Gadget Glance

Submitted by: Noor Fatima (2022-CS-8)
Abeer Fatima (2022-CS-39)

Session: 2022 – 2026

Supervised by: Sir Nazeef Ul Haq

Department of Computer Science

University of Engineering and Technology Lahore, Pakistan

Contents

1 Project Description

This project is all about creating a useful tool that gathers data from online shops and simplifies product searches. We'll use Python with a user-friendly interface built using PyQt. To collect data, we'll use Selenium. Smart algorithms will keep everything organized. With this tool, users can easily grab product details from online stores, sort the data as they like, and can easily find the products they're looking for.

2 Key Features

2.1 Web Scraping Capabilities:

The application will provide web scraping functionality to collect product data from e-commerce websites, including product titles, prices, ratings, reviews, availability, discounts, shipping prices, brands, and countries.

2.2 Data Sorting:

Users can sort the scraped data based on different criteria such as price, rating, and availability. Sorting algorithms, including comparison-based sorting algorithms like insertion sort, bubble sort, selection sort, merge sort, hybrid merge sort, quicksort, heapsort, and shellsort, and non-comparison-based sorting algorithms like counting sort, radix sort, bucket sort, and pigeonhole sort, are available for data arrangement.

2.3 Multi-Sorting:

The application will support multi-sorting, allowing users to apply all the sorting algorithms mentioned above for various columns in the data.

2.4 Data Searching:

Users can perform single-level searches to find products that match specific criteria, such as product name or price range.

2.5 Multi-Level Searching:

Searching functionalities will allow users to perform multi-level searches with logical functions like AND, OR, and NOT. Users can also use other search functions, including search_with, end_with, and contains, to specify what they really want to search.

2.6 Web Scraping Capabilities:

This application includes web scraping functionalities, which allow the user to scrape data from Amazon. Users can get product information, including product names, prices, ratings, reviews, availability, discounts, shipping prices, brand details, and country names.

3 Problem Statement

Most of the time, online shoppers face problems making the right decisions when it comes to buying items. This project's goal is to make online shopping easier by creating a user-friendly tool that helps people quickly find the products they want and make smart choices. Users can analyze which product is more efficient for them. We have done this by organizing the data collected from online stores and providing a user-friendly environment to analyze things.

4 End User of Application

The targeted audience/businesses for the project is a common daily user of Amazon who can see the insights of Digital Products and much more very easily and briefly, presented in both tabular and graphical form.

5 Audience Motivation

Imagine online shopping made easy for you. Just think of an application that does all the hard work for you, gathering and organizing product information, so you can do shopping without any tension and save time. Get ready for an amazing shopping experience like never before!

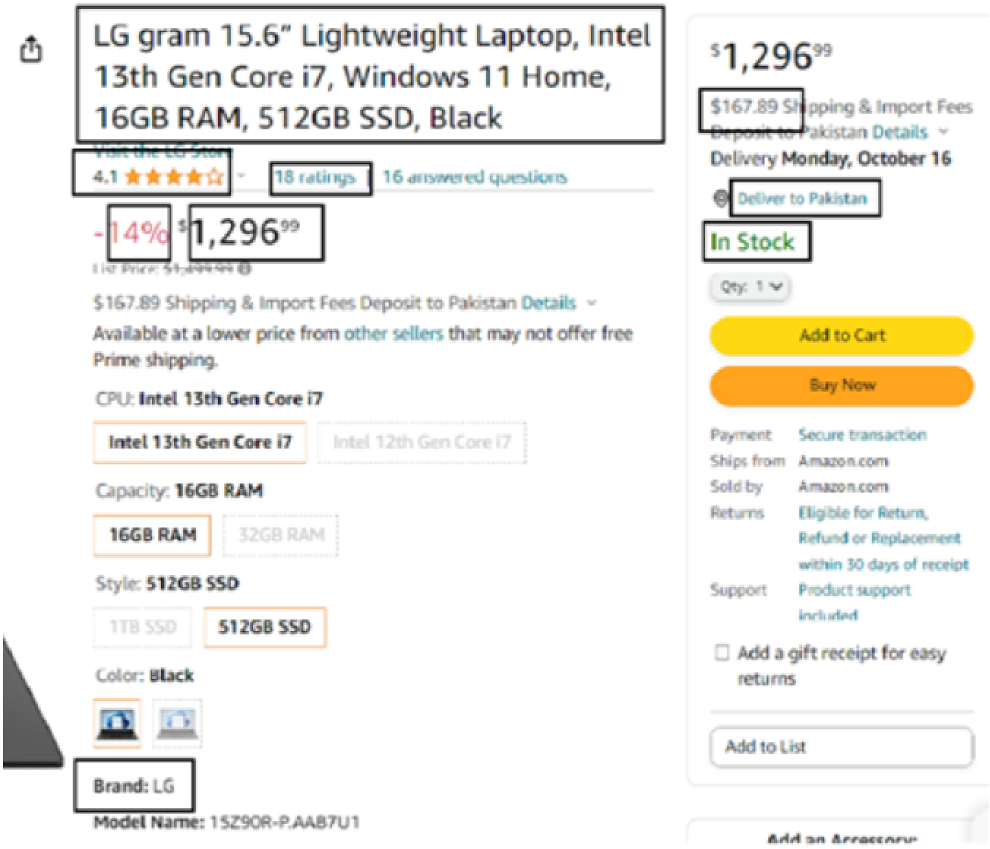
6 Technical Details

6.1 Attributes

Name	**Data Type**	**Description**
Title	String	Name of the Product
Price	Float	Total Price of Product
Discount	Float	Discount On Product
Rating	Float	Total ratings in video
Reviews	Date Type Object	Count of Reviews on the Product
Availability	String	If product is in stock or not
Shipping Price	Float	Shipping price of Product
Brand	String	Brand of Product
Country	String	In which country the product will be delivered.

6.2 Visual Representation

Picture below include all the attributes scraped. Product titles, Prices, Ratings, Reviews, Availability, Discounts, Shipping prices, Brands, and Countries.



LG gram 15.6" Lightweight Laptop, Intel 13th Gen Core i7, Windows 11 Home, 16GB RAM, 512GB SSD, Black

Visit the LG Store

4.1 ★★★★☆ | 18 ratings | 16 answered questions

-14% $^{\$}1,296^{99}$

List Price: ~~$1,499.99~~

$167.89 Shipping & Import Fees Deposit to Pakistan Details

Available at a lower price from other sellers that may not offer free Prime shipping.

CPU: **Intel 13th Gen Core i7**

Intel 13th Gen Core i7 | Intel 12th Gen Core i7

Capacity: **16GB RAM**

16GB RAM | 32GB RAM

Style: **512GB SSD**

1TB SSD | 512GB SSD

Color: **Black**

Brand: LG

Model Name: 15Z90R-P.AAB7U1

$^{\$}1,296^{99}$

$167.89 Shipping & Import Fees Deposit to Pakistan Details

Delivery **Monday, October 16**

Deliver to Pakistan

In Stock

Payment	Secure transaction
Ships from	Amazon.com
Sold by	Amazon.com
Returns	Eligible for Return, Refund or Replacement within 30 days of receipt
Support	Product support included

□ Add a gift receipt for easy returns

Figure 1: Scraped Attributes

7 Business Details

7.1 Overview

Online shopping for gadgets like laptops, watches, and computers can be confusing. We're here to make it easy. Our project will be a user-friendly platform where you can quickly compare different devices and find what's best for you. Think of it like a super-smart shopping assistant. We're inspired by popular websites like AliBaba and Daraz, which make shopping easy. With this project, you can see product details and read user reviews. We're on a mission to make your online gadget shopping experience amazing. It's all about helping you make confident and provode ease when buying digital devices or any other product online.

