Scrap the Movies



Project Supervisor

Mr. Samyan Qayyum Wahla

Group Number (**G19**)

|  |  |
| --- | --- |
| Sharjeel Iftikhar | 2020-CS-139 |
| Saqlain Nawaz | 2020-CS-135 |

Department of Computer Science

University of Engineering and Technology, Lahore Pakistan

Table of Contents

[1. Project Detail 2](#_Toc88185512)

[1.1 - Executive Summary 2](#_Toc88185513)

[1.2 - Business Needs 2](#_Toc88185514)

[1.3 - Domains in Real Life 3](#_Toc88185515)

[1.4 - Motivation for Project 3](#_Toc88185516)

[1.5 - Audience 3](#_Toc88185517)

[1.6 - Technical Details 3](#_Toc88185518)

[1.6.1 – Entities 3](#_Toc88185519)

[1.6.2 - Sample of Scrapping Source 4](#_Toc88185520)

[1.6.3 – GitHub Repository Link 5](#_Toc88185521)

[1.7 - Project Features 5](#_Toc88185522)

[1.7.1 - Scrapping Section 5](#_Toc88185523)

[1.7.2 – Sorting Section 6](#_Toc88185524)

[1.7.3 – Searching Section 6](#_Toc88185525)

[1.8 - Project Plan 6](#_Toc88185526)

[2. - Algorithms 7](#_Toc88185529)

[2.1 - Sorting Algorithms 7](#_Toc88185530)

[2.1.1 - Insertion Sort 7](#_Toc88185531)

[2.1.2 - Merge Sort: 8](#_Toc88185532)

[2.1.3 - Selection Sort: 11](#_Toc88185533)

[2.1.4 - Bubble Sort: 12](#_Toc88185534)

[2.1.5 - Quick Sort: 14](#_Toc88185535)

[2.1.6 - Hybrid Sort: 16](#_Toc88185536)

[2.1.8 - Tree Sort: 18](#_Toc88185537)

[2.2 – Searching Algorithms 20](#_Toc88185538)

[2.2.1 Direct Linear Search 20](#_Toc88185539)

[2.2.2 Start with Linear Search 20](#_Toc88185540)

[2.2.3 End with Linear Search 21](#_Toc88185541)

[3. - Graphical User Interface 23](#_Toc88185542)

[3.1 - Wire Frames 23](#_Toc88185543)

[3.1.1 - Wire Frames Details 24](#_Toc88185544)

[3.2 - User interface 24](#_Toc88185545)

[3.2.1 - Scrapping Section 25](#_Toc88185546)

[3.2.2 - Detail of Scrapping Section 25](#_Toc88185547)

[3.2.3 - Sorting Section 26](#_Toc88185548)

[3.2.4 - Detail of Sorting Section 26](#_Toc88185549)

[3.2.5 - Searching Section 27](#_Toc88185550)

[3.2.6 - Detail of Searching Section 27](#_Toc88185551)

[4. – Technology Stack 28](#_Toc88185552)

[5. – Classes 28](#_Toc88185553)

[6. – Problems Faced During Project 28](#_Toc88185554)

[7. – Data Storage 29](#_Toc88185555)

# 1. Project Detail

## 1.1 - Executive Summary

Program will scrap data from different movies websites such as **YTS**, **MoviesFlix**, and **Melomovies** etc. Scrapped data will include entities such as movie title, movie cast, genre, rating, releasing year, releasing date, movie director and synopsis. We provide a fully user friendly interface in such a way that Program will contain the progress bar of scrapped movies and obviously it will have a facility to pause the scrapping and perform the other features such as sorting and searching and simply start when he wants.

User can **sort** the entities by applying algorithm of his own choice such as by

* Merge sort
* Insertion sort and much more.

And on time of search user also have the facility to apply filters on his search such if he is searching the movie by its title then we’ll provide him facility to search like that if he writes character **A** then program will show him all the movie titles starting with, contain with and end with character **A.**

User also sort the entities according to **Ascending or Descending** order on just one click and there will also multi-Colum sorting in the program.

User can also **save the file** whenever he wants. We also provide him a feature of searching by the basis of his own selected entity such as title wise or on basis of genre or on basis of rating. Program also calculate the time taken by the particular task in seconds.

One of the benefit of this program that we are going to provide to user is that we will help the user to see what type of movies are now on trending and which movie’s rating hold good numbers and on which movie’s genre he should invest. And by with this he can also see that which actor is public’s favorite. By this he can cast him in his next project etc.

## 1.2 - Business Needs

Our project will be focusing on both any movie production company and a random user. If a Production Company is going to view the details of the movies that remain top on rating in recent years and on average which type of genre will be profitable for them according to statistics. Then our project will provide them all the statistics according to their needs. User can also use this project to search the movies of his own desires from data of movies around **1 million** according to his preferences.

## 1.3 - Domains in Real Life

* For user real life domain is **Entertainment.**
* Life domain for film industries is select best choice for their **business**.

## 1.4 - Motivation for Project

Firstly almost all the people have great interest in movies including myself. And with the **completion** of this project we got almost data of 1 million movies in our pc and so we can find any kind of movie without having the access over **Internet.** Secondly as it is mentioned in project requirement that you have get 1 million scrapped data so Movies was the **second** option came in my mind, first one was different products from shopping websites. But our first option we already done practice of it in lab task .And finally it fits according to our interest.

**“A film is the petrified fountain of Thought.”**

**Jean Cocteau.**

## 1.5 - Audience

Actors for project will be

* Streamer
* Film Industry
* Director

**Streamer** willbe any random movie fan who willing to find the best movies of his own taste by applying different filter provide in the project. **Film Industry** in such a way that if particular film industry hiring a best movie director for their movie, then this project will meet all their needs. As similarly the **Director** will select the cast that perform outclass in previous films.

## 1.6 - Technical Details

### 1.6.1 – Entities

Here is the detail of all the entities that we are used in our project for scrapping. These entities will be used for sorting the columns and also in searching section as well. These are the entities that covers all the necessary detail about a movie. Table given below holds all the detail about their name, data type and their description.

