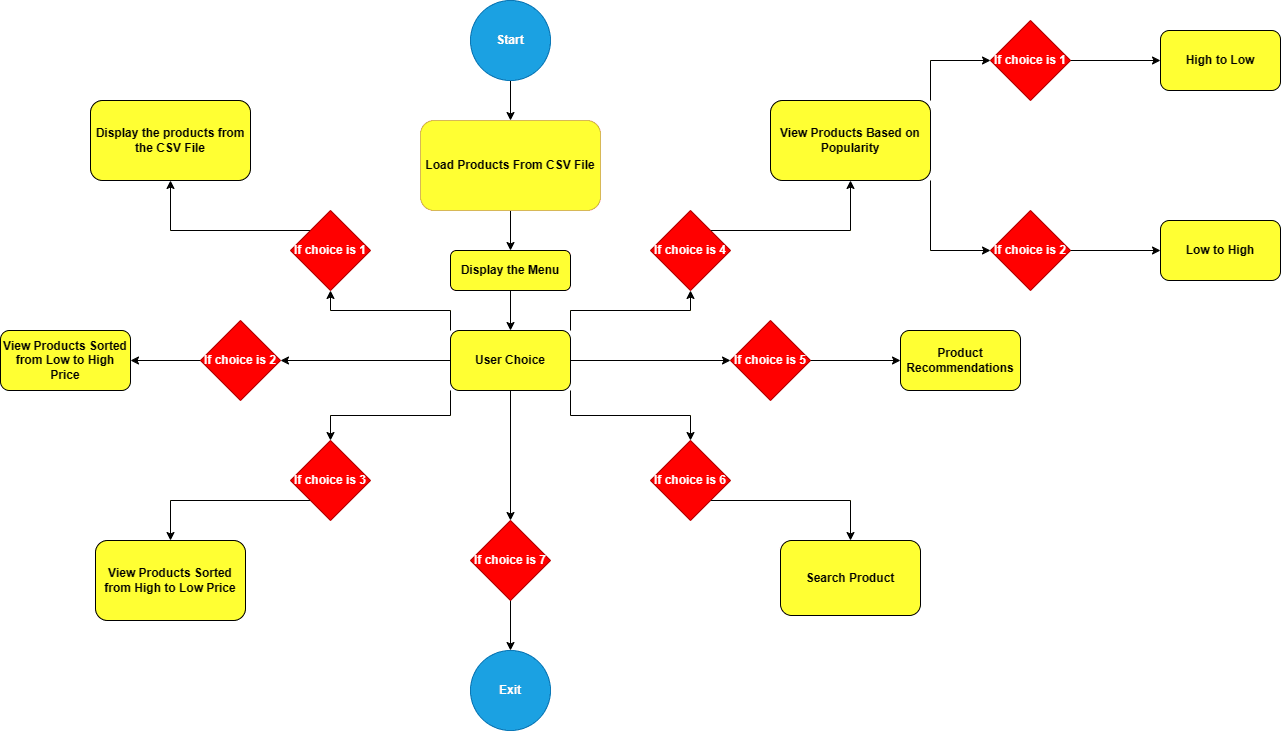


Figure 1: Flow Chart of the Project

**Hardware and Software Used**

**Hardware:**

* Processor: Intel(R) Core(TM) i3-1005G1 CPU @ 1.20GHz 1.19 GHz
* RAM: 8GB
* Device Used: Hp Laptop 15s

**Software:**

* System Type: 64-bit operating system, x64-based processor
* The project was implemented using the C programming language and compiled using GCC (GNU Compiler Collection).
* DEV C++ IDE was used for code writing, debugging and executing the source code.
* Microsoft Excel is used for creating, viewing and editing the csv file which contains the product data.
* Draw.io is used for creating the flowchart of methodology of implementation.

**Dataset Description**

In this project, we are creating a manual dataset and storing it in a csv file format. It’s difficult to capture the live data that how many times a product is getting searched on the platform and then decide the popularity of the product on the platform. That’s why I have created a manual dataset on Microsoft Excel which contains four attributes: the first is the product name, the second is the price of the product in Rs., the third attribute is the popularity of the product which lies between the range of zero to hundred, the higher the popularity number of the product is, the more it is popular on the platform, and the last, the fourth attribute is the category of the product. The fourth category involves three different types of categories:- clothing, accessories and electronics. There are a total of 100 products data in the dataset.

**Executable Code**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

struct Product {

char name[100];

double price;

int popularity;

char category[100];

int height;

struct Product \*left, \*right;

} \*root = NULL;

struct Product temp\_products[1000];

int temp\_count;

int sort\_order = 1; // 1 for High to Low, -1 for Low to High

int height(struct Product \*N) {

struct Product \*p = N;

if (p == NULL)

return 0;

return p->height;

}

int max(int a, int b) {

int p = a, q = b;

return (p > q) ? p : q;

}

struct Product\* rightRotate(struct Product \*y) {

struct Product \*p, \*q, \*t;

p = y;

q = p->left;

t = q->right;

q->right = p;

p->left = t;

p->height = max(height(p->left), height(p->right)) + 1;

q->height = max(height(q->left), height(q->right)) + 1;

return q;

}

struct Product\* leftRotate(struct Product \*x) {

struct Product \*p, \*q, \*t;

p = x;

q = p->right;

t = q->left;

q->left = p;

p->right = t;

p->height = max(height(p->left), height(p->right)) + 1;

q->height = max(height(q->left), height(q->right)) + 1;

return q;

}

int getBalance(struct Product \*N) {

struct Product \*p = N;

if (p == NULL)

return 0;

return height(p->left) - height(p->right);

}

struct Product\* insertProduct(struct Product\* node, char name[], double price, int popularity, char category[]) {

struct Product \*p, \*q;

p = (struct Product\*) malloc(sizeof(struct Product));

if (node == NULL) {

strcpy(p->name, name);

strcpy(p->category, category);

p->price = price;

p->popularity = popularity;

p->left = NULL;

p->right = NULL;

p->height = 1;

return p;

}

q = node;

if (price < q->price)

q->left = insertProduct(q->left, name, price, popularity, category);

else if (price > q->price)

q->right = insertProduct(q->right, name, price, popularity, category);

else

return q;

q->height = 1 + max(height(q->left), height(q->right));

int balance = getBalance(q);

if (balance > 1 && price < q->left->price)

return rightRotate(q);

if (balance < -1 && price > q->right->price)

return leftRotate(q);

if (balance > 1 && price > q->left->price) {

q->left = leftRotate(q->left);

return rightRotate(q);

}

if (balance < -1 && price < q->right->price) {

q->right = rightRotate(q->right);

return leftRotate(q);

}

return q;

}

void displayAllFromTemp() {

int i;

for (i = 0; i < temp\_count; i++) {

printf("Name: %s | Price: %.2lf | Popularity: %d | Category: %s\n", temp\_products[i].name, temp\_products[i].price, temp\_products[i].popularity, temp\_products[i].category);

}

}

void inOrder(struct Product\* root) {

struct Product \*q = root;

if (q != NULL) {

inOrder(q->left);

printf("Name: %s | Price: %.2lf | Popularity: %d | Category: %s\n", q->name, q->price, q->popularity, q->category);

inOrder(q->right);