7.2 For Audience

Our project, "GadgetGlance," is dedicated to revolutionaize the digital shopping experience for online consumers.Our focus is on digital devices like laptops, watches, and computers, Our aim id to address the challenges faced by online shoppers. Here's a closer look at our business details:

1. **User-Centric Approach:** Our project is designed with the user in mind. We prioritize the ease of use, making it accessible to a wide range of online shoppers.
2. **Comparative Analytics:** Our platform allows users to comapare or test products based on various criteria, ensuring they find the device that best suits their needs.
3. **Empowering Decisions:** We understand the importance of confidence when making online purchases.That's why we have created this amazing tool.We enpower users to make satisfying decisions by seeing other users reviews about products and by comparing one product with other.
4. **Revolutionizing Shopping:** Similar to other platforms like AliBaba and Daraz, we aim to revolutionize the digital shopping experience. Our goal is to turn the online gadget shopping into an enjoyable process.
5. **Audience-Centric Service:** Our service targets everyday consumers who shop for digital devices online.

7.3 Business Overview for Developers

GadgetGlance is a valuable resource for developers. Here's how you can use GadgetGlance to your advantage:

1. **Data for Your Apps:** You can use GadgetGlance to get real-time data from online stores and put it in your own apps. So, if you're building something like a price comparison tool or a product reviewing app, GadgetGlance provides the data you need.
2. **Custom Sorting:** You have the freedom to use and even create your sorting methods. If you have a specific way you want to sort data, GadgetGlance can handle it.
3. **Advanced Searching:** GadgetGlance makes searching super smart. You can build powerful search engines for your users.
4. **Collaborate with Us:** We're all about teamwork. If you have ideas to make GadgetGlance better, join us on GitLab and help us build a great platform.

With GadgetGlance, you have a lot of data to create cool apps, improve user experiences, and make smart choices. We welcome you to explore what's possible and be part of our developer community.

7.4 Conclusion

In a world of digital devices and online shopping, GadgetGlance simplifies product searches, provides comprehensive insights, and empowers users. With data scraping, custom sorting, and advanced searching, it revolutionizes the online shopping experience. Welcome to a smarter way of making well-informed choices.

8 Scraping UI

The UI provides control over the web scraping process, allowing you to start, pause, resume, and stop the scraping. It also shows the progress of scraping through the progress bar, and you can choose to resume, pause, and stop the process if needed.

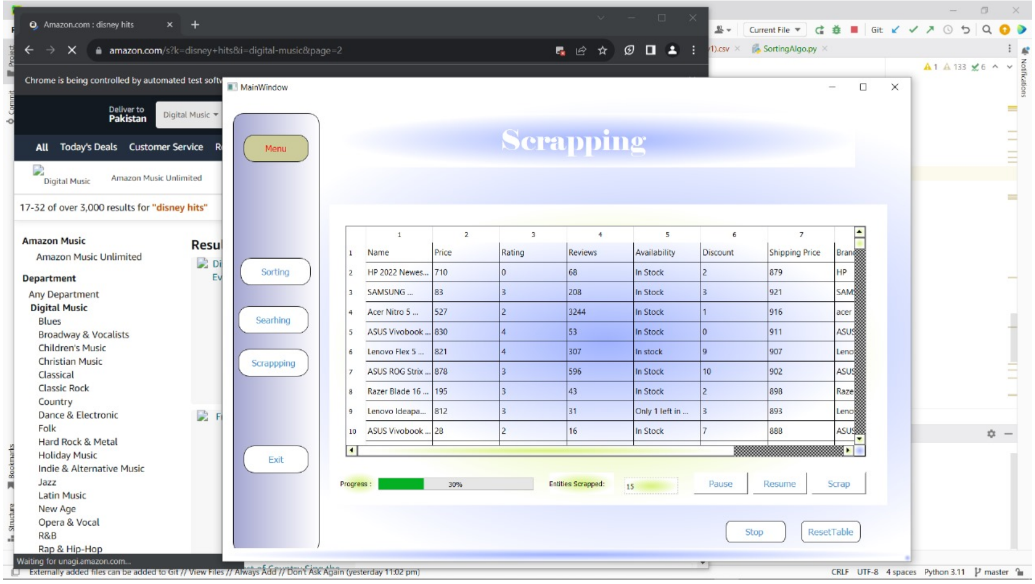
1. **Scrap Button:** When you click the "Scrap" button, the web scraping process begins. It initiates the following actions:
   - Initializes a web driver for scraping web pages (Chrome in this case).
   - Navigates to the URLs provided in the URL list.
   - Retrieves data from the web pages.
   - Extracts information like titles, prices, ratings, reviews, etc.

- Saves the extracted data to a CSV file (either creating a new one or appending to an existing file).
- Updates the progress bar to show the scraping progress.
- The data is added to the table as 1 page of the URL is scrapped.

2. **Pause Button:** Clicking the "Pause" button pauses the scraping process. This is useful if you want to stop scraping without stopping it completely.
3. **Resume Button:** Clicking "Resume" continues the paused scraping process.
4. **Stop Button:** The "Stop" button stops the scraping process entirely. This involves quitting the web driver and stopping data scraping.
5. **Progress Bar:** The progress bar visually displays the progress of the scraping process. It represents the URLs that are scraped in percentage.
6. **Show Sorting Page Button:** This button allows you to switch from the scraping page to the data sorting page. It hides the scraping page's UI and displays the sorting page.
7. **Close Application Button:** Clicking this button exits the entire application.

9 Proposed UI

9.1 Scrapping UI



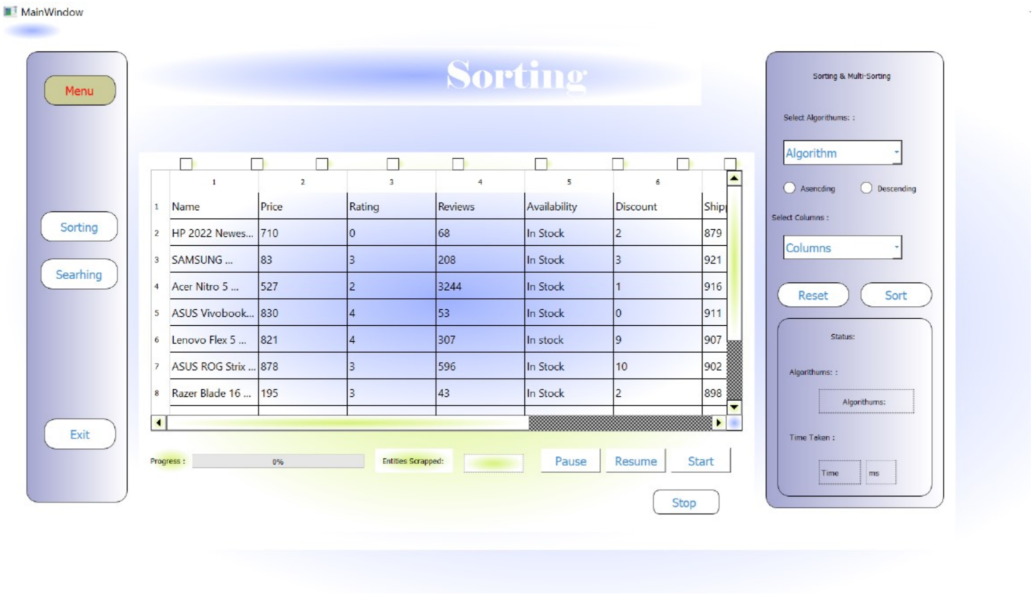
	1	2	3	4	5	6	7	
1	Name	Price	Rating	Reviews	Availability	Discount	Shipping Price	Bran
2	HP 2022 Newes...	710	0	68	In Stock	2	879	HP
3	SAMSUNG ...	83	3	208	In Stock	3	921	SAM
4	Acer Nitro 5 ...	527	2	3244	In Stock	1	916	acer
5	ASUS Vivobook ...	830	4	53	In Stock	0	911	ASUS
6	Lenovo Flex 5 ...	821	4	307	In stock	9	907	Leno
7	ASUS ROG Strix ...	878	3	596	In Stock	10	902	ASUS
8	Razer Blade 16 ...	195	3	43	In Stock	2	898	Raze
9	Lenovo Ideapa...	812	3	31	Only 1 left in ...	3	893	Leno
10	ASUS Vivobook ...	28	2	16	In Stock	7	888	ASUS