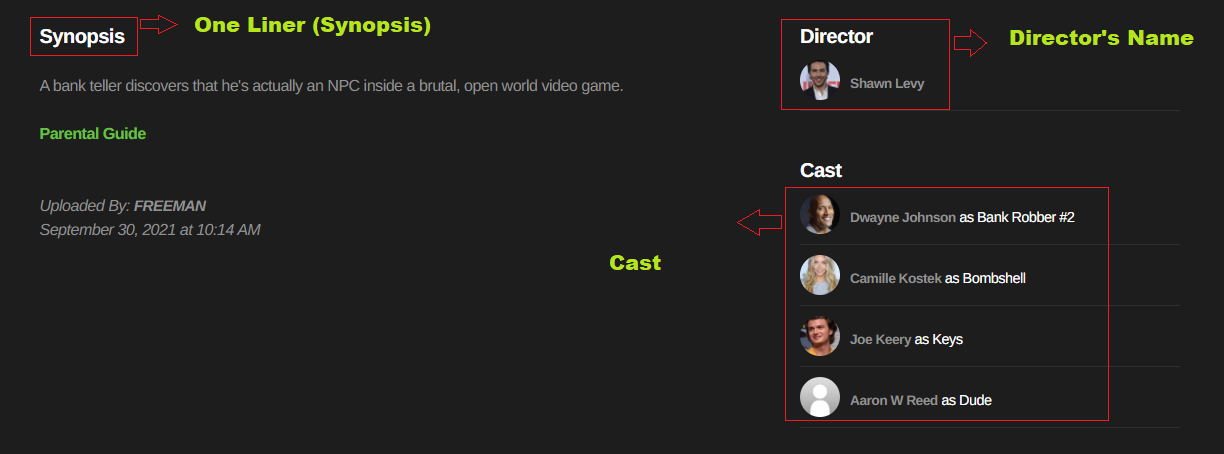
|  |  |  |
| --- | --- | --- |
| ***Name*** | ***Data Type*** | ***Description*** |
| Title | String | It will be Name of the movie |
| Duration | String | Move timing in minutes. |
| Year | String | In string because some movies have year duration such as (2010-2015) |
| Genre | String | Topic of life movie covered. |
| Director | String | The movie creator director. |
| Cast | String | Leading actors and actress |
| Synop | String | One liner of the movie. |

### 1.6.2 - Sample of Scrapping Source

First I started scrapping form https://yts.mx/ and there is the position of the required entities that I scrapped.

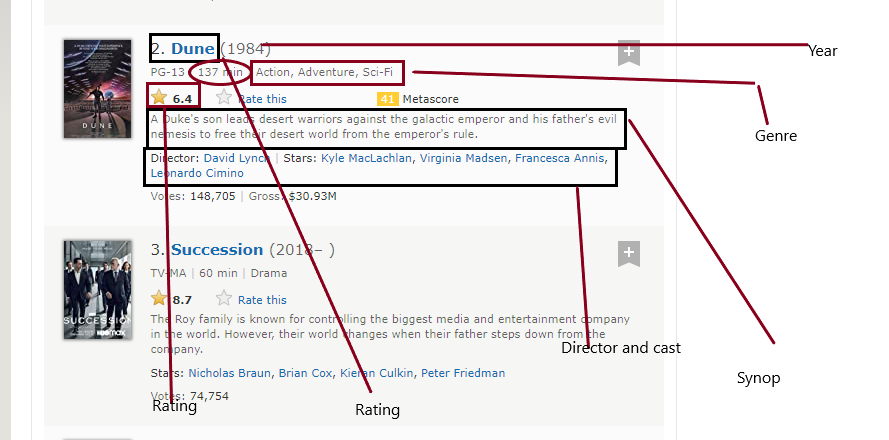
**

***Fig 1.0*** *explain the details of entities from YTS (title, year, genre and rating).*



***Fig 1.1*** *explain the details of entities from YTS (director, cast and Synopsis).*

But the drawback of this website is it contain only 35k movie and also most of the movie details such as director, cast as well as synopsis was missing. That’s why I shifted to Imbd website which is all time best website about movies. The image given below explain all the details from the imbd website.



***Fig 1.3*** *explain the details of entities from IMBD (title, year, time, genre, director, cast, rating and Synopsis)*

### 1.6.3 – GitHub Repository Link

Here is the GitHub repository link of our project.

<https://github.com/The-SaqlainNawaz/CS261F21PID19>

## 1.7 - Project Features

Our project (Scrap the Movies) will be capable to perform following features:

### 1.7.1 - Scrapping Section

* User can select the number of pages he wants to scrap and the scrapped movies will be the multiple of 50 to the no of pages.
* After selecting the no of pages user will get access to **start** button.
* A progress bar will show the no of movies scrapped.
* After all movies scrapped user will get the time taken in scrapping as well as no of movies scrapped.
* User can **Pause/Resume** and **Cancel** the scrapping process at any time.
* At the start of the program user does not have access to the save csv file button, he can only import file from pc.
* User can save the csv file by either pressing the pause button or cancel button. And of course at the end of the scrapping as well.
* All the scrapped movies data will be displayed by the user in the table during the run time of scrapping process.

### 1.7.2 – Sorting Section

* User can sort by either choosing the scrapped movies data from scrapping process or he can also export the csv file and perform sorting on it as well.
* User only get access to list from scrapping button if the scrapping process is completed.
* User can sort the data choosing the provided filters.
* User can also select the **sorting algorithm** from the list provided there.
* User can sort the column of his choice from the list.
* After the sorting completes time spend on sorting will be displayed to user.

### 1.7.3 – Searching Section

* User can search by either importing csv from pc or first perform scrapping process and then selecting that list.
* User can perform search operation on all the columns.
* User can also select the **start** with and **end** with options.
* The result of the searching will be displayed in the table.
* Once file is selected either form scrapping or after importing form pc, then user can save the csv file whenever he wants.
* Time taken on searching will also be displayed in seconds by user.