}

}

void reverseInOrder(struct Product\* root) {

struct Product \*q = root;

if (q != NULL) {

reverseInOrder(q->right);

printf("Name: %s | Price: %.2lf | Popularity: %d | Category: %s\n", q->name, q->price, q->popularity, q->category);

reverseInOrder(q->left);

}

}

int comparePopularity(const void \*a, const void \*b) {

struct Product \*productA = (struct Product \*)a;

struct Product \*productB = (struct Product \*)b;

return sort\_order \* (productB->popularity - productA->popularity);

}

void inOrderStore(struct Product\* root) {

if (root) {

inOrderStore(root->left);

temp\_products[temp\_count++] = \*root;

inOrderStore(root->right);

}

}

void displayByPopularity(int order) {

temp\_count = 0; // Reset the counter

inOrderStore(root); // Fill the temp\_products array

sort\_order = order; // Set the global sorting order

qsort(temp\_products, temp\_count, sizeof(struct Product), comparePopularity); // Sort by popularity

int i;

for (i = 0; i < temp\_count; i++) {

printf("Name: %s | Price: %.2lf | Popularity: %d | Category: %s\n", temp\_products[i].name, temp\_products[i].price, temp\_products[i].popularity, temp\_products[i].category);

}

}

int containsSubstring(char\* str, char\* substr) {

int strLen = strlen(str);

int substrLen = strlen(substr);

int i,j;

for (i = 0; i <= strLen - substrLen; i++) {

for (j = 0; j < substrLen; j++) {

if (str[i + j] != substr[j]) {

break;

}

}

if (j == substrLen) {

return 1; // Found the substring

}

}

return 0;

}

struct Product\* findProductByName(struct Product\* node, char\* productName) {

struct Product \*p, \*q;

q = node;

if (q == NULL) return NULL;

if (containsSubstring(q->name, productName)) {

return q;

}

p = findProductByName(q->left, productName);

if (p) return p;

return findProductByName(q->right, productName);

}

void recommendByCategory(struct Product\* node, char\* category) {

struct Product \*q = node;

if (q == NULL) return;

if (strcmp(q->category, category) == 0) {

printf("Name: %s | Price: %.2lf | Popularity: %d | Category: %s\n", q->name, q->price, q->popularity, q->category);

}

recommendByCategory(q->left, category);

recommendByCategory(q->right, category);

}

void searchProduct(struct Product\* node, char\* searchStr) {

if (node == NULL) return;

if (containsSubstring(node->name, searchStr)) {

printf("Name: %s | Price: %.2lf | Popularity: %d | Category: %s\n", node->name, node->price, node->popularity, node->category);

}

else{

printf("Product not Found!!!\n");

return;

}

searchProduct(node->left, searchStr);

searchProduct(node->right, searchStr);

}

struct Product\* loadProductsFromCSV(const char\* filePath) {

FILE \*file = fopen(filePath, "r");

if (file == NULL) {

printf("Error: Cannot open file.\n");

return NULL;

}

char line[256];

fgets(line, sizeof(line), file);

while (fgets(line, sizeof(line), file)) {

char \*name = strtok(line, ",");

char \*price\_str = strtok(NULL, ",");

char \*popularity\_str = strtok(NULL, ",");

char \*category = strtok(NULL, ",");

size\_t len = strlen(category);

while(len > 0 && (category[len - 1] == '\n' || category[len - 1] == ' ')) {

category[--len] = '\0';

}

double price = atof(price\_str);

int popularity = atoi(popularity\_str);

root = insertProduct(root, name, price, popularity, category);

// Populate temp\_products array with the current product

strcpy(temp\_products[temp\_count].name, name);

strcpy(temp\_products[temp\_count].category, category);

temp\_products[temp\_count].price = price;

temp\_products[temp\_count].popularity = popularity;

temp\_count++;

}

fclose(file);

return root;

}

int main() {

root = loadProductsFromCSV("products.csv");

int choice;

do {

printf("\nE-Commerce Platform:\n");

printf("1. Display All Products\n");

printf("2. View Products Sorted By Price (Low to High)\n");

printf("3. View Products Sorted By Price (High to Low)\n");

printf("4. View Products Based On Popularity\n");

printf("5. Product Recommendations\n");

printf("6. Search Product\n");

printf("7. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

displayAllFromTemp(); // Display products from temp\_products

break;

case 2:

printf("\nProducts Sorted By Price (Low to High):\n");

inOrder(root);

break;

case 3:

printf("\nProducts Sorted By Price (High to Low):\n");

reverseInOrder(root);

break;

case 4:

printf("\nProducts Sorted By Popularity:\n");

printf("1. High to Low\n");

printf("2. Low to High\n");

printf("Enter Your Choice: ");

int popOrderChoice;

scanf("%d", &popOrderChoice);

if (popOrderChoice == 1) {

displayByPopularity(1);

} else if (popOrderChoice==2) {

displayByPopularity(-1);

} else {

printf("\nInvalid Input\n");

}

break;

case 5:

printf("Enter product name for recommendations: ");

char productName[100];

scanf("%s", productName);

struct Product\* foundProduct = findProductByName(root, productName);

if (foundProduct) {

printf("\nRecommendations in the category of '%s':\n", foundProduct->category);

recommendByCategory(root, foundProduct->category);

} else {

printf("Product not found!\n");

}

break;

case 6:

printf("Enter product name to search: ");

char searchName[100];

scanf("%s", searchName);

searchProduct(root, searchName);

break;

case 7:

printf("Exiting...\n");

return 1;

default:

printf("Invalid choice! Please try again.\n");

break;

}

} while(1);

return 1;

}

**Result Analysis with Output Screen Shot**

* Menu Drive

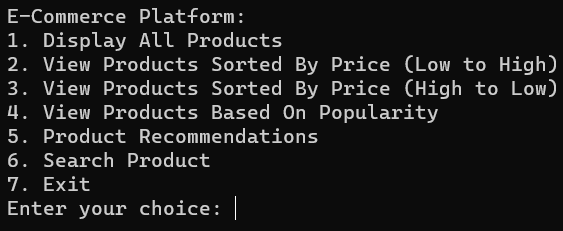


Figure 2: Menu Drive Output Screenshot

* When the user’s choice is 1 and we are displaying all the products.