Figure 2: Scraped Attributes

Name	Type	Description
scrap_progress_bar	Progress Bar	Shows the details of scrap operation from the website.
Pause	Button	Pause the scrap operation.
Resume	Button	Resumes the scrap operation.
Stop	Button	Stops the scrap operation.
Scrap	Button	Start the scrap operation.

Table 1: Scrapping Page detail

9.2 Sorting UI



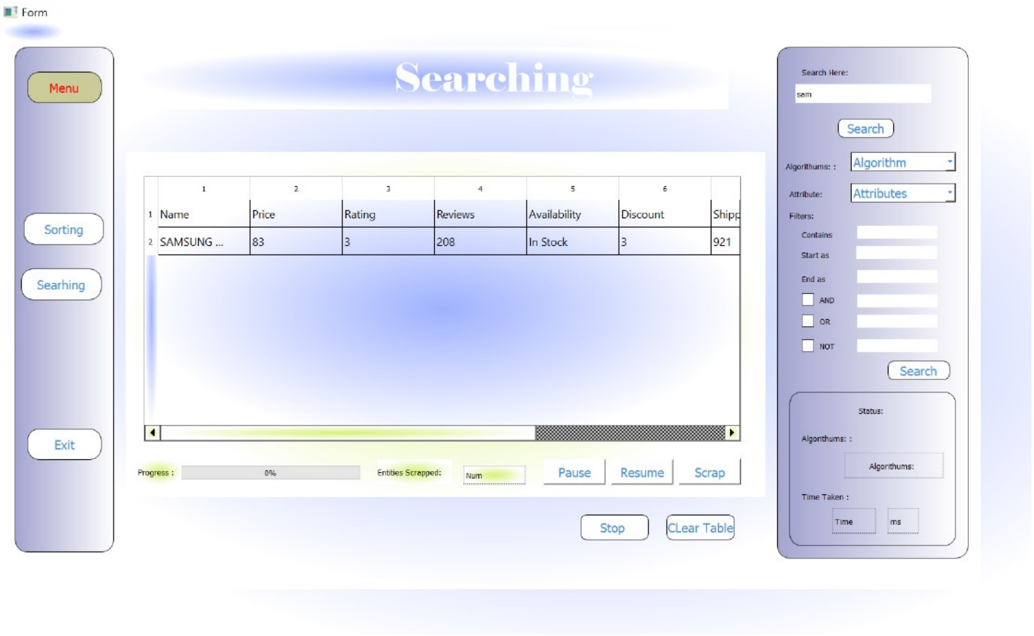
	1	2	3	4	5	6	
	□	□	□	□	□	□	□
1	Name	Price	Rating	Reviews	Availability	Discount	Ship
2	HP 2022 Newes...	710	0	68	In Stock	2	879
3	SAMSUNG ...	83	3	208	In Stock	3	921
4	Acer Nitro 5 ...	527	2	3244	In Stock	1	916
5	ASUS Vivobook...	830	4	53	In Stock	0	911
6	Lenovo Flex 5 ...	821	4	307	In stock	9	907
7	ASUS ROG Strix ...	878	3	596	In Stock	10	902
8	Razer Blade 16 ...	195	3	43	In Stock	2	898

Figure 3: Sorting UI

Name	Type	Description
Sort	Button	Starts the sorting operation
Column	Combo Box	Choose the column for search
Contains	Text Edit	Contains the term for search
Algorithm	Combo Box	List of Algorithms for sorting
Attribute	Combo Box	List of Attribute for user to sort
X milliseconds as of now	Label	Gives the time of search operation
Table Header	Check box	contains check boxes to select columns for multi sorting
Table for entities	Table	Holds and displays Entities
Menu Button	Push Button	Navigate to welcome page
Sort Button	Push Button	Navigate to sorting page
Search Button	Search Button	Navigate to searching page
Scrape Button	Push Button	Navigate to scraping page
Exit Button	Push Button	To close Application
Horizontal scroll bar	Scroll bar	Horizontal scrolling
Vertical scroll bar	Scroll bar	Vertical scrolling

Table 2: Sorting Page detail

9.3 Searching UI



	1	2	3	4	5	6	
1	Name	Price	Rating	Reviews	Availability	Discount	Shipp
2	SAMSUNG ...	83	3	208	In Stock	3	921

Figure 4: Searching UI

Name	Type	Description
Search	Button	Starts the searching operation
Column	Combo Box	Choose the column for search
Contains	Text Edit	Contains the term for search
Search Line	Text Edit	Used to add text user want to search
Starts with	Text Edit	Term for search starts with
Ends with	Text Edit	Term for search ends with
AND	Check Box + Text Edit	Composite Filter for search
OR	Check Box + Text Edit	Composite Filter for search
NOT	Check Box + Text Edit	Composite Filter for search
Algorithm	Combo Box	List of Algorithms for searching
Attribute	Combo Box	List of Attribute for user to search
X milliseconds as of now	Label	Gives the time of search operation
Table for entities	Table	Holds and displays Entities
Menu Button	Push Button	Navigate to welcome page
Sort Button	Push Button	Navigate to sorting page
Search Button	Search Button	Navigate to searching page
Scrape Button	Push Button	Navigate to scraping page

Table 3: Searching Page detail

10 Sorting Algorithm

10.1 Comparison Based

The data we scrapped from the Amazon website is first Loaded in a data table that we made in QT Designer using PQT-5. A total of 9 entities are scrapped and the data table has 9 columns; Name, Price, Rating, Reviews, Availability, Discount, Shipping Price, Brand, and Country. On these attributes, we apply different sorting and searching algorithms. In sorting there are two levels of sorting, Single Level, and Multi Level.

10.1.1 Single Level Sorting

In single-level sorting one column is selected and sorted, and on the base of that column whole data is sorted.

10.1.2 Multi Level Sorting

In Multi-level sorting, first, we select a column, sort the data on that base then we select another column and sort that sorted data on the base of that column.

10.2 Algorithums

In total, we used 12 algorithms. Out of them, 8 comparison-based sorting algorithms that we used for sorting are;

- Bubble Sort
- Insertion Sort
- Selection Sort
- Merge Sort
- Hybrid-Merge Sort
- Quick Sort
- Shell Sort
- Heap Sort

Their description, How we used them, and their pseudo-code is given bellow.

Algorithm Name	Description
Merge Sort	The Merge Sort algorithm is a sorting algorithm that is based on the Divide and Conquer paradigm. In this algorithm, the array is initially divided into two halves and then they are combined in a sorted manner.