## 1.8 - Project Plan

The project plan contain the details of the tasks divided among the group members and also mentioned the estimated time during which we tried to complete the task. It is the brief display of how we collaborated during the project.

|  |  |  |
| --- | --- | --- |
| **Assignment Name** | **Member Name** | **Estimated Completion Data** |
| Project Proposal | Saqlain Nawaz | 15 October 2021 |
| Scrapping | Sharjeel Iftikhar  Saqlain Nawaz | 19 October 2021 |
| UI Implementation | Saqlain Nawaz | 24 October 2021 |
| Sorting Algorithms | Sharjeel Iftikhar | 27 October 2021 |
| Integration | Sharjeel Iftikhar  Saqlain Nawaz | 3 November 2021 |
| Searching | Sharjeel Iftikhar  Saqlain Nawaz | 4 November 2021 |
| Project Report | Sharjeel Iftikhar | 5 November 2021 |

# 2. - Algorithms

## 2.1 - Sorting Algorithms

### 2.1.1 - **Insertion Sort**

|  |  |
| --- | --- |
| **Description** | The basic idea of this algorithm is to divide into an **unsorted** and sorted region. Initially, the first term is the only member of the sorted region. The insertion sorts repeatedly scans the list of items, each time inserting the item in the unordered sequence into its correct position.  Then it find the correct place for a given element in a sorted list. |
| **Pseudo Code** | Insertion\_Sort(A)  for i=1 to A.length  key=A[i]  //Insert A[i] into sorted sequence A[1….j-1]  j= i–1  while j>=0 and A[j]>key  A[j+1] =A[j]  J=j–1  A[j+1]=key |
| **Code** | def Insertion\_Sort(arr):      key=0      for j in range(1,len(arr)):          key = arr[j]          i=j-1          while((i>0 or i==0) and arr[i]>key):              arr[i+1]=arr[i]              i=i-1          arr[i+1]=key      return arr |
| **Time Complexity** | The **best-case** time complexity of Insertion Sort is: ***O(n).***  The **worst-case** time complexity of Insertion Sort is: ***O(n²).*** |
| **Proof of Correctness** | We will proof the correctness of the algorithm by using loop invariant.  **Initialization:** As before the start of the loop key is initialized to zero and **j=1** and **i=0** which indicate the array of one element and as we know single element is always sorted.  **Maintenance:** The outer **for** loop selects element A[ j] and key element and positions it properly into A[1.. j-1] via the while loop and iterate j to **j+1**. Since the array A[1.. j-1] before j is sorted, inserting element A[j] into the proper place produces A[1.. j] in sorted order.  **Termination:** for loop will end if j become greater than length of array i.e. **j=n+1.** And the array before j-1 A[1.. j-1] is present in sorted form. |
| **Three Strengths** | * Space requirement is minimum. * **Stable** doesnotchange the relative order of key. * Main advantage of insertion sort is its simplicity and efficiency. * It has good ability to sort small lists. |
| **Three Weakness** | * It requires **n2** number of steps for sorting. * It does not perform well with large numbers. * It is not friendly for **practical** applications. |
| **Dry Run** | Main array  [6, 8, 9, 5, 4, 7, 2, 1]  Dry run  [6, 8, 9, 5, 4, 7, 2, 1]  [5, 6, 8, 9, 4, 7, 2, 1]  [4, 5, 6, 8, 9, 7, 2, 1]  [4, 5, 6, 7, 8, 9, 2, 1]  [2, 4, 5, 6, 7, 8, 9, 1]  [1, 2, 4, 5, 6, 7, 8, 9] |

### 2.1.2 - Merge Sort:

|  |  |
| --- | --- |
| **Description** | This algorithm is based on **divide and conquer** rule. **Firstly,** we **s**tartdividing the array into two equal parts until we get the single element in divided arrays. This is called dividing part. Then comes the **Conquer** partinwhich we merge the divided parts of the array into final sorted array. |
| **Pseudo Code** | **MERGE-SORT(A, p, r):**  If p > r  return  q = ( p + r) / 2  MERGE\_SORT(A, p , q )  MERGE\_SORT(A, p +1 , q )  Merge(A, p, q, r )    **Merge( A, p , q , r )**  n1 = q – p + 1  n2 = r – q  Let L[ 1…n1 + 1] and R [1… n2 +1 ] be new arrays  ***For*** *i = 1* ***to*** *n1*  *L[i] = A[ p + i -1 ]*  ***For*** *j = 1* ***to*** *n2*  *R[j] = A[ q + j]*  *i = 1*  *j = 1*  *for k = p* ***to*** *r*  ***if***  *L[i] ≤ R[j]*  *A[k] = L[i]*  *i = i + 1*  ***else*** *A[k] = R[j]*  *j = j + 1* |
| **Code** | #Merge SOrt arry  def Merge\_Sort(arr,start,end):      if(start<end):          mid = (start+end)//2          Merge\_Sort(arr,start,mid)          Merge\_Sort(arr,mid+1,end)          Merge(arr,start,mid,end)      else:          return  function merge  def Merge(arr,start,mid,end):      start -=1      mid -=1      end -=1      left\_arr=[]      right\_arr=[]      for i in range(start,mid+1):          left\_arr.append(arr[i])      for j in range(mid+1,end+1):          right\_arr.append(arr[j])      i=0      j=0      k=start      for k in range(start,end):          while i < len(left\_arr) and j < len(right\_arr):              if(left\_arr[i]<right\_arr[j] or left\_arr[i]==right\_arr[j]):                  arr[k]=left\_arr[i]                  i = i+1              else:                  arr[k]=right\_arr[j]                  j=j+1              k=k+1  # Checking if any element was left          while i < len(left\_arr):              arr[k] = left\_arr[i]              i += 1              k += 1          while j < len(right\_arr):              arr[k] = right\_arr[j]              j += 1              k += 1      return |
| **Time Complexity** | Time complexity of Merge Sort is **O (n log n)** in **best and worst** both case. |
| **Proof of Correctness** | Proof by loop invariant is given by  **Initialization:** As prior to the first iteration of the loop we initialize **k=start** A[start ….k-1] is empty so there will be no element present in left\_arr and right\_arr.  **Maintenance:** suppose that left\_arr[i] ≤ right\_arr[j] .Then left\_arr[i] is the smallest element not yet copied back into A. Because A[start … k – 1] contains the k - p smallest elements, left\_arr[i] into A[k], the subarray A[start… k] will contain k-start+1 smallest elements. Then increment k to k+1 and if left\_arr[i] ≥ right\_arr[j] then reverse the conditions.  **Termination:** k = end+1 the subarray A[start… k – 1], which is A[start… end] contains the k-start = end- start + 1 smallest elements in sorted order. |
| **Three Strengths** | * It has better run time than insertion, selection and bubble so it is quicker in implementation * It is also efficient for large numbers **n,** andalsois **stable.** * It has constant run time. |
| **Three Weakness** | * Even if we provide sorted array it run through whole process. * Slower comparative to the other sort algorithms for smaller tasks. * Use more memory space to store initial split arrays during **dividing step.** |
| **Dry Run** | Main array  [8, 9, 7, 6, 5, 3, 1]  Dry run  [8, 9, 6, 7, 5, 3, 1]  [6, 7, 8, 9, 5, 3, 1]  [6, 7, 8, 9, 5, 3, 1]  [6, 7, 8, 9, 5, 3, 1]  [6, 7, 8, 9, 3, 5, 1]  [6, 7, 8, 9, 1, 3, 5]  [6, 7, 8, 9, 1, 3, 5]  [1, 3, 5, 6, 7, 8, 9]  [1, 3, 5, 6, 7, 8, 9]  [1, 3, 5, 6, 7, 8, 9]  [1, 3, 5, 6, 7, 8, 9]  [1, 3, 5, 6, 7, 8, 9]  [1, 3, 5, 6, 7, 8, 9] |