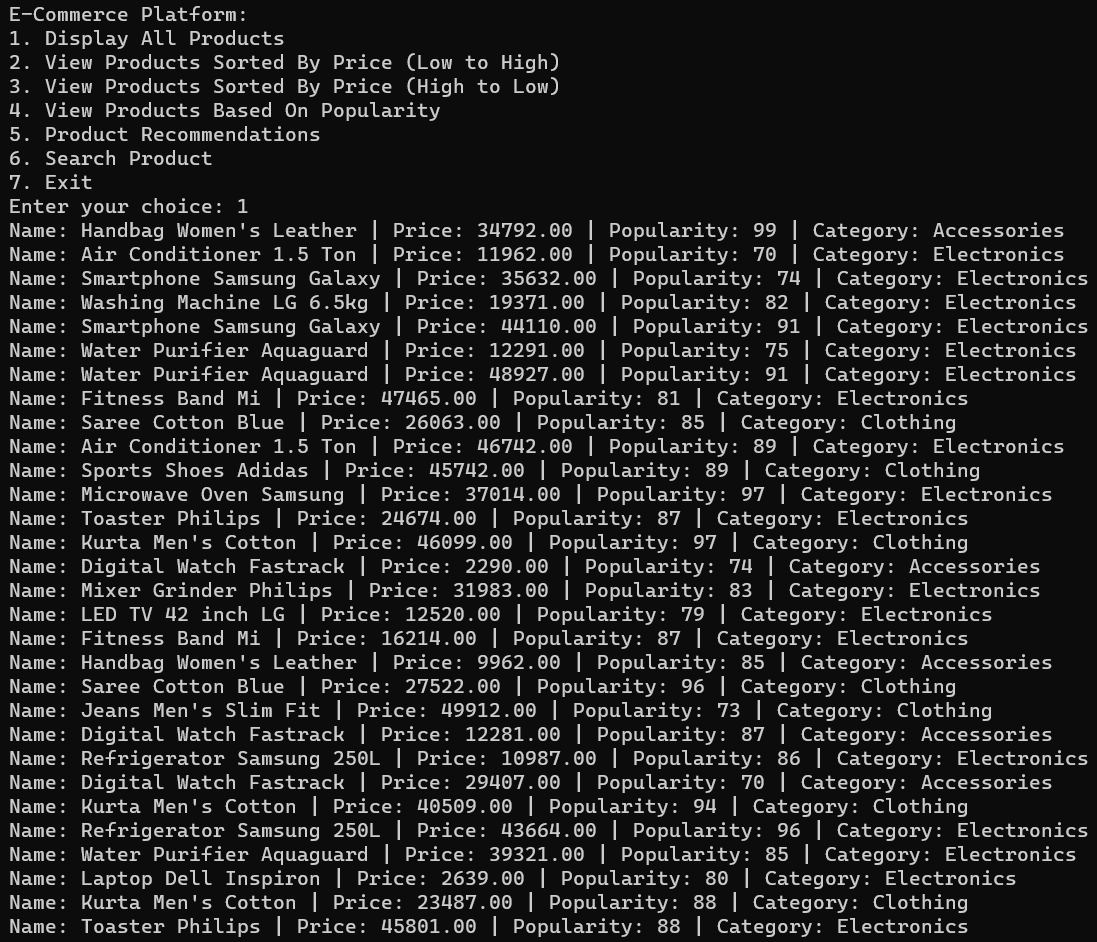


Figure 3.1: Displaying All Products Output Screenshot



Figure 3.2: Displaying All Products Output Screenshot



Figure 3.3: Displaying All Products Output Screenshot

* When the user’s choice is 2 and the user is viewing products sorted by price (Low to High).

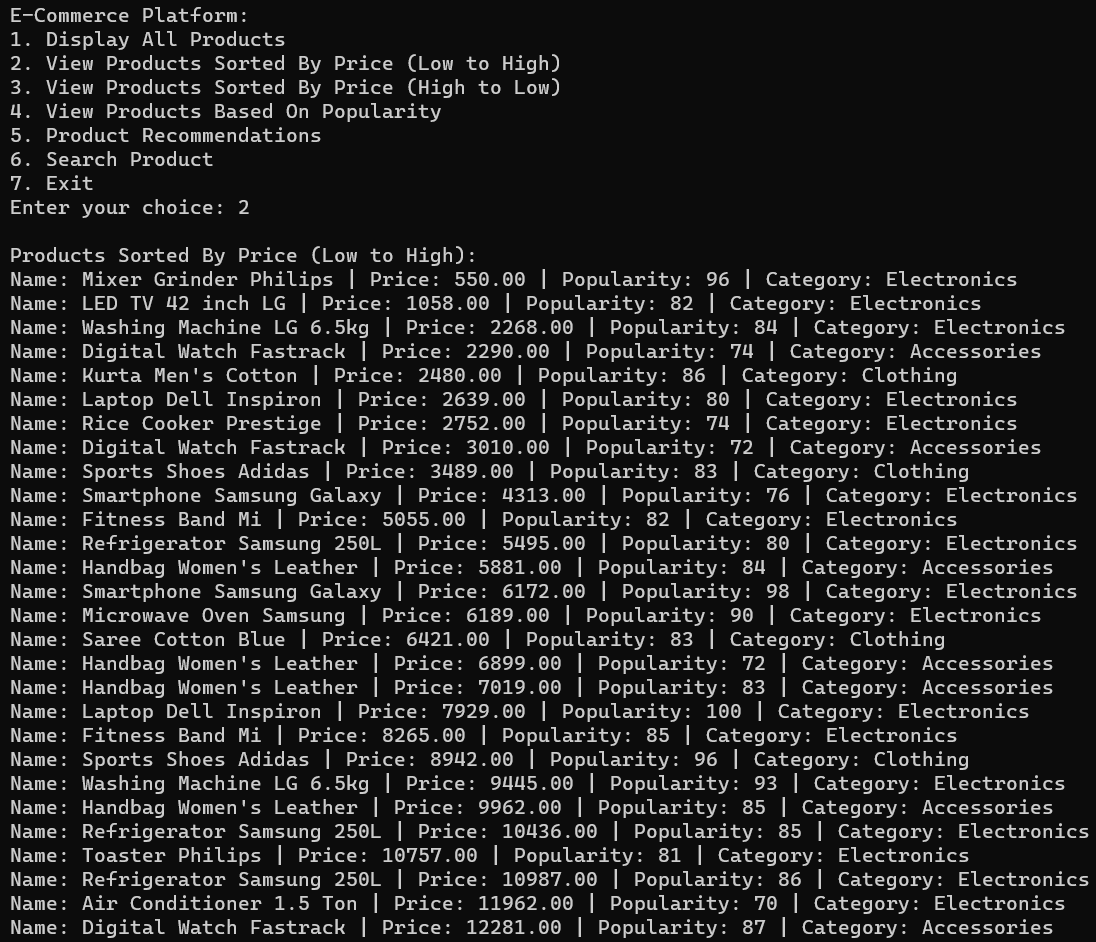


Figure 4.1: View Products Sorted by Price (Low to High) Output Screenshot



Figure 4.2: View Products Sorted by Price (Low to High) Output Screenshot



Figure 4.3: View Products Sorted by Price (Low to High) Output Screenshot

* When the user’s choice is 3 and the user is viewing products sorted by price (high to low).

