**Design Structure and Algorithm**

**Project Title: Cine- Scrap**



**Project Supervisor**

Mr. Samyan Qayyum Wahla

**Group ID : 62**

**Project Members**

|  |  |
| --- | --- |
| Names | Registration no. |
| Esha Tanveer | 2020-CS-58 |
| Khadija Iqbal | 2020-CS-89 |

Department of Computer Science

University of Engineering and Technology, Lahore – Pakistan

**Table of Contents**

[**Learning Outcome:** 3](#_Toc86992193)

[**Proposal Details** 3](#_Toc86992194)

[**Proposed Project Title:** 3](#_Toc86992195)

[**Executive Summary:** 3](#_Toc86992196)

[**Planning:** 4](#_Toc86992197)

[**Motivation:** 4](#_Toc86992198)

[**Task Division:** 4](#_Toc86992199)

[**Collaboration in Integration:** 4](#_Toc86992200)

[**Data Scrapping:** 4](#_Toc86992201)

[**Description:** 4](#_Toc86992202)

[**Description of the scrapped attributes:** 4](#_Toc86992203)

[URL 5](#_Toc86992204)

[GitHub Link 5](#_Toc86992205)

[**GUI:** 6](#_Toc86992206)

[**Pencil Tool:** 6](#_Toc86992207)

[**Description of Components:** 6](#_Toc86992208)

[**Actual Gui:** 7](#_Toc86992209)

[**Description of Components:** 7](#_Toc86992210)

[**Description of GUI:** 8](#_Toc86992211)

[**Loading Data:** 8](#_Toc86992212)

[**Start Scrapping:** 8](#_Toc86992213)

[**Stop and pause Scrapping:** 8](#_Toc86992214)

[**Progress of scrapping:** 8](#_Toc86992215)

[**Algorithms:** 8](#_Toc86992216)

[**Sorting Algoritms:** 8](#_Toc86992217)

[**Description:** 8](#_Toc86992218)

[**Pseudo Codes** 9](#_Toc86992219)

[**Time Complexities:** 10](#_Toc86992220)

[**Python Codes:** 12](#_Toc86992221)

[**Space Complexities:** 14](#_Toc86992222)

[**Searching Algorithms:** 14](#_Toc86992223)

[**Description:** 14](#_Toc86992224)

[**Time Complexities:** 14](#_Toc86992225)

[**Space Complexities:** 15](#_Toc86992226)

# **Learning Outcome:**

The main idea of this Project is to sort a large amount of data through different types of sorting algorithms in an efficient way. We should be able to apply the searching and sorting algorithms on a very huge amount of Data.

**Proposal Details**

## **Proposed Project Title:**

Cine-Scrap

**Executive Summary:**

Web scrapping is mostly used by Business companies and industries usually with or without any technical knowledge. This project is entirely based upon the scrapping of movies. It works in a way that it extracts the relevant data provided by the user, scrapes it, and will move it on the list shown on GUI. The purpose of web scrappers is to collect as much data as it can. Same is the case with this project, It will scrap data of almost 1 million movies / TV shows. This project will not only show the data of one genre but it will show the data of multiple genres at a time i.e.; Comedy, Sci- Fi, Horror, Comedy-Romance, Fantasy etc.

Every movie has separate attributes through which 7 of them will be scrapped. The attributes for scrapping are:

* Title
* Ratings
* Duration
* Year
* Genre
* Votes
* Certificates

Name of the movie/show

It is GUI based project. The tools used for this purpose will be:

* **Pandas** (in case to assemble the data in Data Frame)
* **Beautiful Soap** (that enables us to parse the HTML files)
* **PyQt5** (used to make the GUI)
* **Warnings** (to deal with the unnecessary warnings)
* **Request** (to open the URL of the Website from which the data is scrapped)

This gui based project consists of viewing a list of thousands and millions of shows with sorting and searching techniques. Also, it will pause scrapping if the user presses the pause button. Film Marketing Companies uses the movie scrappers to analyze the main factors, measured by film ratings that affect the success of films.

Some social media scrappers uses it to differentiate between best movies.

# **Planning:**

## **Motivation:**

If we talk about today's generation, it is very obvious that every second person is fond of watching seasons and movies. What if you are given the description of every season without rolling into several pages and finding the perfect one. This idea might be quite helpful to select the best among all, to select your favorite genre and then watch the movie. This can only be done through scrapping. This idea of ease and the fond of people towards shows urged us to select the scrapping of movies and shows from the best website of TV-shows.