### **2.1.3 - Selection Sort**:

|  |  |
| --- | --- |
| **Description** | The selection sort works by repeatedly going through the list of items. In the first step we find the **minimum** element from the array and put it in the declared variable in which store the minimum value we find, and at the last of the array the digit stored in declared variable of min is placed at the **beginning** of the array. Now the iteration occur and now the complete above process repeated for the next digit and so on for the last element of the array.  . |
| **Pseudo Code** | *Selection\_Sort(Arr):*  *n = size of Arr*  *for i = 0 to n-1*  *#declare a variable to store the min value*  *min = i*  *# another for loop to check whole array*  *for j = i+1 to n-1*  *if Arr[j] > Arr[i]*  *min = j*  *# Now arrange the position of min value by swapping*  *if min != i*  *swap Arr[min] and Arr[j]* |
| **Code** | # Selection Sort  def Selection\_Sort(arr):      n=len(arr)      for i in range(n):          min = i          for j in range(i+1,n):              if (arr[min]>arr[j]):                  min=j           # Swapping the elemnts          arr[i],arr[min] = arr[min],arr[i]      return |
| **Time Complexity** | Worst case of Selection sort is **O(n2).**  Best case of Selection sort is also **O (n2).** |
| **Proof of Correctness** | Proof by loop invariant is given by  **Initialization:** As in the starting **i=0** and array is empty.  **Maintenance**: We initialize min=i and we iterate through second for loop in which we start from A[i+1 …. N] and find small number than min, if finds then swap it with A[i]. And in the next iteration array before the **i** will be present in sorted array.  **Termination:** loop will break when i=n+1 i.e. i>n. |
| **Three Strengths** | * It uses few operations, so where data movement is costly it is more useful. * It does not depend upon the **initial arrangement** of the data. * It is better for small no of arrays. * Does not need any extra memory |
| **Three Weakness** | * It is not **stable.** * Not suitable for large numbers because of **n2** runtime. * Even if we provide sorted array it run through whole process. |
| **Dry Run** | Main Arrray  [1, 6, 2, 5, 4, 7]  Dry Run  [1, 2, 6, 5, 4, 7]  [1, 2, 4, 5, 6, 7]  [1, 2, 4, 5, 6, 7]  [1, 2, 4, 5, 6, 7]  [1, 2, 4, 5, 6, 7] |

### 2.1.4 - Bubble Sort:

|  |  |
| --- | --- |
| **Description** | Bubble Sort is also referred as **Sink\_Sort.** It is very simple sorting algorithm. It is quite familiar with the insertion sort, the main difference is that in insertion we compare the key to the whole elements of array while in bubble sort we only compare the **adjacent pairs** and swap it according to particular condition. This passing procedure is repeated until no swaps are required, indicating that the list is sorted. In this algorithm the small digits are gradually start moving top of the list. |
| **Pseudo Code** | Bubble\_Sort(arr)  c = arr .length  for i to c -1  flag = false  for j = 0 to c-1  if arr[ j ] > arr[ j+1 ]  swap arr [ j ] and arr [j + 1 ]  flag = true  if flag == false  break  return arr |
| **Code** | # Bubble Sort  def Bubble\_Sort(arr):      control=len(arr)      for i in range(control-1):          flag=False          for j in range(control-1):              if(arr[j]>A[j+1]):                  temp=arr[j]                  arr[j]=arr[j+1]                  arr[j+1]=temp                  flag=True          if(flag==False):              break      return arr |
| **Time Complexity** | The **best-case time** complexity of Bubble Sort is: **O (n).**  The **worst-case time** complexity of Bubble Sort is: **O (n²).** |
| **Proof of Correctness** | Proof by loop invariant is given by  **Initialization:** In the start we initialize **i=0** and flag = falseso the array before i is empty and present in sorted form.  **Maintenance:** the second for loop i+1 and compare the adjacent elements andif condition satisfy then we swap the elements and the array before i is present in sorted array A[0….i]. loop will break if swap did not occur i.e. arr[j] is less then  Arr[j+1] then iteration will occur to **i+1.**  **Termination:** loop will break if i>n and remaining array A[0….n] will be our sorted array. |
| **Three Strengths** | * It is simple to **implement** and easy to understand. * Best advantage of Bubble sort is memory space it require very few space and does **not** need any **temporary memory**. * When we provide sorted array its run time is **O (n)** which is its best case. |
| **Three Weakness** | * Its worst case is **O (n2).** * It is not very useful in practical application (real world). * It does not deal well with a list containing a huge number of items. |
| **Dry Run** | Main array  [5, 2, 6, 1, 4, 7]  Dry Run  [5, 2, 1, 6, 4, 7]  [5, 2, 1, 4, 6, 7]  [2, 5, 1, 4, 6, 7]  [2, 1, 5, 4, 6, 7]  [2, 1, 4, 5, 6, 7]  [1, 2, 4, 5, 6, 7] |