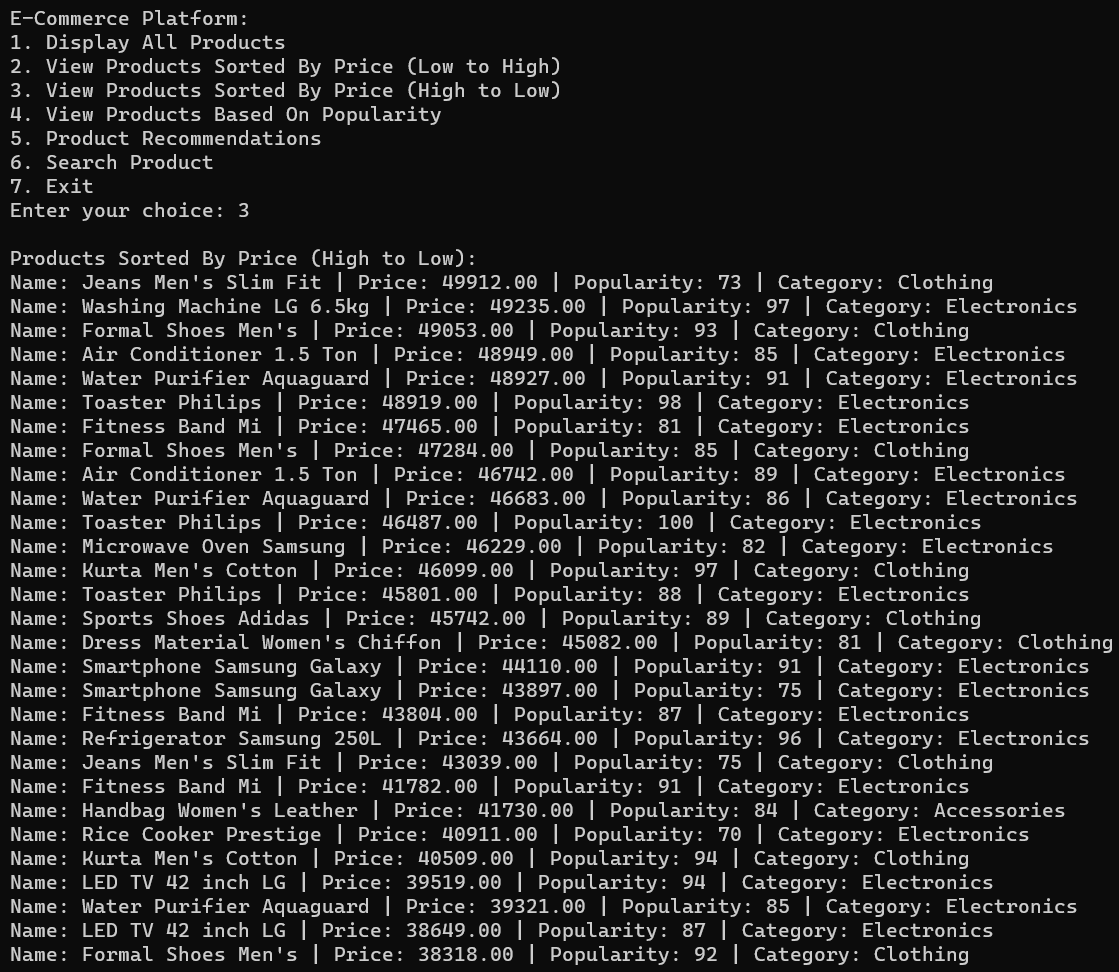


Figure 5.1: View Products Sorted by Price (High to Low) Output Screenshot



Figure 5.2: View Products Sorted by Price (High to Low) Output Screenshot



Figure 5.3: View Products Sorted by Price (High to Low) Output Screenshot

* When the user’s choice is 4 and then his choice is 1 and the user is viewing the products based on their popularity from high to low.

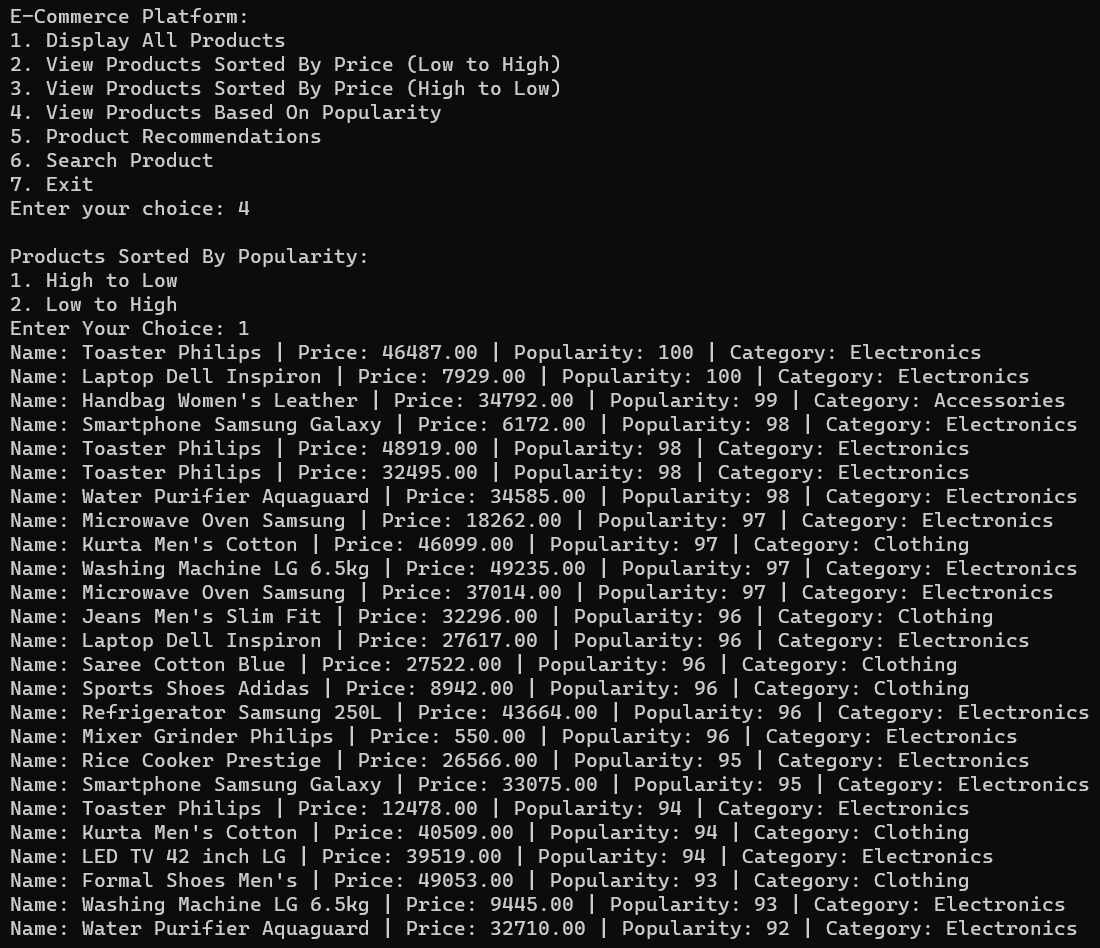


Figure 6.1: View Products Based on Popularity (High to Low) Output Screenshot



Figure 6.2: View Products Based on Popularity (High to Low) Output Screenshot



Figure 6.3: View Products Based on Popularity (High to Low) Output Screenshot

* When the user’s choice is 4 and then his choice is 2 and the user is viewing the products based on their popularity from low to high.