## **Task Division:**

The planning of the project was started the next day of the acceptance of proposal of project. Though the task division at the start was not much clear because of no idea of complexities in the project. It is very obvious that group task needs a good collaboration of both team members. Hence after the submission of first report, we started to divide the tasks of milestone given. Most of the work was done in university with the ideas of two minds. Scrapping was also not only done by one person. The code was changed several times and both of us worked together.

## **Collaboration in Integration:**

Data loaded from csv to table by 2020-CS-89

Algorithms written by 2020-CS-58

Searching Algorithms written by 2020-CS-58

The merge of files by 2020-CS-89

**Data Scrapping:**

## **Description:**

The data of this project has been scrapped from the very famous website of Movies and TV-shows and seasons; IMDb. IMDb with a full form of Internet Movie Database, is an online database of information about films, television shows, home videos, video games, and online streaming content, including cast, production crew, and personal biographies, plot summaries, votes, ratings, and fan and critical reviews. IMDb's database contains approximately 8 million titles.

The purpose of scrapping through this website was its variety and the amount of data it contains. This project had scrapped of more than 1 million data and the common genre in all of them is **Comedy.**

## **Description of the scrapped attributes:**

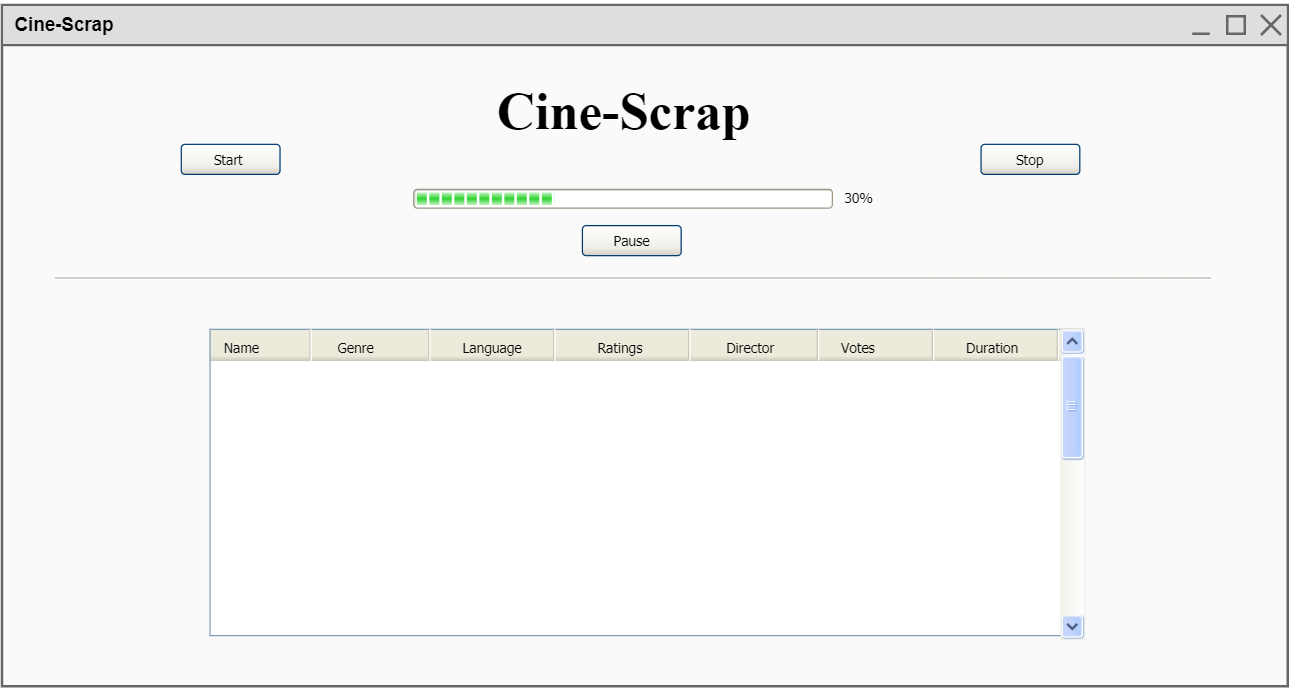
|  |  |  |
| --- | --- | --- |
| Name | Data Type | Description |
| Title | String | It will provide the name of the movie or show. |
| Year | String | It will inform the released year of the movie. |
| Duration | String | It will give info about the total time of the movie. |
| Genre | String | It will tell which type of the movie is, either it is comedy or action or any other. |
| Ratings | Float | It will give the movie ratings in numbers. |
| Votes | String | It will give the votes given to the Show. |
| Certificate | String | It will inform whether it is PG-13 or R or any other. |