Code

```python

def merge(left_index, mid_index, right_index, data, column_index, ascending=True):
    merged_result = []
    i = left_index
    j = mid_index + 1

    while i <= mid_index and j <= right_index:
        key1 = clean_value(data[i][column_index])
        key2 = clean_value(data[j][column_index])

        if ascending:
            if key1 < key2:
                merged_result.append(data[i])
                i += 1
            else:
                merged_result.append(data[j])
                j += 1
        else:
            if key1 > key2:
                merged_result.append(data[i])
                i += 1
            else:
                merged_result.append(data[j])
                j += 1

    while i <= mid_index:
        merged_result.append(data[i])
        i += 1

    while j <= right_index:
        merged_result.append(data[j])
        j += 1

    for k in range(len(merged_result)):
        data[left_index + k] = merged_result[k]

    return data
```

Code

```

def merge_sort(data, column_index, ascending=True, start=None, end=None):
    if start is None:
        start = 0
    if end is None:
        end = len(data) - 1

    if start < end:
        mid = (start + end) // 2
        merge_sort(data, column_index, ascending, start, mid)
        merge_sort(data, column_index, ascending, mid + 1, end)
        merge(start, mid, end, data, column_index, ascending)
    return data
```

Hybrid Merge

The Hybrid Merge Sort algorithm is a sorting algorithm that is based on the Divide and Conquer paradigm. Hybrid Merge Sort optimizes the sorting process by using Merge Sort for large subarrays and switching to Insertion Sort for small subarrays, resulting in a more efficient sorting algorithm overall.

Code

```
def hybrid_merge_sort(data, column_index, ascending=True, start=0, end=None, n=10):
    if end is None:
        end = len(data) - 1

    if start < end:
        if end - start <= n:
            insertion_sort(start, end, data, column_index, ascending)
        else:
            mid = (start + end) // 2
            hybrid_merge_sort(data, column_index, ascending, start, mid, n)
            hybrid_merge_sort(data, column_index, ascending, mid + 1, end, n)
            merge(start, mid, end, data, column_index, ascending)

    return data
```

Heap Sort	Heap Sort is a comparison-based sorting technique based on the Binary Heap data structure. It is similar to the selection sort where we first find the minimum element and place the minimum element at the beginning. Repeat the same process for the remaining elements.

Code

```python


def heapify(arr, start, end, i, column_index, ascending=True):
    largest = i
    left = 2 * i + 1
    right = 2 * i + 2

    if left <= end and (
    (clean_value(arr[left][column_index]) > clean_value(arr[largest][column_index])) if ascending else (
            clean_value(arr[left][column_index]) < clean_value(arr[largest][column_index]))):
        largest = left

    if right <= end and (
    (clean_value(arr[right][column_index]) > clean_value(arr[largest][column_index])) if ascending else (
            clean_value(arr[right][column_index]) < clean_value(arr[largest][column_index]))):
        largest = right

    if largest != i:
        arr[i], arr[largest] = arr[largest], arr[i]
        heapify(arr, start, end, largest, column_index, ascending)
```

Code

```python

def heapSort(arr, start, end, column_index, ascending=True):
    try:
        if start < 0 or end >= len(arr):
            raise ValueError("Invalid start or end index")

        n = end - start + 1
        for i in range(n // 2 - 1, -1, -1):
            heapify(arr, start, end, i, column_index, ascending)

        for i in range(n - 1, 0, -1):
            arr[start + i], arr[start] = arr[start], arr[start + i]  # Swap
            heapify(arr, start, start + i - 1, 0, column_index, ascending)  # Adjusted the end index
        return arr
    except Exception as e:
        print("Error occurred during sorting:", str(e))
        return arr
```

<table>
<tr><td>Insertion Sort</td><td>Insertion sort is a simple sorting algorithm that works similarly to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed in the correct position in the sorted part.</td></tr>
<tr><td>Code</td><td><pre><code>def insertion_sort(start, end, data, column_index, ascending=True):
    for i in range(start + 1, end + 1):
        key = data[i]
        j = i - 1
        while j >= start and (clean_value(data[j][column_index]) > clean_value(key[column_index]) if ascending else clean_value(data[j][column_index]) < clean_value(key[column_index])):
            data[j + 1] = data[j]
            j -= 1
        data[j + 1] = key

def clean_value(value):
    try:
        return int(value.replace(",", "").replace("$", "").replace("In␣Stock", "").strip())
    except ValueError:
        try:
            return float(value.replace(",", "").replace("$", "").replace("In␣Stock", "").strip())
        except ValueError:
            return value.strip()
</code></pre></td></tr>
</table>

Bubble Sort

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order. This algorithm is not suitable for large data sets as its average and worst-case time complexity is quite high.

Code

```python
def bubble_sort(data, column_index, ascending=True):
    for i in range(len(data)):
        for j in range(0, len(data) - i - 1):
            key1 = clean_value(data[j][column_index])
            key2 = clean_value(data[j + 1][column_index])
            if ascending:
                if key1 > key2:
                    data[j], data[j + 1] = data[j + 1], data[j]
            else:
                if key1 < key2:
                    data[j], data[j + 1] = data[j + 1], data[j]
    return data
def clean_value(value):
    try:
        return int(value.replace(",", "").replace("$", "").replace("In Stock", "").strip())
    except ValueError:
        try:
            return float(value.replace(",", "").replace("$", "").replace("In Stock", "").strip())
        except ValueError:
            return value.strip()
```

Selection Sort

Selection sort is a sorting algorithm that selects the smallest element from an unsorted list in each iteration and places that element at the beginning of the unsorted list.

Code

```python

def selection_sort(data, column_index, ascending=True):
    for i in range(len(data)):
        least_value_index = i
        for j in range(i + 1, len(data)):
            key1 = clean_value(data[j][column_index])
            key2 = clean_value(data[least_value_index][column_index])
            if ascending:
                if key1 < key2:
                    least_value_index = j
            else:
                if key1 > key2:
                    least_value_index = j
        data[i], data[least_value_index] = data[least_value_index], data[i]
    return data
def clean_value(value):
    try:
        return int(value.replace(",", "").replace("$", "").replace("In Stock", "").strip())
    except ValueError:
        try:
            return float(value.replace(",", "").replace("$", "").replace("In Stock", "").strip())
        except ValueError:
            return value.strip()
```

Quick Sort	A sorting technique that sequences a list by continuously dividing the list into two parts and moving the lower items to one side and the higher items to the other. It starts by picking one item in the entire list to serve as a pivot point. The pivot could be the first item or a randomly chosen one.

Code

```python
import random
def QuickSort(arr, start, end, column_index, ascending=True):
    if start < end:
        part = partitionRandom(arr, start, end, column_index, ascending)
        QuickSort(arr, start, part - 1, column_index, ascending)
        QuickSort(arr, part + 1, end, column_index, ascending)
    return arr
def partitionRandom(arr, start, end, column_index, ascending=True):
    rand = random.randint(start, end)
    arr[end], arr[rand] = arr[rand], arr[end]
    return partition(arr, start, end, column_index, ascending)

def partition(arr, start, end, column_index, ascending=True):
    pivot = clean_value(arr[end][column_index])
    i = start - 1
    for j in range(start, end):
        key = clean_value(arr[j][column_index])
        if (key <= pivot) if ascending else (key >= pivot):
            i += 1
            arr[i], arr[j] = arr[j], arr[i]
    arr[i + 1], arr[end] = arr[end], arr[i + 1]
    return i + 1

def clean_value(value):
    try:
        return int(value.replace(",", "").replace("$", "").replace("In Stock", "").strip())
    except ValueError:
        try:
            return float(value.replace(",", "").replace("$", "").replace("In Stock", "").strip())
        except ValueError:
            return value.strip()
```

Shell Sort	Shell sort is a generalized version of the insertion sort algorithm. It first sorts elements that are far apart from each other and successively reduces the interval between the elements to be sorted. The interval between the elements is reduced based on the sequence used.
Code	

```python
def shellSort(arr, start, end, column_index, ascending=True):
    try:
        gap = (end - start) // 2
        while gap > 0:
            j = gap + start
            while j <= end:
                i = j - gap
                key = arr[j][column_index]
                key_value = clean_value(key)
                while i >= start and ((clean_value(arr[i][column_index]) > key_value) if ascending else (clean_value(arr[i][column_index]) < key_value)):
                    arr[i + gap][column_index] = arr[i][column_index]
                    i -= gap
                arr[i + gap][column_index] = key
                j += 1
            gap //= 2
        return arr
    except Exception as e:
        print("Error␣occurred␣during␣sorting:", str(e))
        return arr
```

11 Non-Comparison Based Algorithm

Non-comparison-based sorting algorithms are sorting algorithms that do not rely on comparing elements directly to sort a collection of items. Unlike comparison-based algorithms (such as bubble sort, insertion sort, merge sort, etc.)

