**Data Structure and Algorithm**



Session: 2022 – 2026

**Submitted by:**

Faisal Ilyas 2022-CS-63

Gul-e-Zahra 2022-CS-75

**Supervised by:**

Sir Nazeef ul Haq

Department of Computer Science

**University of Engineering and Technology**

**Lahore Pakistan**

Table of Contents

Proposed Project Title 3

Problem Statement3

Proposed Solution 3

Description 3

Project Domain 3

Sample Scrapping URL’s 3

Attributes3

Gitlab Repository link3

Algorithms3

Sample of Scrapping Source 3

UI Components4

# **Proposed Project Title:**

**Ecommerce Data Scrapper**

**Problem Statement:**

E-commerce platforms hosts vast amounts of valuable data of products. Extracting this data efficiently for market research and business insights presents a significant challenge. Our client needs a best scraping solution to extract, structure this data for efficient use.

**Proposed Solution:**

We propose building a custom web scraping solution. Our approach involves utilizing **Python with Selenium** for automated browsing and **Beautiful Soup** for parsing and extracting data accurately. The solution will include advanced error handling, pagination support, and efficient data storage methods.

# **Description:**

An e-commerce data scraper is like a digital detective that helps to collect information from online shopping websites. It looks at the web pages of these websites and finds important details like product prices, descriptions, rating, delivery details, discount price, and customer ratings and reviews. People use these scrapers to compare prices, keep track of what’s available, and understand what customers are saying about products.

# **Project Domain**:

The project domain typically revolves around the e-commerce industry itself. It gathering information about products, prices, availability, reviews, and other related data.

# **Sample Scrapping URL’s:**

**Flipkart:**

**URL:** https://www.flipkart.com

**Amazon**

**URL:** <https://www.amazon.com>

**Daraz:**

**URL:** https://www.daraz.pk

# **Attributes:**

|  |  |
| --- | --- |
| **Attributes** | **Definition** |
| Name | Contains product name |
| Price | Indicate price of Product |
| Description | Category of Product |
| Rating | Rating given by users |
| No. of Reviews | Reviews given by the users |
| Discounted Price | Price after discount |
| Delivery | Product delivery |
| Discount Percentage | OFF % |

# **Gitlab Repository link:**

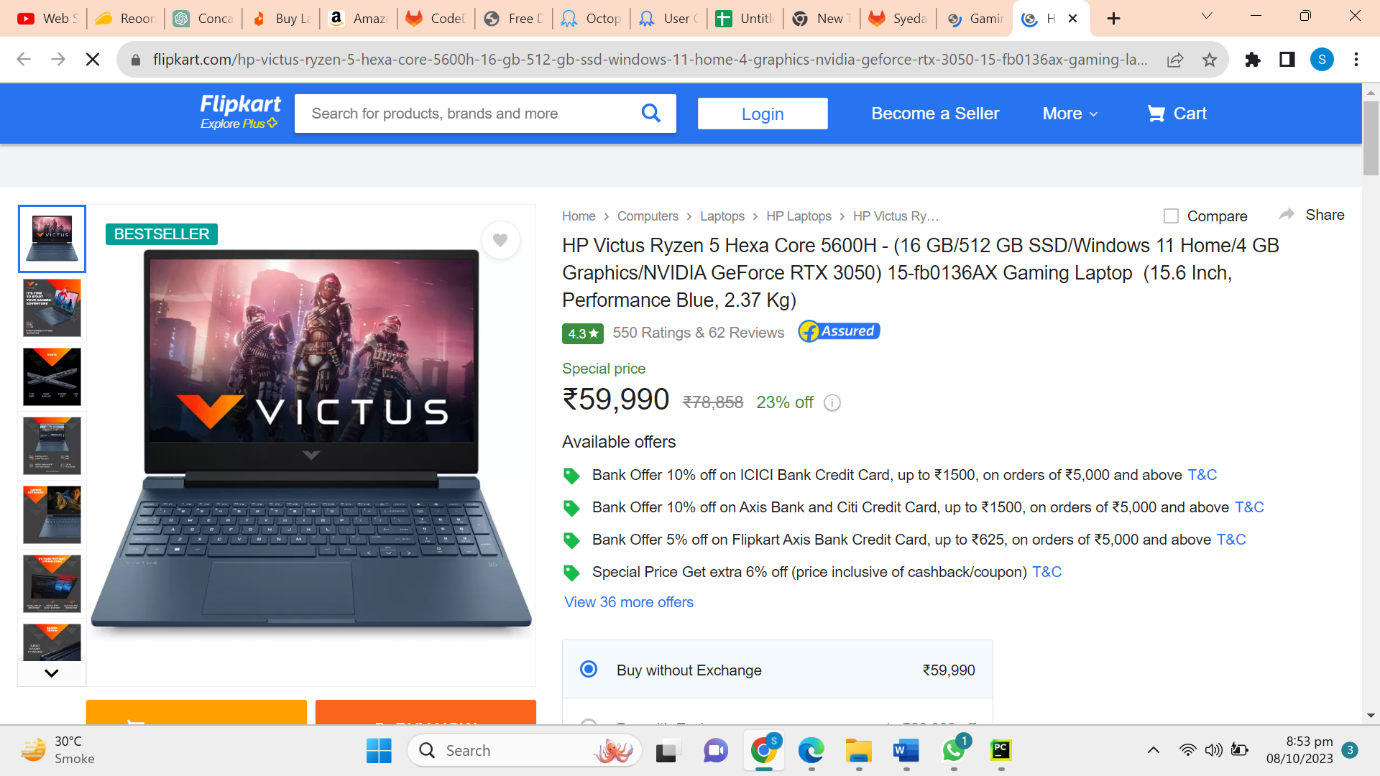
**Link**: <https://gitlab.com/faisal-ilyas/mid-porject>