URL= <https://www.imdb.com/search/title/genres=comedy&start=51&explore=title_type,genres&ref_=adv_nxt>

GitHub Link: <https://github.com/eshatanvr/CS261F21PID62>

# **GUI:**

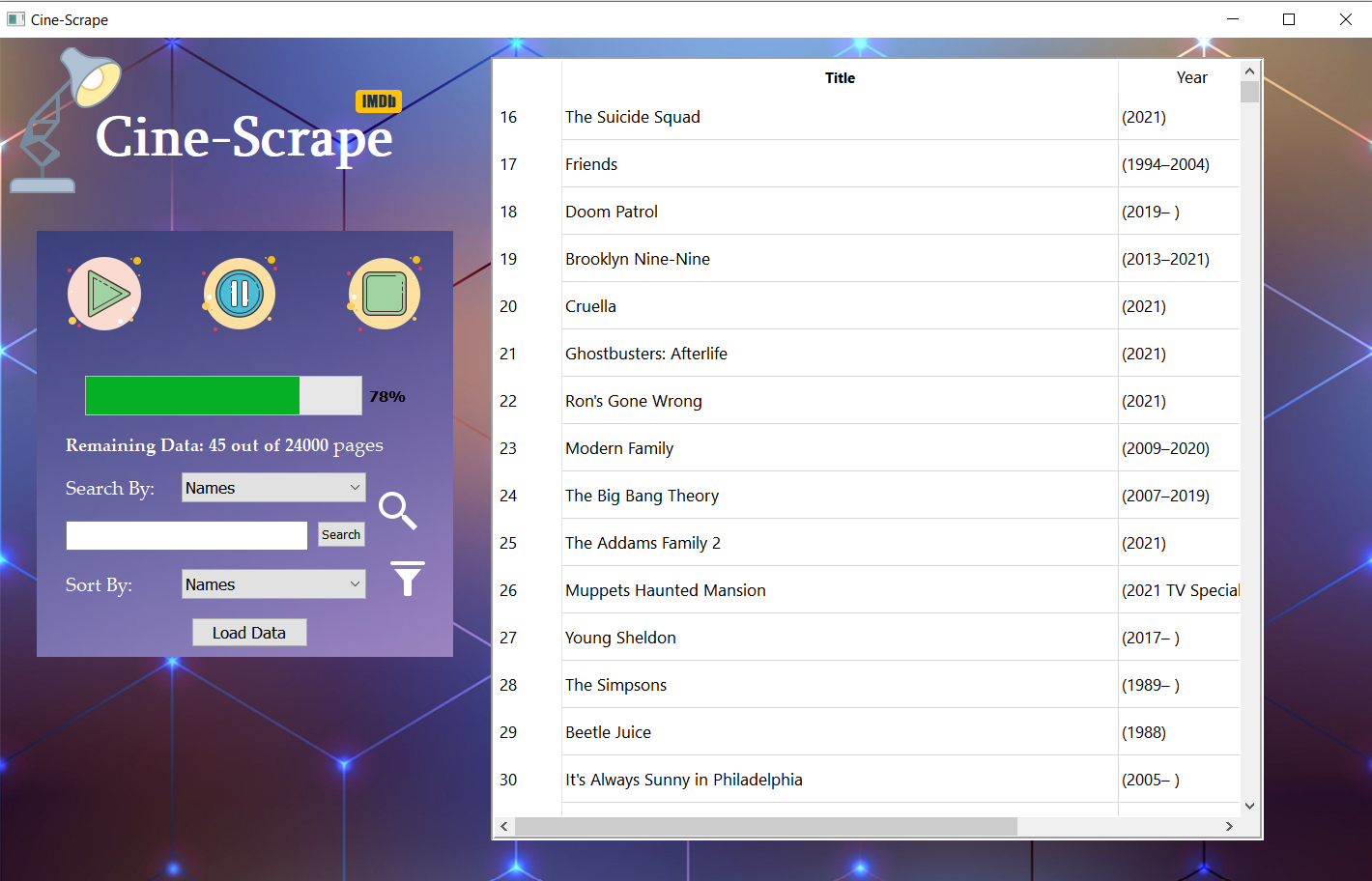
## **Pencil Tool:**



### **Description of Components:**

|  |  |  |
| --- | --- | --- |
| Name | Object Name | Purpose |
| Start Button | Push Button | To start the scrapping of website |
| Pause Button | Push Button | To pause the scrapping |
| Stop Button | Push Button | To stop the Scrapping |
| Progress Bar | ProgressBar | To check the progress of scrapping |
| Scroll bar | Scrollbar | To scroll the table |
| Table | Table | To display the data of scrapped items |