11.1 Algorithums

In total, we used 12 algorithms. Out of them, 4 non-comparison-based or Linear sorting algorithms that we used for sorting are;

- Bucket Sort
- Radix Sort
- Couting Sort
- Pigeon Sort

Their description, How we used them, and their pseudo-code is given bellow.

<table>
<tr><th>Algorithm Name</th><th>Description</th></tr>
<tr><td>Bucket Sort</td><td>Bucket Sort is a sorting algorithm that divides the unsorted array elements into several groups called buckets. Each bucket is then sorted by using any of the suitable sorting algorithms or recursively applying the same bucket algorithm. Finally, the sorted buckets are combined to form a final sorted array.</td></tr>
<tr><td>Code</td><td>

```

def bucket_sort(data, column_index, ascending=True):
    max_value = max(clean_value(row[column_index]) for row in data)
    min_value = min(clean_value(row[column_index]) for row in data)
    bucket_range = (max_value - min_value) / len(data)
    num_buckets = len(data)
    buckets = [[] for _ in range(num_buckets)]

    # Distribute the elements into buckets
    for row in data:
        value = clean_value(row[column_index])
        bucket_index = int((value - min_value) / bucket_range)
        if bucket_index == num_buckets:
            bucket_index -= 1
        buckets[bucket_index].append(row)

    # Sort individual buckets using insertion sort
    for i in range(num_buckets):
        buckets[i] = insertion_sort(0, len(buckets[i]), buckets[i],
            column_index, ascending)

    # Concatenate the sorted buckets
    sorted_data = []
    for bucket in reversed(buckets) if not ascending else buckets:
        sorted_data.extend(bucket)

    return sorted_data
```

</td></tr>
</table>

Counting Sort	Counting sort is a sorting algorithm that sorts the elements of an array by counting the number of occurrences of each unique element in the array.

Code

```python

def counting_sort(data, column_index, ascending=True):
    if not data:
        return data

    max_value = max(int(row[column_index]) for row in data)
    min_value = min(int(row[column_index]) for row in data)
    range_size = max_value - min_value + 1

    count = [0] * range_size
    output = [None] * len(data)

    for row in data:
        value = int(row[column_index]) - min_value
        count[value] += 1

    for i in range(1, len(count)):
        count[i] += count[i - 1]

    for row in reversed(data):
        value = int(row[column_index]) - min_value
        output[count[value] - 1] = row
        count[value] -= 1

    return output if ascending else output[::-1]
```

Radix Sort	Radix sort is a sorting algorithm that sorts the elements by first grouping the individual digits of the same place value. Then, sort the elements according to their increasing/decreasing order.

Code

```python
def get_digit(num, digit_index):
    # Extract the digit at the given index from the number
    return num // 10**digit_index % 10

def counting_sort_radix(data, column_index, digit_index, ascending=True):
    count = [0] * 10
    output = [None] * len(data)

    for row in data:
        num = int(row[column_index])
        digit = get_digit(num, digit_index)
        count[digit] += 1

    if ascending:
        for i in range(1, 10):
            count[i] += count[i - 1]
    else:
        for i in range(8, -1, -1):
            count[i] += count[i + 1]

    i = len(data) - 1
    while i >= 0:
        num = int(data[i][column_index])
        digit = get_digit(num, digit_index)
        output[count[digit] - 1] = data[i]
        count[digit] -= 1
        i -= 1

    return output


def radix_sort(data, column_index, ascending=True):
    # Find the maximum number to determine the number of digits
    max_num = max(int(row[column_index]) for row in data)
    digit_index = 0
    while max_num // 10**digit_index > 0:
        data = counting_sort_radix(data, column_index, digit_index, ascending)
        digit_index += 1
    return data
```

Pegion Hole Sort

Pigeonhole Sort is a simple sorting algorithm that works well for sorting a small range of integers by distributing elements into ”pigeonholes” (buckets) based on their values and then gathering them back in sorted order.

Code

```
def pigeonhole_sort(data, column_index, ascending=True):
    min_value = float("inf")
    max_value = float("-inf")
    for row in data:
        value = clean_value(row[column_index])
        if isinstance(value, (int, float)):
            min_value = min(min_value, value)
            max_value = max(max_value, value)

    range_size = int(max_value - min_value) + 1

    pigeonholes = [[] for _ in range(range_size)]
    for row in data:
        value = clean_value(row[column_index])
        pigeonhole_index = int(value - min_value)
        pigeonholes[pigeonhole_index].append(row)
    sorted_data = []
    for pigeonhole in pigeonholes:
        sorted_data.extend(pigeonhole)

    return sorted_data if ascending else sorted_data[::-1]

import random
import math

def clean_value(value):
    try:
        return int(value.replace(",", "").replace("$", "").replace("In␣Stock", "").strip())
    except ValueError:
        try:
            return float(value.replace(",", "").replace("$", "").replace("In␣Stock", "").strip())
        except ValueError:
            return value.strip()
```

<table>
<tr><th>Algorithm Name</th><th>Description</th></tr>
<tr><td>Hash Search</td><td>The Hash Search Algorithm is a method used to efficiently search for specific items in a dataset. It works by creating a hash table, which is a data structure that enables quick data retrieval. In this algorithm, each data item is hashed (converted into a unique numerical value) and stored in the hash table..</td></tr>
<tr><td>Code</td><td>

```
def hash_search(data, search_text, attribute_index):
    search_text = search_text.lower()
    search_results = []

    hash_table = {}
    for row in data[1:]:
        # Convert the attribute value to lowercase for case-insensitive search
        value = row[attribute_index].lower()

        if search_text in value:
            search_results.append(row)
    return search_results
```