### 2.1.5 - Quick Sort:

|  |  |
| --- | --- |
| **Description** | Quick Sort algorithm is also based on **divide and conquer** rule. In this algorithm we divide the array into two parts just like we did in merge sort but the difference here is we divide on the basis of the **pivot.** The pivot element could be last element of the array or the middle element or the first element of the array. But here I am using the last element as a pivot.  Then according to the pivot element we arrange the array such that elements less than pivot came on **left** and large digits came on right side of the pivot. Then we recursively call the function until our starting index **(low)** < to the end index **(high)**. When this condition fails we find our sorted array. |
| **Pseudo Code** | QuickSort(A ,p ,r):  if p<r:  q=Partition (A, p, r)  QuickSort(A, p,q-1):  QuickSort(A, q+1, r):  Partition (arr , low, high)  // pivot (Element to be placed at right position)  pivot = arr[high];  i = (low - 1) // Index of smaller element and indicates the  // right position of pivot found so far  for (j = low; j <= high- 1; j++)  // If current element is smaller than the pivot  if (arr[j] < pivot)  {  i++; // increment index of smaller element  swap arr[i] and arr[j]  swap arr[i + 1] and arr[high])  return (i + 1) |
| **Code** | #Quick Sort  def quick\_Sort(arr,low,high):      if(low<high):          pi=partition(arr,low,high)          quick\_Sort(arr,low,pi-1)          quick\_Sort(arr,pi+1,high)  Partition Function  def partition(arr,low,high):      # pi is partitioning index, arr[pi] is now at right place      pivot=arr[high]      i=low-1      for j in range(low,high):      # If current element is smaller than the pivot        if(arr[j]<pivot):              i+=1              temp=arr[i]              arr[i]=arr[j]              arr[j]=temp       # Smaller part end and now we swap pivot to its correct place      temp=arr[i+1]      arr[i+1]=arr[high]      arr[high]=temp      return i+1 |
| **Time Complexity** | Time complexity of Quick Sort in **best-case** is **O (n\*logn)**.  Time complexity of Quick Sort in **worst-case** is **O (n²)**. |
| **Proof of Correctness** | **Initialization**: Prior to the first iteration of the loop, i = low - 1 and j = p. Because no values lie between p and i and no values lie between i + 1 and j + 1, the first two conditions of the loop invariant are trivially satisfied.  **Maintenance:** when A[j] > pivot; the only action in the loop is to increment j . After j is incremented, condition 2 holds for A[j – 1] and all other entries remain unchanged when A[j ≤ pivot] the loop increments i, swaps A[i] and A[j], and then increments j . Because of the swap, we now have that A[i≤ pivot], and condition 1 is satisfied. Similarly, we also have that A[j – 1]> x, since the item that was swapped into A[j – 1] is, by the loop invariant, greater than x.  **Termination:** At termination, j = r. Therefore, every entry in the array is in one of the three sets described by the invariant, and we have partitioned the values in the array into three sets: those less than or equal to x, those greater than x, and a singleton set containing x. |
| **Three Strengths** | * It is said to be the best sorting algorithm. * It is able to deal well with huge lists because its worst case is also **n\*log n**. * Space friendly, No additional storage is required. |
| **Three Weakness** | * Its worst-case performance is equal to average case performance of insertion sort. * It is **not stable** rather it is **fragile**. * If the list is already sorted than bubble sort it is not much more **efficient**. * It also not deal well with **negative** integers. |
| **Dry Run** | Main Array  [5, 6, 2, 1, 4, 7]  Dry Run  [5, 6, 2, 1, 4, 7]  [5, 6, 2, 1, 4, 7]  [5, 6, 2, 1, 4, 7]  [5, 6, 2, 1, 4, 7]  [5, 6, 2, 1, 4, 7]  [2, 6, 5, 1, 4, 7]  [2, 1, 5, 6, 4, 7]  [2, 1, 4, 6, 5, 7]  [1, 2, 4, 6, 5, 7]  [1, 2, 4, 5, 6, 7] |

### 2.1.6 - Hybrid Sort:

|  |  |
| --- | --- |
| **Description** | This is the combination of Quick Sort and Insertion sort. As we know Insertion sort perform very well on partially sorted arrays and quick also is also perform nlog(n) times when partition is balanced so we make a hybrid of both of them. It perform in this manner that if our first array length is > 9 then we perform first the quicksort then we check if left one or right one is smaller and on smaller array we perform insertion sort and so on. |
| **Pseudo Code** | QuickSort(A ,p ,r):  if p<r:  if (low +high +1<10)  insertionSort(arr,low,high)  else  if (pivot –low < high – pivot )  hybrid\_Sort(arr,low,pivot-1)  else  opposite of above  Partition (arr , low, high)  // pivot (Element to be placed at right position)  pivot = arr[high];  i = (low - 1) // Index of smaller element and indicates the  // right position of pivot found so far  for (j = low; j <= high- 1; j++)  // If current element is smaller than the pivot  if (arr[j] < pivot)  {  i++; // increment index of smaller element  swap arr[i] and arr[j]  swap arr[i + 1] and arr[high])  return (i + 1) |
| **Code** | def Insertion\_Hybrid(self,arr,low,high,col):          #key=0          for j in range(low+1,high+1):              key = arr[j]              i=j-1              while((i>j or i==j) and arr[i][col]>key[col]):                  arr[i+1]=arr[i]                  i=i-1              arr[i+1]=key          return      # Hybrid Quick Sort (Quick and Insertion)      def Asc(self,arr,low,high,col):          print(high)          while(low<high):              # First check if the array is smallr enough(less than 10) to apply insertion sort or not              if(high-low+1<10):                  self.Insertion\_Hybrid(arr,low,high,col)                  break              else:                  pi=self.partition\_Asc(arr,low,high,col)                  # NOw find out on both sids of pivot that which array is larger so that we can apply                  # recursive hybridsort on it                  if(pi-low<high-pi):                      self.Asc(arr,low,pi-1,col)                      low = pi +1                  else:                      self.Asc(arr,pi+1,high,col)                      high=pi-1 |
| **Time Complexity** | Time complexity is **O(n2).** |
| **Proof of Correctness** | **Initialization**: Prior to the first iteration of the loop, i = low - 1 and j = p. Because no values lie between p and i and no values lie between i + 1 and j + 1, the first two conditions of the loop invariant are trivially satisfied.  **Maintenance:** when A[j] > pivot; the only action in the loop is to increment j . After j is incremented, condition 2 holds for A[j – 1] and all other entries remain unchanged when A[j ≤ pivot] the loop increments i, swaps A[i] and A[j], and then increments j . Because of the swap, we now have that A[i≤ pivot], and condition 1 is satisfied. Similarly, we also have that A[j – 1]> x, since the item that was swapped into A[j – 1] is, by the loop invariant, greater than x.  **Termination:** At termination, j = r. Therefore, every entry in the array is in one of the three sets described by the invariant, and we have partitioned the values in the array into three sets: those less than or equal to x, those greater than x, and a singleton set containing x. |
| **Three Strengths** | * It is able to deal well with huge lists because its worst case is also **n\*log n**. * Space friendly, No additional storage is required. * Main advantage of insertion sort is its simplicity and efficiency. |
| **Three Weakness** | * It is not friendly for **practical** applications. * It requires **n2** number of steps for sorting. |