## **Actual Gui:**



### **Description of Components:**

|  |  |  |
| --- | --- | --- |
| Name | Object Name | Purpose |
| Start Button | Label\_8 | To start the scrapping of website |
| Pause Button | Label\_9 | To pause the scrapping |
| Stop Button | Label\_10 | To stop the Scrapping |
| Progress Bar | progressBar | To show the progress of scrapping |
| Searching bar | ComboBox | To search among the data |
| Sorting Menu | ComboBox\_3 | To sort the data |
| Table | tableWidget | To display the data |
| Search Button | Pushbutton | To press for searching |
| Load button | PushButton\_2 | To load the data from csv |

## **Description of GUI:**

### **Loading Data:**

When you run the project, the only window that appears is the one shown above. The already scraped data is contained in the file MoviesData.csv. If the user wants to load that data, he or she can do so by pressing the Load Data button. It will take some time because a large amount of data cannot be loaded in a matter of seconds.

### **Start Scrapping:**

When the start button is pressed, it will begin scraping from the website, and the existing file will be edited with the scrapped data.

### **Stop and pause Scrapping:**

If the user does not want to scrape all of the data, he or she can pause or stop the scraping and resume functioning on previously scraped data.

### **Progress of scrapping:**

While the scrapping is in process, the user must want to see the current progress of the scrapping. This progress bar will help there to show the progress of scrapping data.

# **Algorithms:**

## **Sorting Algoritms:**

### **Description:**

**Selection Sort:**

The selection sort algorithm sorts an array by repeatedly identifying the least unsorted element.

**Insertion Sort:**

In insertion sort algorithm Values from the unsorted part are picked and placed at the correct position.

**Merge Sort:**

Merge sort repeatedly divides a list into sub lists then merges those sub lists into a sorted list.

**Quick Sort:**

Like Merge sort, Quick sort also falls into the category of divide and conquer approach but it sets a pivot.

**Bubble Sort:**

Bubble sorting goes through the list repeatedly, compare and swaps adjacent elements.

**Heap Sort:**

Heapsort splits its input into a sorted and unsorted region, and recursively reduces the unsorted by obtaining the largest element from it.

**Bucket Sort:**

Bucket sort divides the elements of an array into a number of buckets. Each bucket is then individually sorted.

**Radix Sort:**

Radix sort sorts by using the digits of individual numbers. Sorting is done from least to most significant digits.

**Counting Sort:**

A counting sort is a sorting algorithm that sorts an array's elements by counting the number of occurrences.

**Pseudo Codes:**

**Insertion Sort:**

For i =2 to array.length do

Key = array[i]

j= i-1

while j>0 and array[j]>key do

array[i+1] = array[i]

i=i-1

array[i+1]=key

**Bucket Sort:**

For i=0 to A.length - 1 do

B[i]=0

For i=0 to A.length - 1 do

B[i]=B[A.length\*A[i]]

*Sort with Insertion sort and concatenate*

**Counting Sort:**

k = range of elements of array

count=array of k + 1 zeros

output= array of same length as input

for i = 0 to length(input) - 1 do

j = key(input[i])

count[j] += 1

for i = 1 to k do

count[i] += count[i - 1]

for i = length(input) - 1 down to 0 do

j = key(input[i])

count[j] -= 1

output[count[j]] = input[i]

**Selection Sort:**

for i = 1 to A.length - 1 do

min = i

for j = i+1 to A.length do

if A[j] < A[min] then

min = j;

if min != i then

swap A[min] and A[i]

**Bubble sort:**

for i = 0 to A.length-1 do:

flag= false

for j = 0 to A.length -1 do:

if A[j] > A[j+1] then

swap( A[j], A[j+1] )

swapped = true

If (flag==false) then

break

**MergeSort(arr[], l, r)**

If r > l

m = l+ (r-l)/2

mergeSort(arr, l, m)

mergeSort(arr, m+1, r)

merge(arr, l, m, r)

**Radix sort:**

Any stable sort.