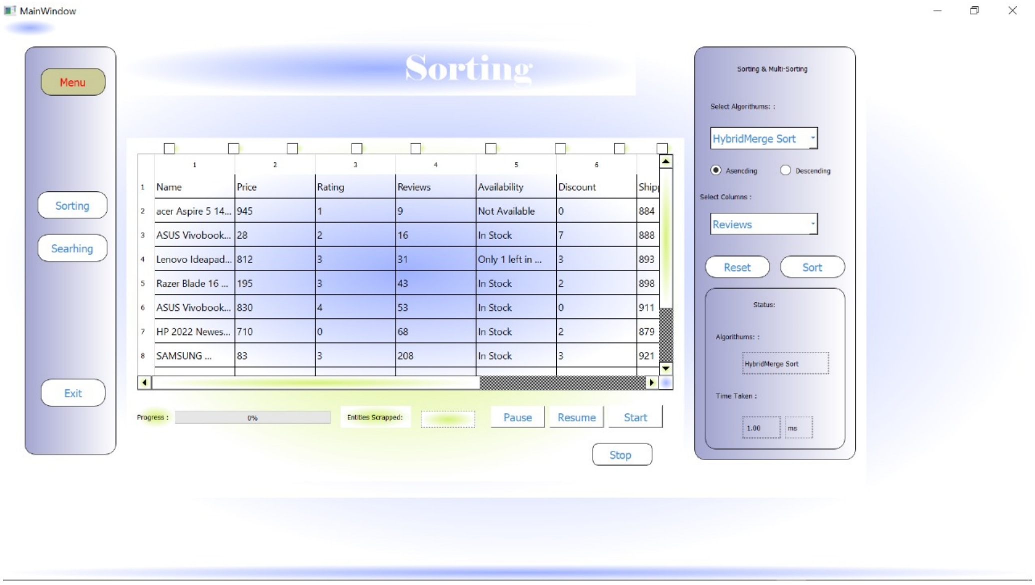
</td></tr>
<tr><td>Linear Search</td><td>The Linear Search Algorithm, also known as a sequential search, is a simple and straightforward method used to find a specific item in a collection of data. It works by examining each item in the dataset one by one, starting from the beginning, until it finds a match with the target search term.</td></tr>
<tr><td>Code</td><td>

```
def linear_search(data, search_text, attribute_index):
    search_text = search_text.lower()
    results = []

    for row in data:
        col_text = row[attribute_index].lower()
        if search_text in col_text:
            results.append(row)

    return results
```

</td></tr>
</table>

12 Snap Shots

12.1 Sorting in Ascending



	1	2	3	4	5	6	
1	Name	Price	Rating	Reviews	Availability	Discount	Ship
2	acer Aspire 5 14...	945	1	9	Not Available	0	884
3	ASUS Vivobook...	28	2	16	In Stock	7	888
4	Lenovo Ideapad...	812	3	31	Only 1 left in ...	3	893
5	Razer Blade 16 ...	195	3	43	In Stock	2	898
6	ASUS Vivobook...	830	4	53	In Stock	0	911
7	HP 2022 Newes...	710	0	68	In Stock	2	879
8	SAMSUNG ...	83	3	208	In Stock	3	921

Progress : 0%

Entities Scrapped:

Sorting & Multi-Sorting

Select Algorithums: : HybridMerge Sort

● Asencding ○ Descending

Select Columns : Reviews

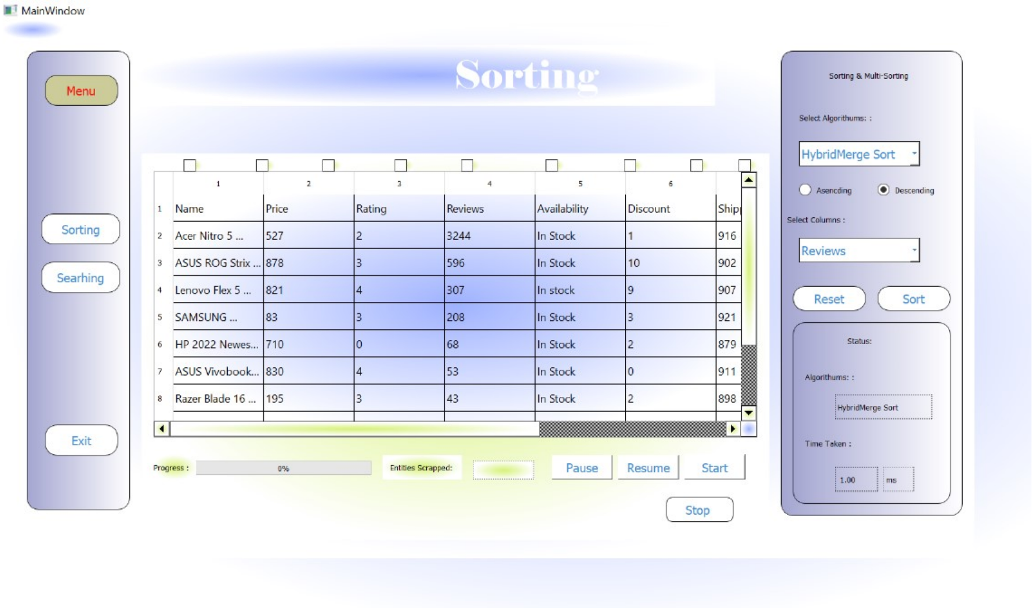
Status:

Algorithums: : HybridMerge Sort

Time Taken : 1.00 ms

Figure 5: Sorting in Ascending

12.2 Sorting in Descending



	1	2	3	4	5	6	
1	Name	Price	Rating	Reviews	Availability	Discount	Ship
2	Acer Nitro 5 ...	527	2	3244	In Stock	1	916
3	ASUS ROG Strix ...	878	3	596	In Stock	10	902
4	Lenovo Flex 5 ...	821	4	307	In stock	9	907
5	SAMSUNG ...	83	3	208	In Stock	3	921
6	HP 2022 Newes...	710	0	68	In Stock	2	879
7	ASUS Vivobook...	830	4	53	In Stock	0	911
8	Razer Blade 16 ...	195	3	43	In Stock	2	898

Progress : 0%

Entities Scrapped:

Sorting & Multi-Sorting

Select Algorithums: : HybridMerge Sort

○ Asencding ● Descending

Select Columns : Reviews

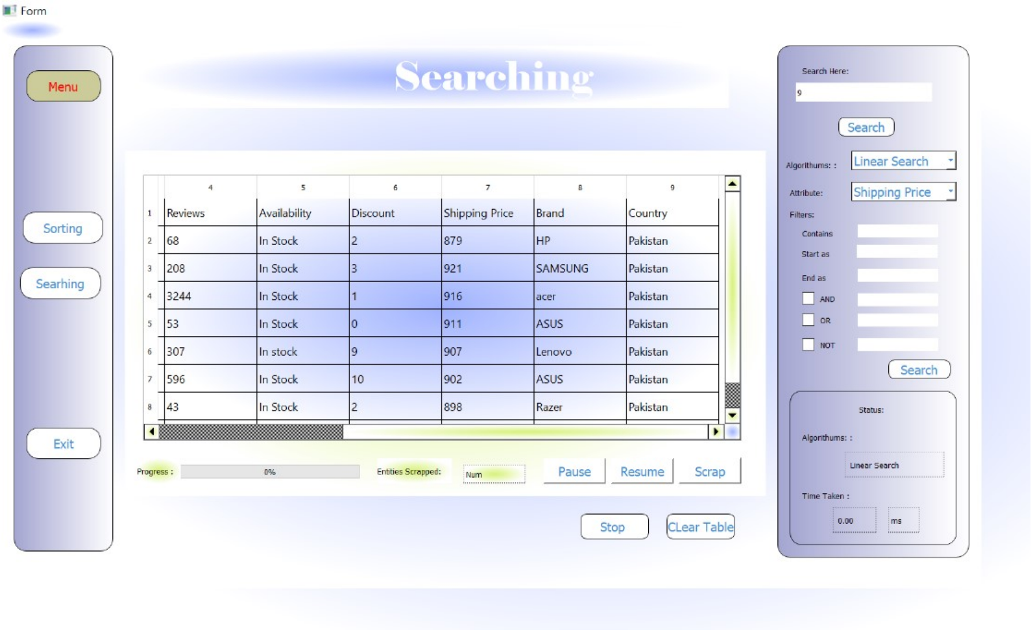
Status:

Algorithums: : HybridMerge Sort

Time Taken : 1.00 ms

Figure 6: Sorting in Descending

12.3 Searching



	4	5	6	7	8	9
1	Reviews	Availability	Discount	Shipping Price	Brand	Country
2	68	In Stock	2	879	HP	Pakistan
3	208	In Stock	3	921	SAMSUNG	Pakistan
4	3244	In Stock	1	916	acer	Pakistan
5	53	In Stock	0	911	ASUS	Pakistan
6	307	In stock	9	907	Lenovo	Pakistan
7	596	In Stock	10	902	ASUS	Pakistan
8	43	In Stock	2	898	Razer	Pakistan

Progress : 0%

Entities Scrapped: Num

Search Here: 9

Algorithums: : Linear Search

Attribute: Shipping Price

Filters:

Contains

Start as

End as

□ AND

□ OR

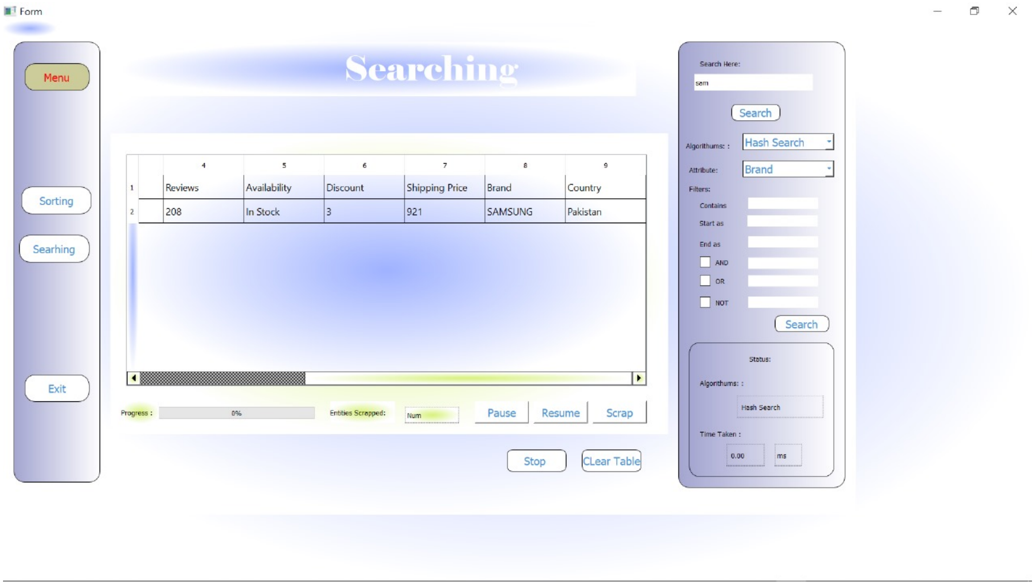
□ NOT

Algorithums: : Linear Search

Time Taken : 0.00 ms

Figure 7: Search

12.4 Using Hash search



	4	5	6	7	8	9
1	Reviews	Availability	Discount	Shipping Price	Brand	Country
2	208	In Stock	3	921	SAMSUNG	Pakistan

Progress : 0%

Entities Scrapped: Num

Search Here: sam

Algorithums: : Hash Search

Attribute: Brand

Filters:

Contains

Start as

End as

□ AND

□ OR

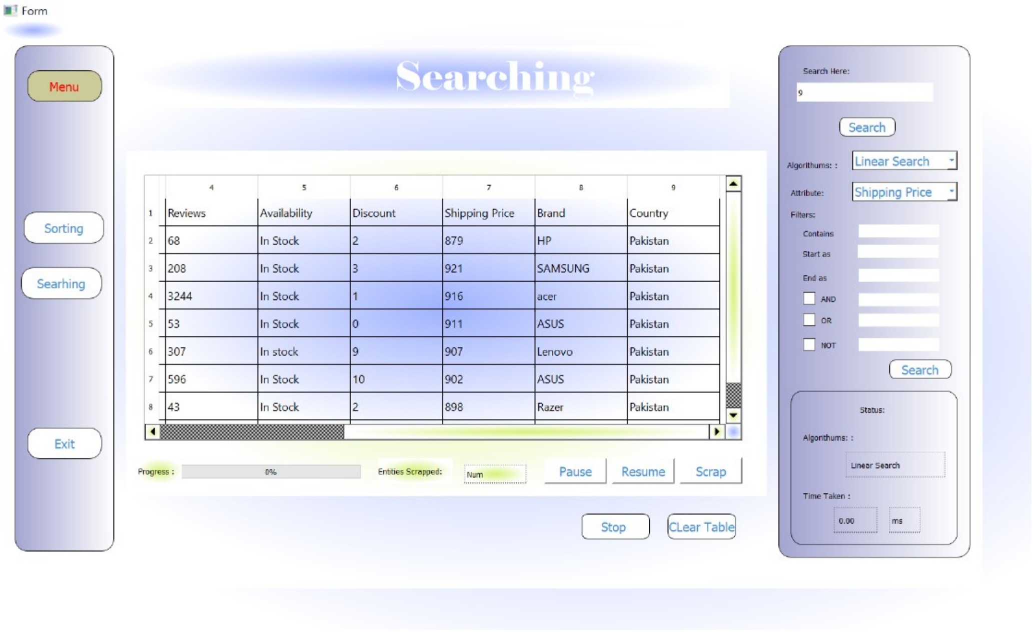
□ NOT

Algorithums: : Hash Search

Time Taken : 0.00 ms

Figure 8: Hash Search

12.5 Using Linear search



Form

Searching

	4	5	6	7	8	9
1	Reviews	Availability	Discount	Shipping Price	Brand	Country
2	68	In Stock	2	879	HP	Pakistan
3	208	In Stock	3	921	SAMSUNG	Pakistan
4	3244	In Stock	1	916	acer	Pakistan
5	53	In Stock	0	911	ASUS	Pakistan
6	307	In stock	9	907	Lenovo	Pakistan
7	596	In Stock	10	902	ASUS	Pakistan
8	43	In Stock	2	898	Razer	Pakistan

Progress : 0%

Entities Scrapped: Num

Search Here: 9

Algorithums: : Linear Search

Attribute: Shipping Price

Filters:

Contains

Start as

End as

□ AND

□ OR

□ NOT

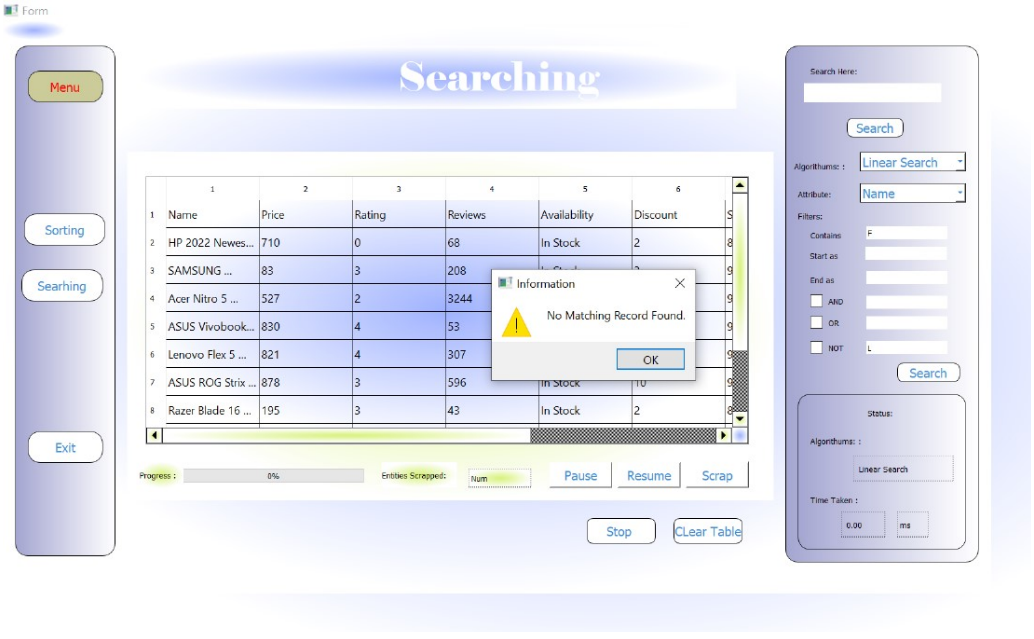
Status:

Algorithums: : Linear Search

Time Taken : 0.00 ms

Figure 9: Linear Search

12.6 Using NOT



Form

Searching

	1	2	3	4	5	6	
1	Name	Price	Rating	Reviews	Availability	Discount	S
2	HP 2022 Newes...	710	0	68	In Stock	2	8
3	SAMSUNG ...	83	3	208	In Stock	3	9
4	Acer Nitro 5 ...	527	2	3244			9
5	ASUS Vivobook...	830	4	53			9
6	Lenovo Flex 5 ...	821	4	307			9
7	ASUS ROG Strix ...	878	3	596	In Stock	10	9
8	Razer Blade 16 ...	195	3	43	In Stock	2	8

Information

No Matching Record Found.

Progress : 0%

Entities Scrapped: Num

Search Here:

Algorithums: : Linear Search

Attribute: Name

Filters:

Contains F

Start as

End as

□ AND

□ OR

□ NOT L

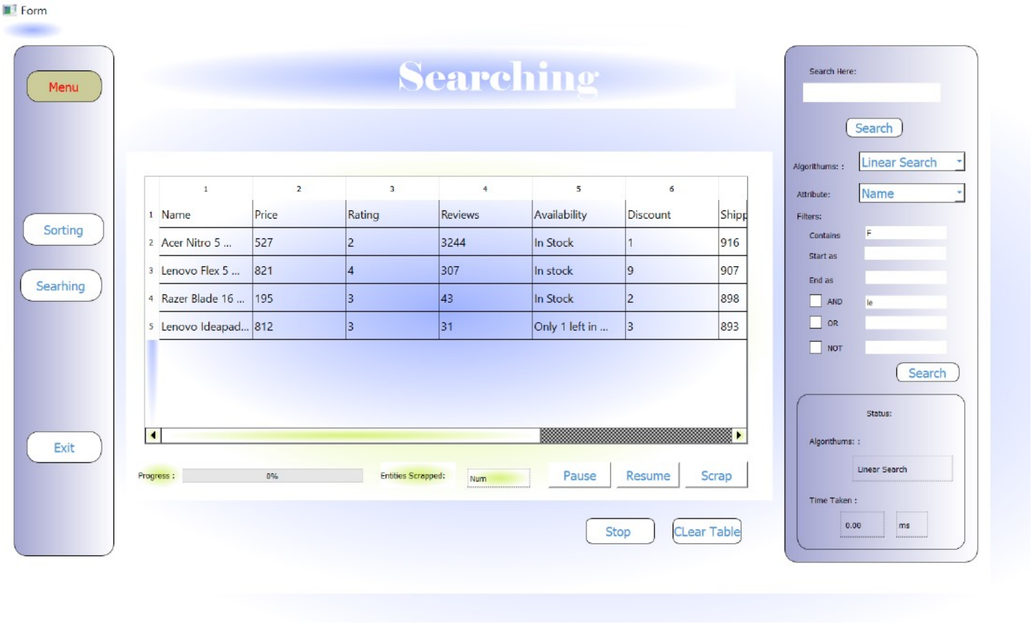
Status:

Algorithums: : Linear Search

Time Taken : 0.00 ms

Figure 10: Not Search

12.7 Using AND



	1	2	3	4	5	6	
1	Name	Price	Rating	Reviews	Availability	Discount	Shipp
2	Acer Nitro 5 ...	527	2	3244	In Stock	1	916
3	Lenovo Flex 5 ...	821	4	307	In stock	9	907
4	Razer Blade 16 ...	195	3	43	In Stock	2	898
5	Lenovo Ideapad...	812	3	31	Only 1 left in ...	3	893

Progress : 0%

Entities Screpped: Num

Search Here:

Algorithums: : Linear Search

Attribute: Name

Filters:

Contains F

Start as

End as

□ AND le

□ OR

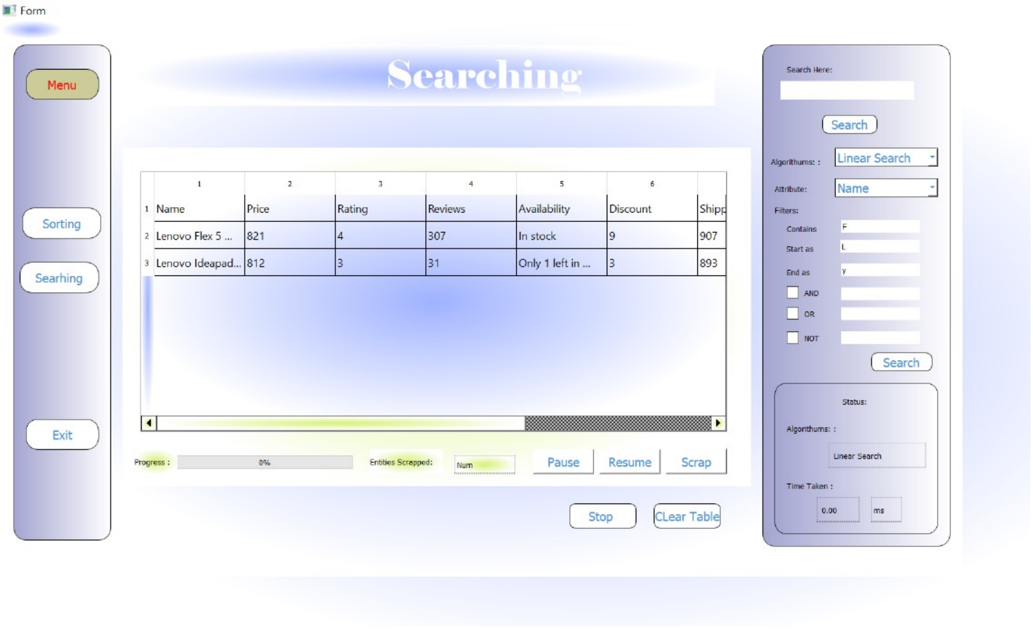
□ NOT

Algorithums: : Linear Search

Time Taken : 0.00 ms

Figure 11: AND Search

12.8 Using Contains,start and end



	1	2	3	4	5	6	
1	Name	Price	Rating	Reviews	Availability	Discount	Shipp
2	Lenovo Flex 5 ...	821	4	307	In stock	9	907
3	Lenovo Ideapad...	812	3	31	Only 1 left in ...	3	893

Progress : 0%

Entities Screpped: Num

Search Here:

Algorithums: : Linear Search

Attribute: Name

Filters:

Contains F

Start as L

End as y

□ AND

□ OR

□ NOT

Algorithums: : Linear Search

Time Taken : 0.00 ms

Figure 12: Contains,start and end

13 Conclusion

In conclusion, ”Gadget Glance” aims to provide users with a user-friendly tool to amaze their online shopping experience, helping them make well-informed decisions when purchasing digital products. By scraping data from online stores and offering various sorting and searching options, the project empowers users to find the products that best suit their needs. It also offers a wide range of sorting and searching algorithms in order to deal with various user preferences.