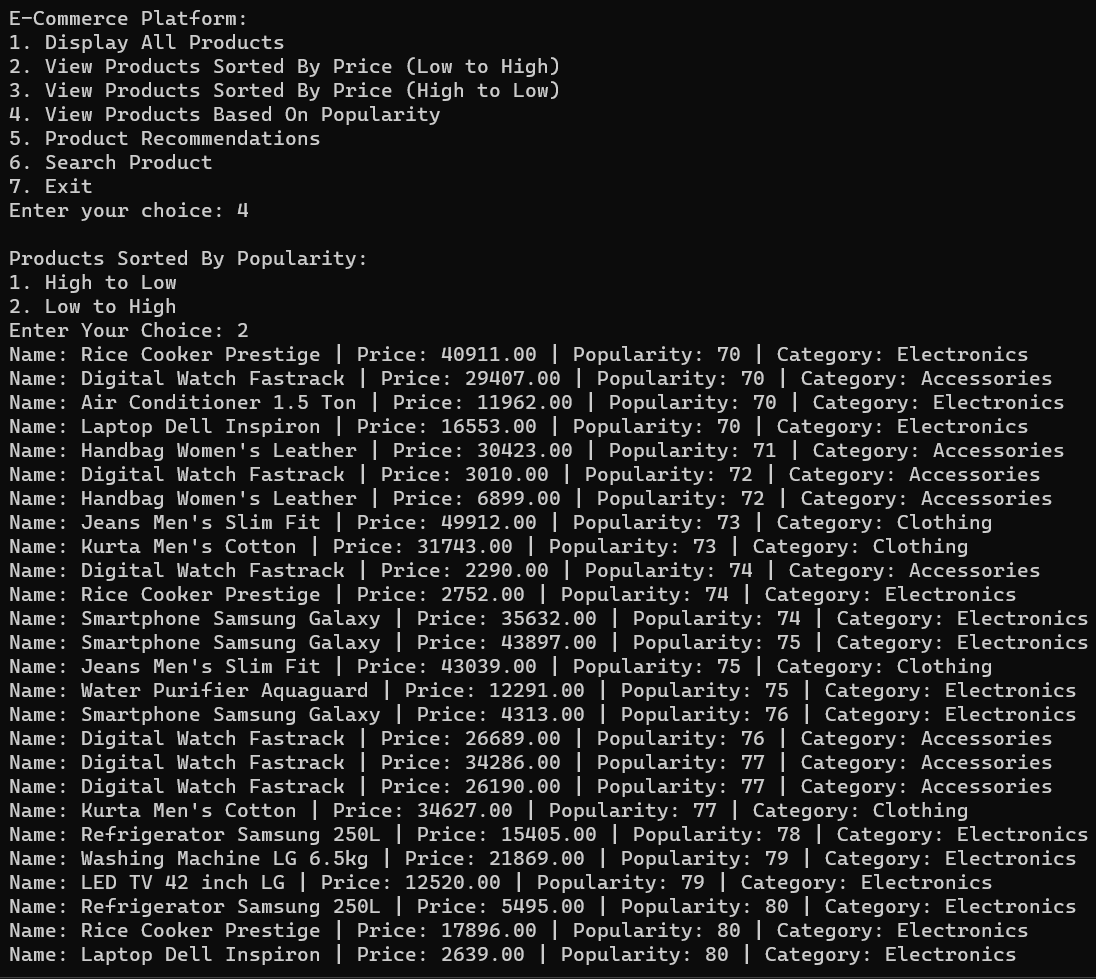


Figure 7.1: View Products Based on Popularity (Low to High) Output Screenshot



Figure 7.2: View Products Based on Popularity (Low to High) Output Screenshot



Figure 7.3: View Products Based on Popularity (Low to High) Output Screenshot

* When the user’s choice is 5 and the user is getting recommendation from the Electronics Category.

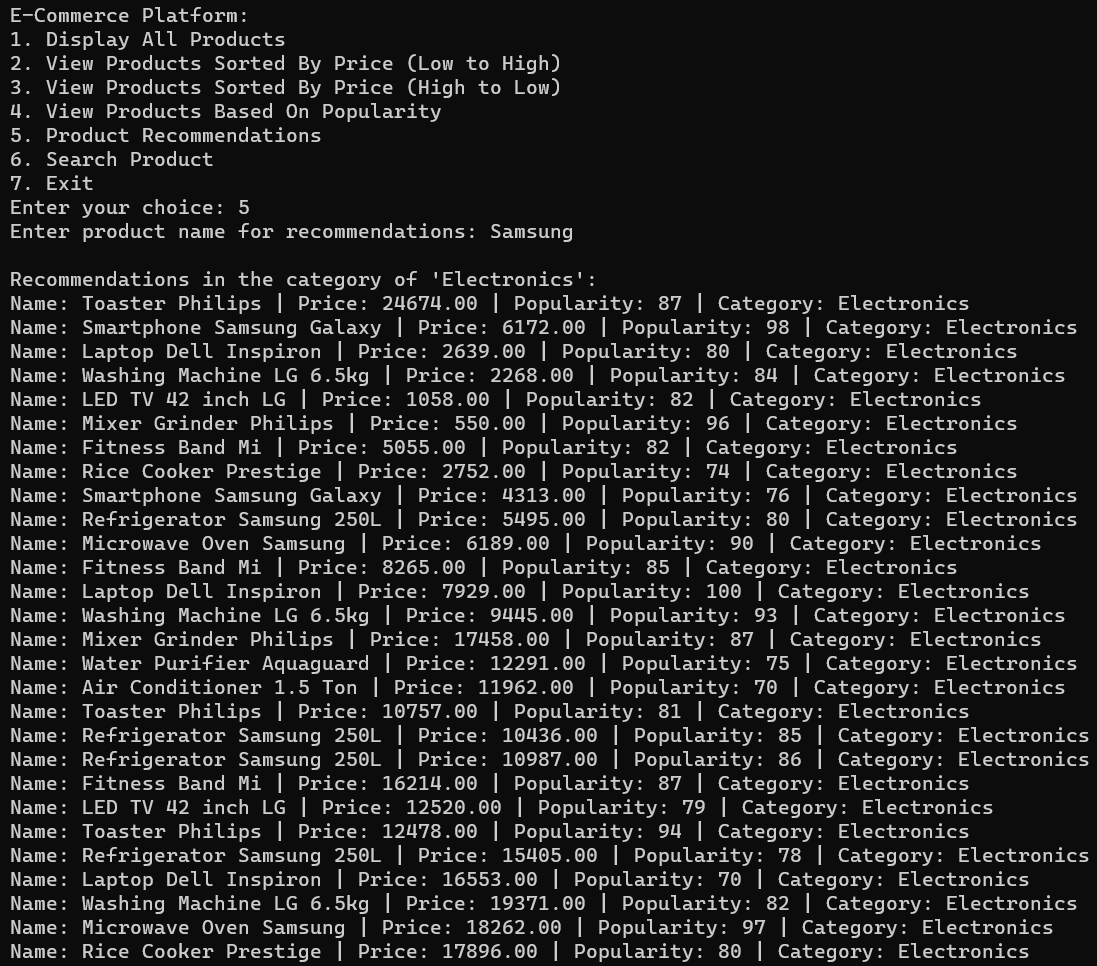


Figure 8.1: Product Recommendations from Electronics Category Output Screenshot



Figure 8.2: Product Recommendations from Electronics Category Output Screenshot

* When the user’s choice is 5 and he is getting recommendations from the Clothing category.