### 2.1.8 - Tree Sort:

|  |  |
| --- | --- |
| **Description** | Tree sort is an online sorting algorithm. It uses the binary search tree data structure to store the elements. The elements can be retrieved in sorted order by doing an in-order traversal of the binary search tree. Since it is an online sorting algorithm, the elements inserted are always maintained in sorted order. |
| **Pseudo Code** | Class Node:  Data  Node.left  Node.right  Class Tree:  Root=null  Coming node is smaller or equal to root then add it to left child  Else wise add to right side  Inorder traversal for ascending |
| **Code** | class Tree:      def \_\_init\_\_(self):          self.root=None      def insert\_node(self,ddata,root,col):          if(self.root==None):              self.root=ddata          else:              if(ddata.data[col] < root.data[col] or ddata.data[col] == root.data[col]):                  if(root.left==None):                      root.left=ddata                      ddata.parent=root                  else:                      self.insert\_node(ddata,root.left,col)              elif(ddata.data[col] > root.data[col]):                  if(root.right==None):                      root.right=ddata                      ddata.parent=root                  else:                      self.insert\_node(ddata,root.right,col)      def in\_order\_traversal(self,node,col,arr):          if(self.root==None):              print("tree is empty")          else:              if(node.left!=None):                  self.in\_order\_traversal(node.left,col,arr)              arr.append(node.data)              #print(node.data)              #print()              if(node.right!=None):                  self.in\_order\_traversal(node.right,col,arr)          return arr |
| **Time Complexity** | Worst case is **O(n2)**  Best case is **n(logn)** |
| **Three Strengths** | * Tree sort algorithm is as fast as quick sort algorithm. * Tree sort is stable * Best for large number of arrys |
| **Three Weakness** | * Worst case occur when the elements in an array is already sorted. * In the running time of tree sort algorithm is O (n2) |

## 2.2 – Searching Algorithms

### 2.2.1 Direct Linear Search

|  |  |
| --- | --- |
| **Description** | In this algorithm we simply get the input from user and whatever he/she entered we search in the sorted data if user’s input became equal to the array’s element we append that entity into a new array. |
| **Pseudo Code** | Search(arr,input):  new arry  For I to arr(length)  If(item become equal arr[i]) then  Add arr[i] to new arry |
| **Code** | def Simple\_Search(arr,item,col):      new\_lit=[]      for i in range(len(arr)):          if(arr[i][col]==item):              new\_lit.append(arr[i])      return new\_lit |
| **Time Complexity** | Worst case is **O(n).** |

### 2.2.2 Start with Linear Search

|  |  |
| --- | --- |
| **Description** | In this algorithm we simply get the input from user and whatever he/she entered. First we whether the length of our stored data entity is greater or equal to the entered item then we only search it till the length of input item from sorted elements if upto the length of entered item match with the starting digits from stored data entity then we append it to a separate list. |
| **Pseudo Code** | Search(arr,input):  new arry  For i to arr(length)  For j to input(length)  First check length of input is less  Then check for equality  If(item become equal arr[i]) then  Add arr[i] to new arry |
| **Code** | def Search\_StartWith(arr,item,col):      newArr =[]      size = len(arr)      for i in range(size):          entit\_size=len(arr[i][col])          k=0          flag=False          count=0            if(entit\_size<len(item)):              i+=1          else:              for j in range(entit\_size):                  while(k!=len(item)):                        if(arr[i][col][j]==item[k]):                          count+=1                          k+=1                          j=j+1                          if(count==len(item)):                              flag = True                      elif(arr[i][col][j]!=item[k]):                          k+=1                          j+=1                      else:                          print()                    if(flag==True):                      newArr.append(arr[i])                      break                  elif(flag==False):                      break      return newArr |
| **Time Complexity** | Worst case is **O(n2).**  Best casealso **O(n2).** |

### 2.2.3 End with Linear Search

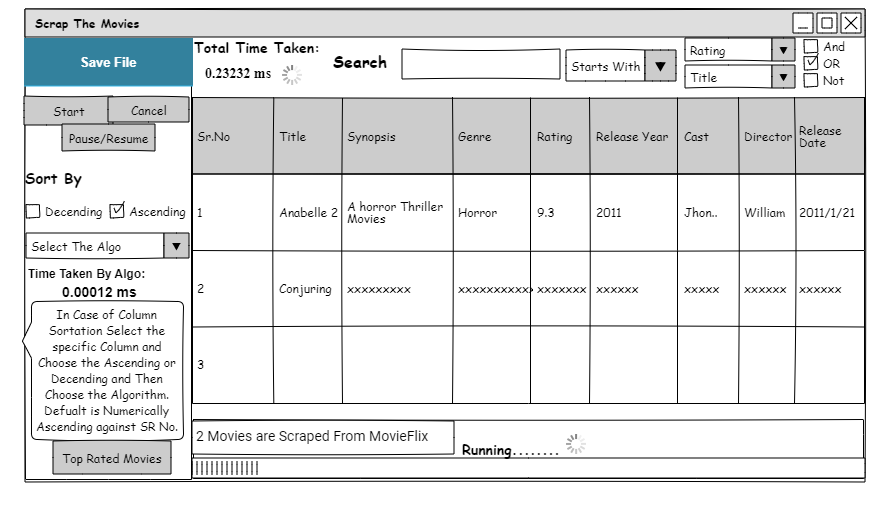
|  |  |
| --- | --- |
| **Description** | In this algorithm we simply get the input from user and whatever he/she entered. First we whether the length of our stored data entity is greater or equal to the entered item then we only search from the end of stored data entity of the length equal to input item. |
| **Pseudo Code** | Search(arr,input):  new arry  For i to arr(length)  For j to input(length)  First check length of input is less  Then check for equality from the end digits  If(item become equal arr[i]) then  Add arr[i] to new arry |
| **Code** | def End\_Search(arr,item,col):      new\_list=[]      size = len(arr)      size\_entit=len(item)      for i in range(size):          s = len(arr[i][col])          if(s<size\_entit):                i+=1          else:              t=(s-size\_entit)              if(t>=size\_entit):                  k=0                  count=0                  flag=False                  for j in range(t,s):                      while(k!=len(item)):                          if(arr[i][col][j]==item[k]):                              k+=1                              j+=1                              count+=1                              if(count==size\_entit):                                 flag = True                          elif(arr[i][col][j]!=item[k]):                              k+=1                              j+=1                          else:                              print()                      if(flag==True):                          new\_list.append(arr[i])                          break                      elif(flag==False):                          break              else:                    i+=1      return new\_list |
| **Time Complexity** | Worst case is **O(n2).**  Best casealso **O(n2).** |