# **Algorithms:**

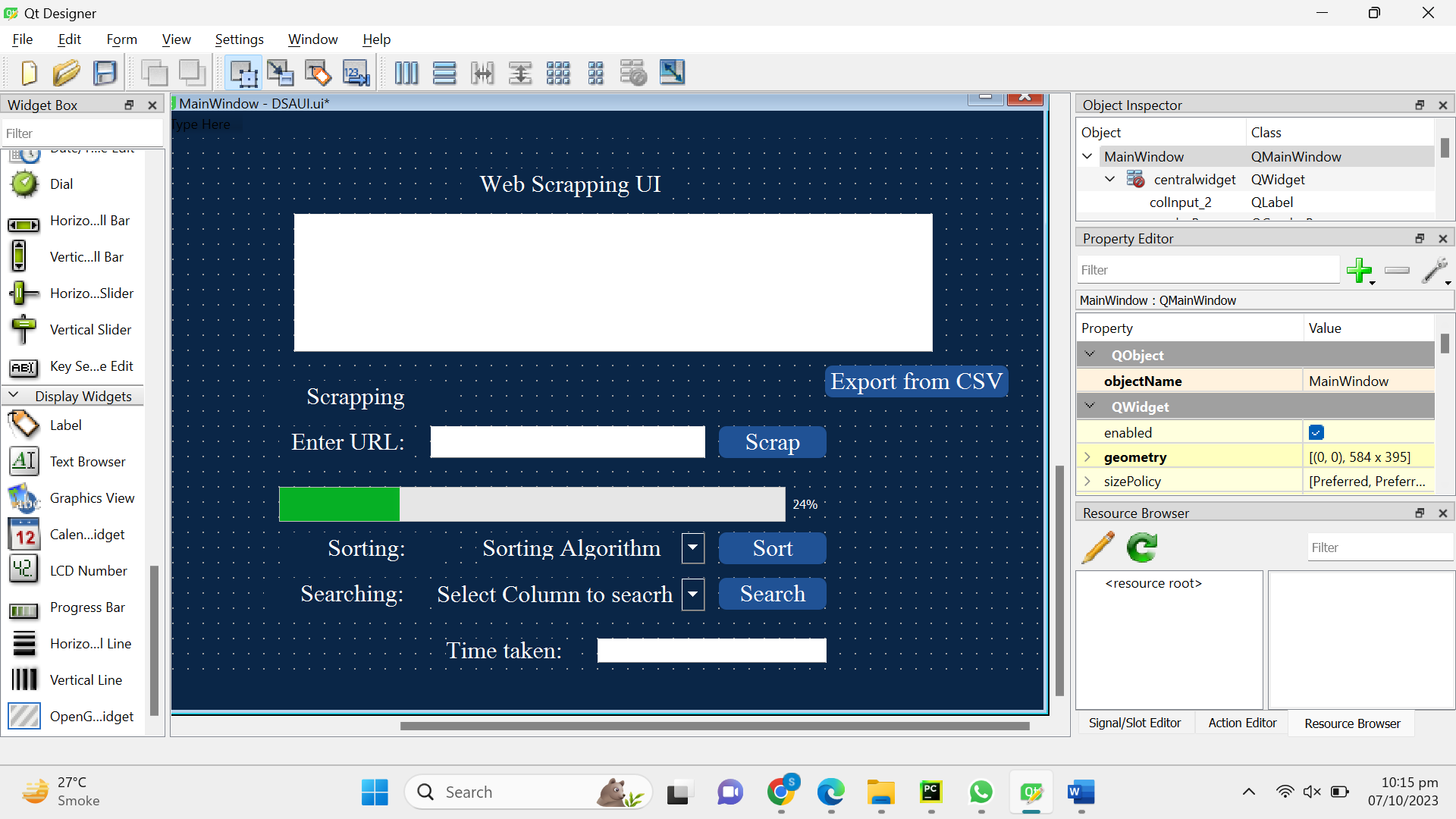
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sorting Algorithms** | | | | | |
| **Sorting Algorithm** | **Time Complexity (Average)** | **Time Complexity (Worst)** | **Space Complexity** | **Stable?** | **Short Description** |
| Bubble Sort | O(n^2) | O(n^2) | O(1) | Yes | Repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. |
| Selection Sort | O(n^2) | O(n^2) | O(1) | No | Divides the input list into two parts: a sorted sublist and an unsorted sublist. The algorithm repeatedly selects the smallest (or largest) element from the unsorted sublist and moves it to the sorted sublist. |
| Insertion Sort | O(n^2) | O(n^2) | O(1) | Yes | Builds the sorted array or list one element at a time by repeatedly picking the next element from the unsorted part and inserting it into its correct position within the sorted part. |
| Merge Sort | O(n log n) | O(n log n) | O(n) | Yes | Divides the unsorted list into n sublists, each containing one element, and repeatedly merges sublists to produce new sorted sublists until there is only one sublist remaining. |
| Hybrid Merge Sort | O(n log n) | O(n log n) | O(n) | Yes | A variation of merge sort that switches to a different sorting algorithm (like insertion sort) for small subarrays to reduce overhead. |
| Quick Sort | O(n log n) | O(n^2) | O(log n) to O(n) | No | Chooses a 'pivot' element and partitions the array or list into two sub-arrays or sub-lists according to the pivot. The sub-arrays are then sorted recursively. |
| Counting Sort | O(n + k) | O(n + k) | O(k) | Yes | Sorts integers within a specific range without comparing them. Uses counting occurrences of each element to determine its position in the output sequence. |
| Bucket Sort | O(n^2) | O(n^2) | O(n) | Yes | Distributes the elements into a number of buckets and then sorts each bucket individually, either using a different sorting algorithm or recursively applying the bucket sort. |
| Radix Sort | O(nk) | O(nk) | O(n + k) | Yes | Processes the individual digits of the numbers to sort. Sorts numbers by processing individual digits from the least significant digit to the most significant digit. |
| Heap Sort | O(n log n) | O(n log n) | O(1) | No | First converts the input array into a binary heap. It then repeatedly removes the largest element from the heap and rebuilds the heap until the heap is empty. |
| Shell Sort | O(n log n) | O(n^2) | O(1) | Yes | A variation of insertion sort that allows the exchange of items that are far apart to produce partially sorted arrays that can be efficiently sorted, eventually leading to a fully sorted array. |