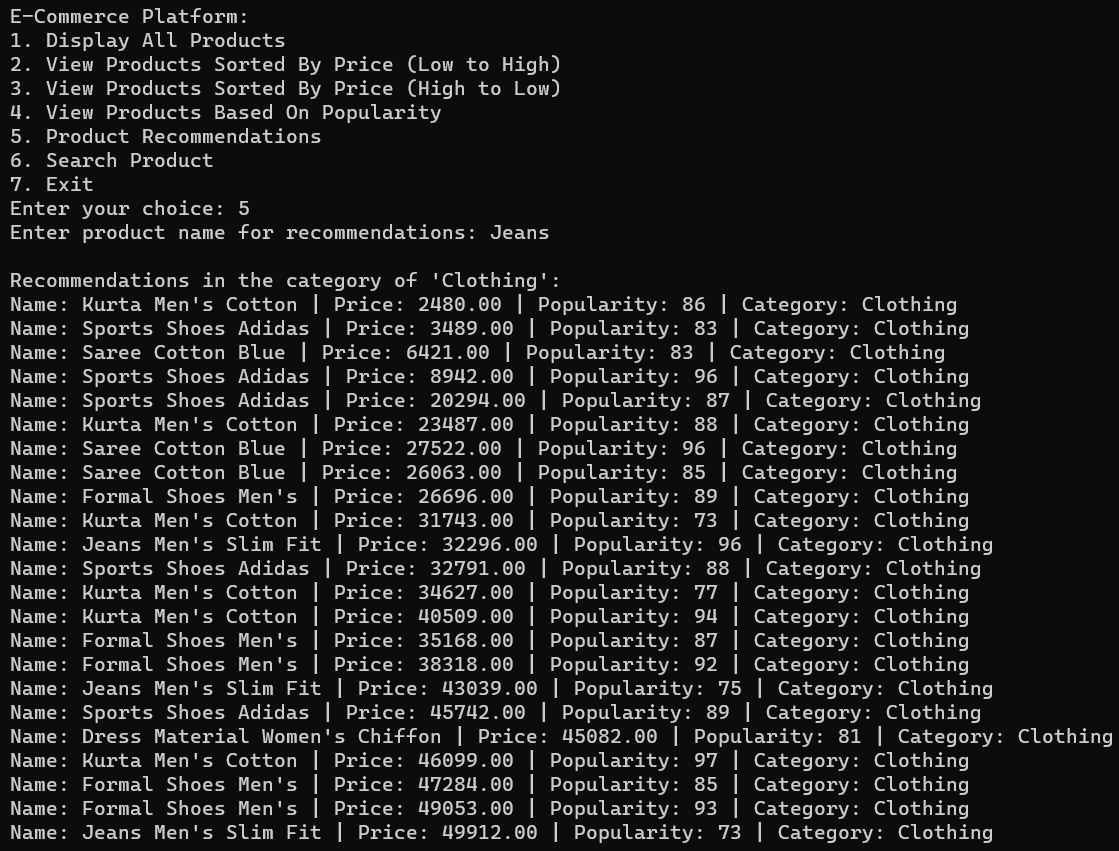


Figure 9: Product Recommendations from Clothing Category Output Screenshot

* When the user’s choice is 5 and the user is getting recommendation from the Accessories Category.

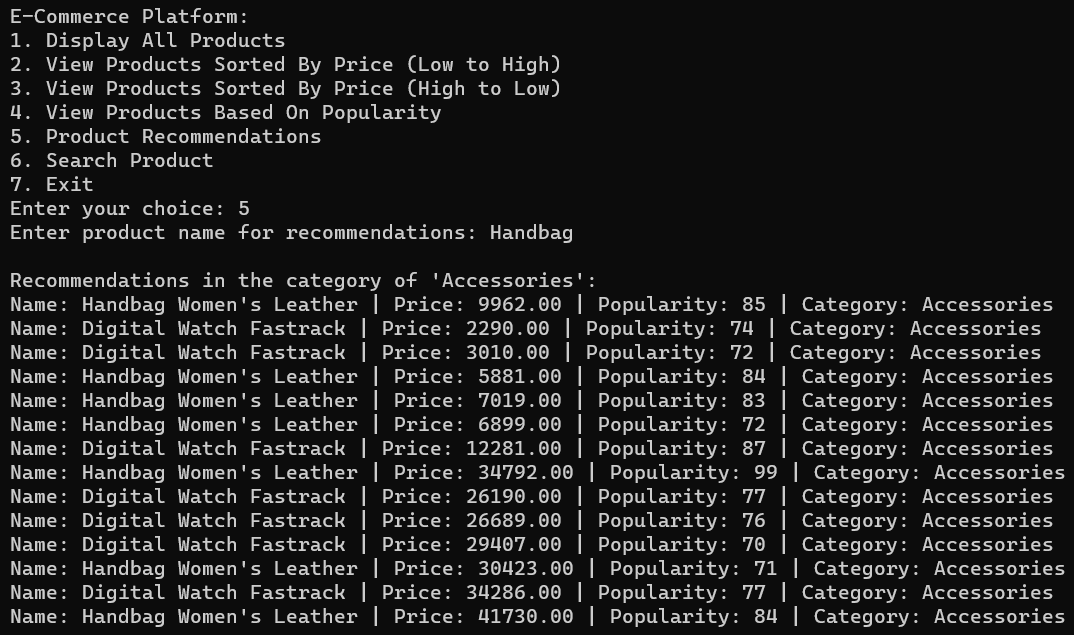


Figure 10: Product Recommendations from Accessories Category Output Screenshot

* When the user’s choice is 6, he searches for a product and the product is found and displayed on the terminal.

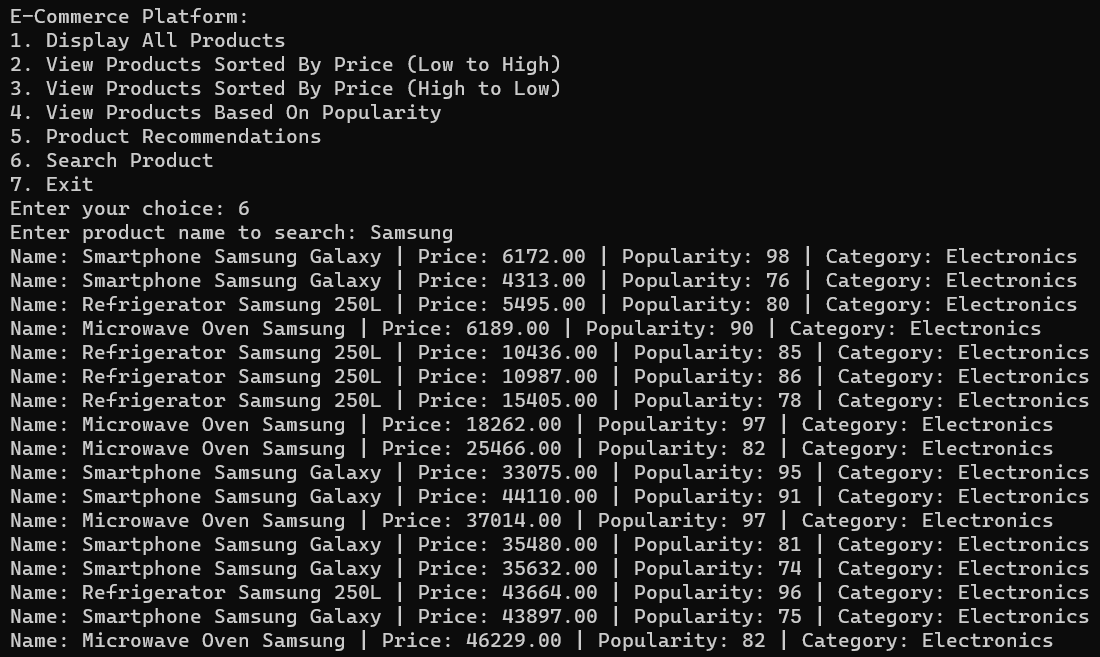


Figure 11: Searching Product and Displaying it Output Screenshot

* When the user’s choice is 7 and he is exiting from the program.