# 3. - Graphical User Interface

GUI is front face of the program displayed to the user and it is all because of user’s ease. So that user can perform any task without any difficulty and operate according to our provided instructions. The detail of the GUI we made for our project is given below.

## 3.1 - Wire Frames

Wire frames are the initial design we made for our project at the time of project proposal.

**

***Fig 3.1*** *explain all the possible operations of wire frame*

And the detail of all the components used in wire frame and their description is explained given below in the table.

|  |  |  |
| --- | --- | --- |
| **UI Component Name** | **UI component Type** | **Description** |
| Pause/Resume | Button | To pause the scraping process. |
| Sort By | Label | To Let user understand the sorting interface. |
| Descending | Check box | To sort the Data in Descending order. |
| Ascending | Check box | To sort the Data in Ascending order. |
| Select The Algo | Combo box | To select the algorithm for sorting. |
| Time Taken By Algo | Label | Head for Time Taken by each algorithm. |
| 0.00000012 ms | Label | To show the time taken by algorithm. |
| Note | Note | To show how each column will sort. |
| Time Taken | Label | Heading for time scraping. |
| 0.239494 ms | Label | Shows the time taken by scraping. |
| Search | Label | Heading for Search operation. |
| Top Rated Movies | Button | Shows the Top ranking movies. |
| Start With | Combo box | To use as filter for searching. |
| Rating | Combo box | To use as filter for searching. |
| Title | Combo box | To use as filter for searching. |
| AND | Check box | To use as filter for searching. |
| OR | Check box | To use as filter for searching. |
| NOT | Check box | To use as filter for searching. |
| Table | Table | To show the scraping data. |
| 2 movies are scraped | Text box | Shows how many |
| Running | Label | Shows the running stat of scrapping. |
| Progress bar | Progress bar | Shows the running stat of scrapping. |

### 3.1.1 - Wire Frames Details

Wire frame detail include all the detail about the number of UI components used in it. And to explain the details of it I use a table given below.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Text Edits | Dropdown | Table | Buttons | Check box | Menu | Labels | Progress bar |
| 2 | 4 | 1 | 6 | 5 | 0 | 10 | 1 |

## 3.2 - User interface

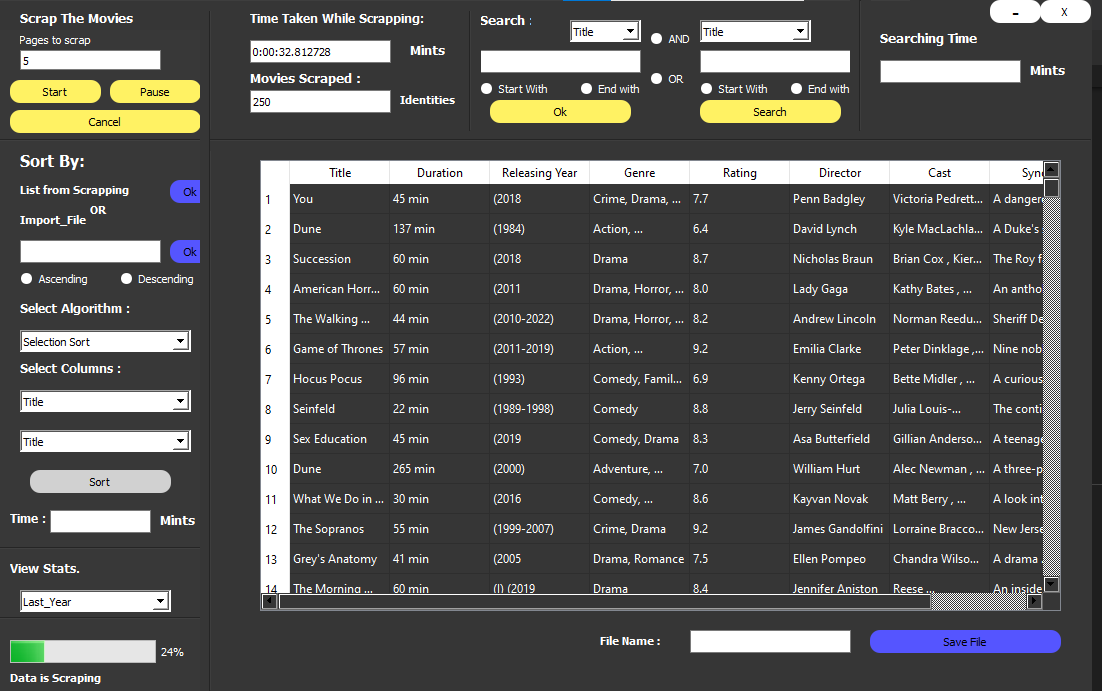
User interface is the final interface of the project. In this we try we give every possibility to the user to operate it easily without facing any difficulty. And it is according to the wire frames provided already to user.

Let’s get all the detail about user interface by dividing it into three parts of

* Scrapping
* Sorting
* Searching

### 3.2.1 - Scrapping Section

This section contain all details about the scrapping process along with detail of its UI components.