| Biotonic Sort | O(log^2(n)) | O(log^2(n)) | O(n) | No | Sorts a bitonic sequence, which is an increasing sequence followed by a decreasing sequence (or vice versa). |
| Pigeonhole Sort | O(n + range) | O(n + range) | O(range) | Yes | Distributes the elements into pigeonholes (buckets) based on their values, and then uses information about the pigeonholes to individually order the elements. |
| Postman Sort | O(n + range) | O(n + range) | O(n + range) | Yes | A variation of pigeonhole sort that can handle data types other than integers. It works by distributing the elements into buckets based on their values and then sorting the buckets using another sorting algorithm. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Searching Algorithms** | | | | | |
| **Searching Algorithm** | **Time Complexity (Average)** | **Time Complexity (Worst)** | **Space Complexity** | **Efficiency** | **Comments** |
| Linear Search | O(n) | O(n) | O(1) | Inefficient | Simple search through each element sequentially. |
| Binary Search | O(log(n)) | O(log(n)) | O(1) | Efficient | Requires a sorted array. Divides the search range in half at each step. |
| Hashing (Hash Table) | O(1) (average) | O(n) (worst) | O(n) | Efficient (if hash function is well-designed) | Uses a hash function to map keys to array indices. |
| Jump Search | O(sqrt(n)) | O(sqrt(n)) | O(1) | Efficient | Requires a sorted array. Jumps ahead in fixed-size blocks. |
| Exponential Search | O(log(n)) | O(log(n)) | O(log(n)) | Efficient | Requires a sorted array. Doubles the search range at each step. |
| Interpolation Search | O(log(log(n))) | O(n) | O(1) | Efficient for uniformly distributed data | Requires a sorted array. Estimates the position of the target element. |

**Sample of Scrapping Source:**



# **UI Components:**



|  |  |  |
| --- | --- | --- |
| **UI Component Name** | **Type** | **Purpose** |
| Enter URL | Text Box | It will take URL as input to scrap. |
| Export from CSV | Push Button | By pushing this button, it will load data from CSV file to data grid. |
| Sorting Algorithm | Drop Down Menu | Display a list to select sorting algorithm. |
| Time Taken | Text Box | Display time taken by algorithm. |
| Scrap | Push Button | Press this button to scrap data from URL. |
| Sort | Push Button | Press this button to sort. |
| Select column to search | Drop Down Menu | Display list of columns to select for sorting. |
| Search | Push Button | Press this button to search. |