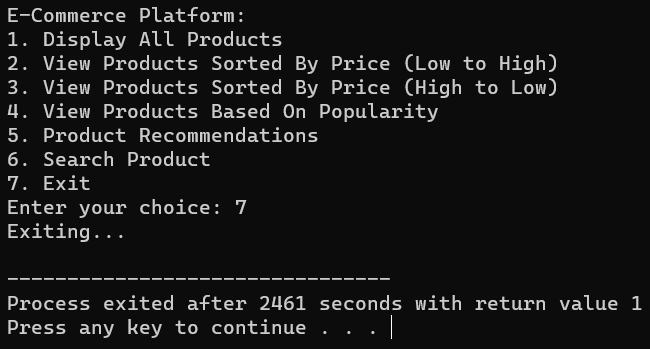


Figure 12: Exiting from the program Output Screenshot

**Result Analysis with Edge Case Screen Shot**

* When the user’s choice is invalid and gives an invalid input.

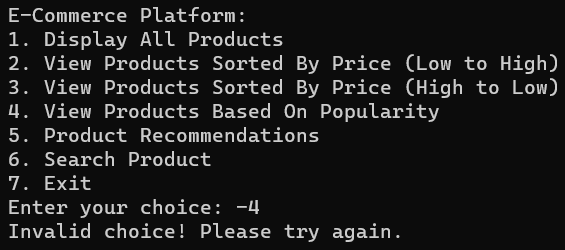


Figure 13: Invalid choice Edge Case Screenshot

* When the user’s choice is invalid and gives a non-integer (alphanumeric or string or negative float) input and the output is getting displayed infinite times.

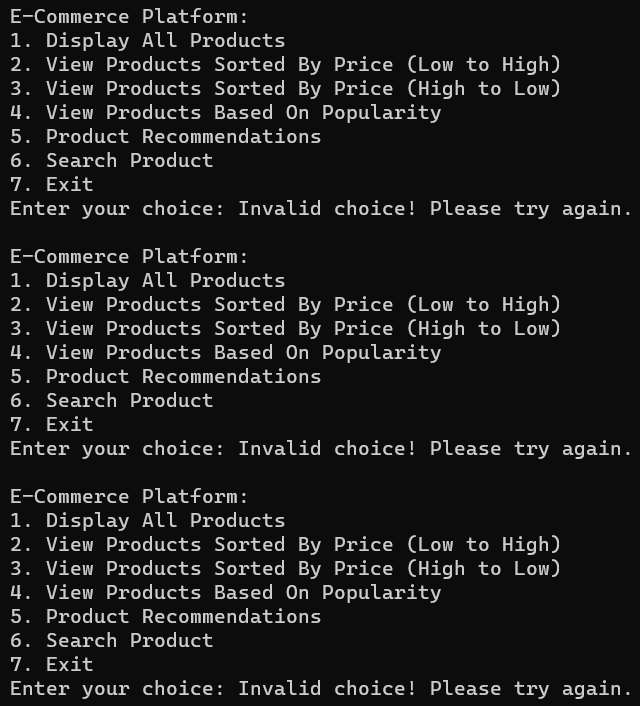


Figure 14: Non-Integer Input Edge Case Screenshot

* When the user’s choice is invalid and gives a non-integer (float) input except from 0 to 5 range in decimals.

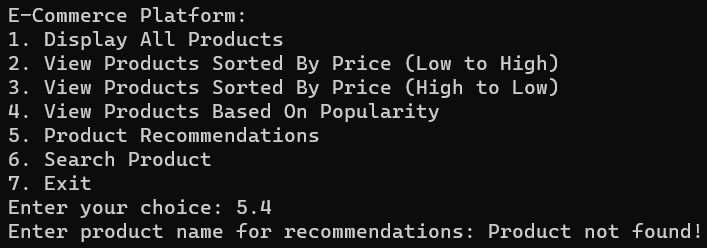


Figure 15.1: Non-Integer Input Edge Case Screenshot

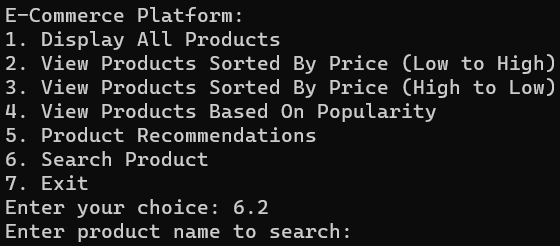


Figure 15.2: Non-Integer Input Edge Case Screenshot

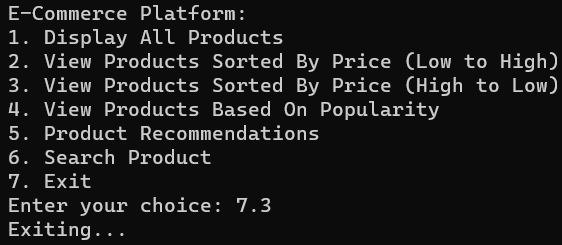


Figure 15.3 Non-Integer Input Edge Case Screenshot

* When the user’s choice is invalid and give a non-integer (float) input from the range 0 to 5 and the output is getting displayed infinite times.