**Quick sort:**

if low < high:  
 pi = partition(array, low, high)  
 quickSort(array, low, pi - 1)  
 quickSort(array, pi + 1, high)

### **Time Complexities:**

Selection Sort:

Best Case: Ω (n^2)

Average Case: θ(n^2)

Worst Case: O(n^2)

Insertion sort:

Best Case: Ω (n )

Average Case: θ (n^2)

Worst Case: O(n^2)

Merge sort:

Best Case: Ω (n lg n)

Average Case: θ (n lg n)

Worst Case: O(n lg n)

Quick sort:

Best Case: Ω (n lg n)

Average Case: θ (n lg n)

Worst Case: O(n^2)

Bucket sort:

Best Case: Ω (n +k)

Average Case: θ (n + k)

Worst Case: O(n^2)

Counting sort:

Best Case: Ω (n +k)

Average Case: θ (n + k)

Worst Case: O(n +k)

Bubble sort:

Best Case: Ω (n)

Average Case: θ (n)

Worst Case: O(n^2)

Radix sort:

Best Case: Ω (n k)

Average Case: θ (n k)

Worst Case: O(n k)

Heap sort:

Best Case: Ω (n lg n)

Average Case: θ (n lg n)

Worst Case: O(n lg n)

### **Python Codes:**

**Insertion Sort:**

def insertionSort(array1):  
 for i in range(1, len(array1)):  
 key = array1[i]  
 j = i - 1  
 while j >= 0 and array1[j] > key:  
 array1[j + 1] = array1[j]  
 j = j - 1  
 array1[j + 1] = key  
 return array1

**Counting Sort:**

def countingSort(array,k):  
 count=[0] \* k  
 output = [0] \* size  
 for i in range(0, len(array)):  
 count[array[i] - (min(array))] += 1  
  
 for i in range(1,k):  
 count[i] = count[i]+count[i-1]  
  
 j=size-1  
 while >=0:  
 output[count[array[j]-(min(array))]-1]= array[j]  
 count[array[j]-(min(array))]-=1  
 j =j-1  
 return output

**Bucket Sort:**

def bucketSort(array):  
 count = []  
 n = 10  
 for i in range(0,n):  
 count.append([])  
 for j in array:  
 index = int(n \* j)  
 count[index].append(j)  
 for i in range(0,n):  
 count[i] = insertionSort(count[i])  
 k = 0  
 for i in range(n):  
 for j in range(len(count[i])):  
 array[k] = count[i][j]  
 k += 1  
 return array

**Quick Sort:**

def partition(array, low, high):  
 pivot = array[high]  
 i = low - 1  
 for j in range(low, high):  
 if array[j] <= pivot:  
 i = i + 1  
 (array[i], array[j]) = (array[j], array[i])  
 (array[i + 1], array[high]) = (array[high], array[i + 1])  
 return i + 1  
  
def quickSort(array, low, high):  
 if low < high:  
 pi = partition(array, low, high)  
 quickSort(array, low, pi - 1)  
 quickSort(array, pi + 1, high)

**Selection Sort:**

for i in range (1, A.length()):

min = i

for j in range (0, A.length()):

if A[j] < A[min]:

min = j;

if min != i then

A[min],A[i]= A[i], A[min]

### **Space Complexities:**

|  |  |
| --- | --- |
| **Algorithms** | **Space Complexity** |
| Selection Sort | O(1) |
| Insertion Sort | O(1) |
| Merge Sort | O(n) |
| Quick sort | O (n lg n) |
| Bucket Sort | O (n) |
| Counting Sort | O(k) |
| Bubble Sort | O(1) |
| Radix Sort | O(n + k) |
| Heap Sort | O(1) |

## **Searching Algorithms:**

### **Description:**

**Linear Search**:

A linear search, also known as a sequential search, is a method for finding an element within a list. It checks each element of the list one by one until a match is discovered or the entire list is searched.

**Binary Search:**

A search technique that discovers the position of a target value within a sorted array is binary search, also known as half-interval search. The target value is compared to the array's middle element in a binary search.

### **Time Complexities:**

**Binary Search:**

Worst-case performance: O(log n)

Best-case performance: O(1)

Average performance: O(log n)

**Linear Search:**

Worst-case performance: O(n)

Best-case performance: O(1)

Average performance: O(n/2)

### **Space Complexities:**

Binary Search: O(1)

Linear Search: O(1)