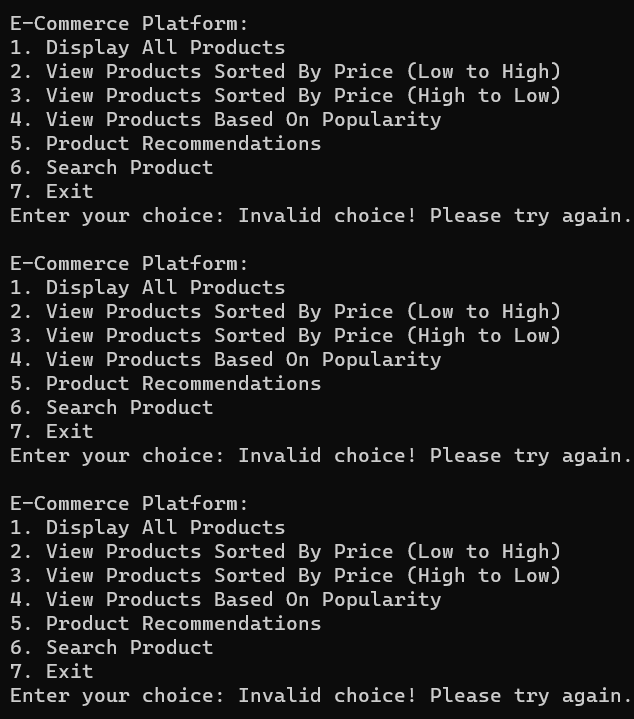


Figure 16.1 Non-Integer Input from 0 to 1 range Edge Case Screenshot



Figure 16.2: Non-Integer Input from 1 to 2 range Edge Case Screenshot

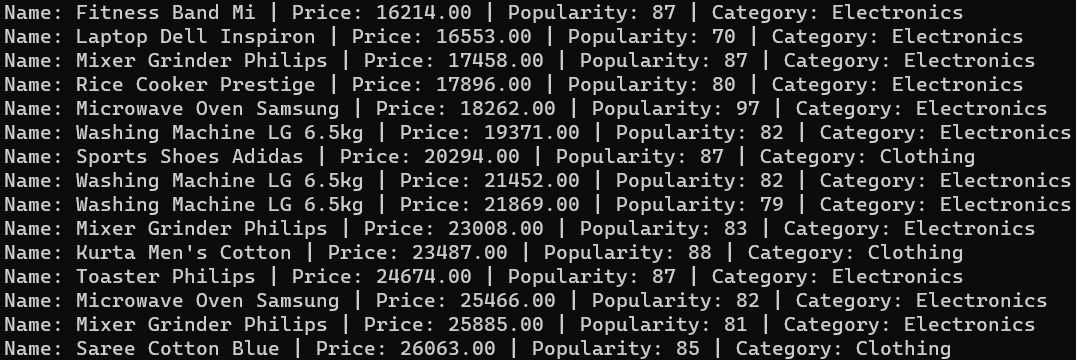


Figure 16.3: Non-Integer Input from 2 to 3 range Edge Case Screenshot



Figure 16.4: Non-Integer Input from 3 to 4 range Edge Case Screenshot

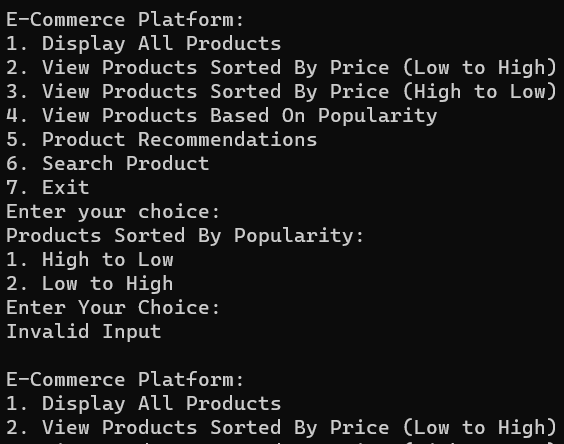


Figure 16.5: Non-Integer Input from 4 to 5 range Edge Case Screenshot

* When the user’s choice is 6 and the user is trying to search for a non-existent product from the program.

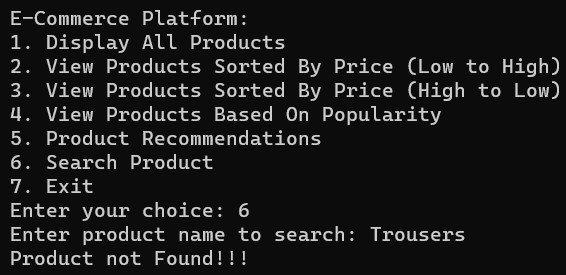


Figure 17: Search for a Non-Existent Product Edge Case Screenshot

* When the user’s choice is 5 and the user is trying to get product recommendations for a non-existent product.

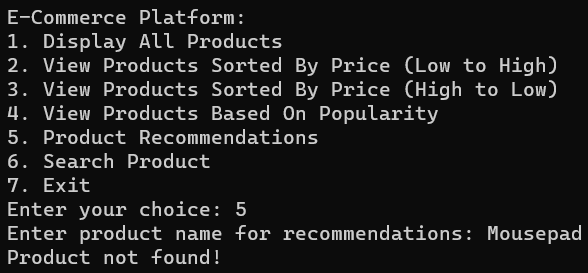


Figure 18: Product Recommendation for a non-existent product Edge Case Screenshot

* When there is no CSV file loading or a wrong file is uploaded in the program.

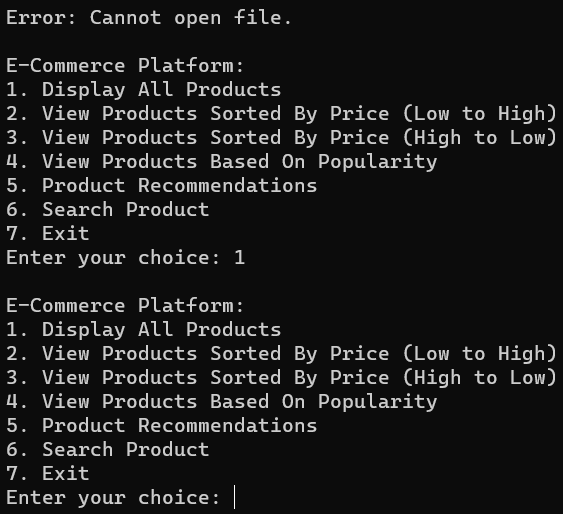


Figure 19: Cannot open file Edge Cases Screenshot

**How can we implement Symbol Table and AVL Tree in the Project?**

* A symbol table is a data structure which is used in compilers. Its primary function is to store variable names and their corresponding attributes like locations, data type etc. However, for our e-commerce project, using a symbol table is not an ideal choice. Our main goal is to maintain a list of products, search through them efficiently and balance the dataset which is stored in a binary search tree which here is an AVL Tree.
* An AVL Tree is a self-balancing binary search tree. In an AVL tree, the height difference between the left and right subtrees is not more than one. This ensures that the tree remains balanced leading to O(log n) time complexity. For our e-commerce project, we have used AVL Tree to ensure that our product list remains balanced, provides efficient search time, and optimize the sorting process for displaying the products based on their prices or popularity.

**Learning Outcomes**

* **Understanding Data Structures:** Throughout this project, I have understood the importance of AVL Tree. It is useful for optimized searching and sorting operations which are crucial for an e-commerce platform.
* **Problem Solving Skills:** This project posed some challenges such as sorting products by their prices or popularity and searching for specific products. By solving these challenges, I have made by problem solving skills and logical thinking better.
* **File Handling in C:** This project helped me to read the product data from csv file in C programming language.

**References**

**[1]**GeeksforGeeks.“AVL Tree | Set 1 (Insertion)” <https://www.geeksforgeeks.org/avl-tree-set-1-insertion/>

**[2]**Programiz.“File Handling in C” <https://www.programiz.com/c-programming/c-file-input-output>

**[3]**GeeksforGeeks.“Using strtok() in C/C++” <https://www.geeksforgeeks.org/strtok-strtok_r-functions-c-examples/>