***Fig 3.2*** *explain all the operation in scrapping process*

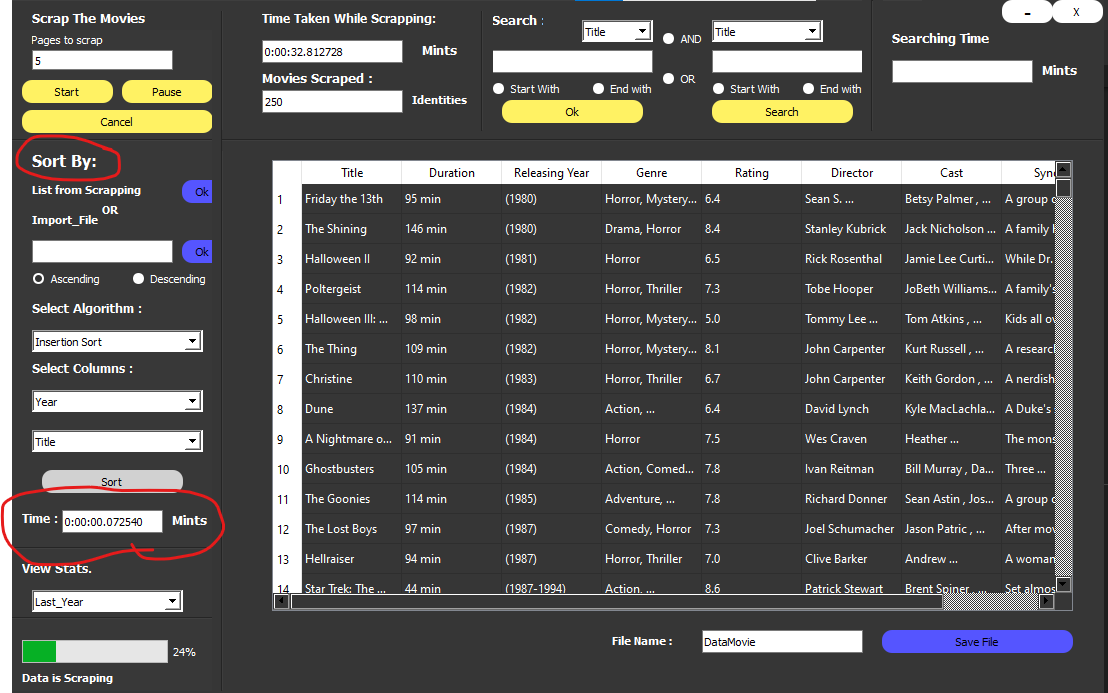
### 3.2.2 - Detail of Scrapping Section

Detail of UI components used in this section along with their description is explained by using a table given below.

|  |  |  |
| --- | --- | --- |
| **UI Component Name** | **UI Component Type** | **Description** |
| Logo | Label | It display the name of the project at the top left corner. |
| Pages to Scrap | Edit Text | To take input from user to scrap the movies pages |
| Start | Button | To start the scrapping process |
| Pause/ Resume | Button | To stop and resume the scrapping at any time. |
| Cancel | Button | To Cancel the scrapping whenever user wants. |
| Mints | Label | To display the time spend on scraping in seconds. |
| Scrapped Entities | Label | To display the total number of scrapped movies |
| File Name | Edit Text | Its purpose is to take name of csv file to be stored in pc. |
| Save File | Button | To save the csv file after writing file name |

### 3.2.3 - Sorting Section

Similarly this section as it is cleared from its name will be used to display all the details of sorting process and also the details of the UI components.



***Fig 3.3*** *covers all the possible operations in sorting process*

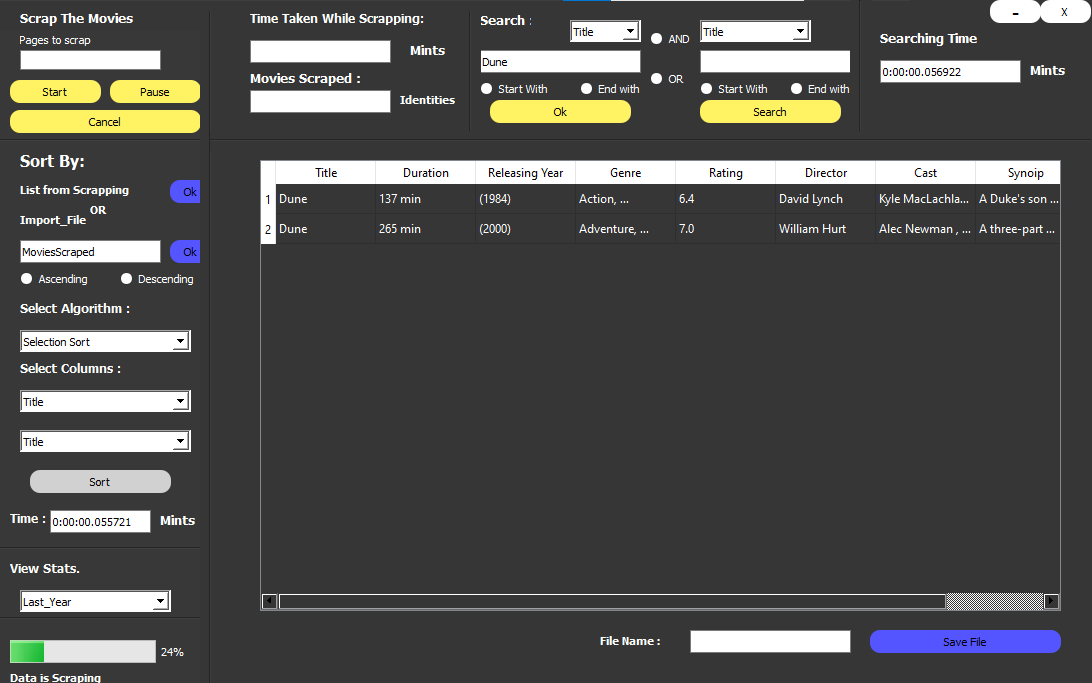
### 3.2.4 - Detail of Sorting Section

Detail of UI components used in this section along with their description is explained by using a table given below.

|  |  |  |
| --- | --- | --- |
| **UI Component Name** | **UI Component Type** | **Description** |
| List from scraping | Button | To use the same list came from scrapping process |
| Import from pc | Button | If user wants to perform all sorting operation on file other than from scraping section |
| Ascending | Radio Btn | To select the filter for sorting |
| Descending | Radio Btn | To sort the array into descending order |
| Select Column | Combo box | It contain list of all sorting algorithms which are implemented to sort the array. |
| Time | Label | To display time spend on sorting |

### 3.2.5 - Searching Section

Similarly this section as it is cleared from its name will be used to display all the details of searching process and also the details of the UI components.



***Fig 3.4*** *show the process of searching*

### 3.2.6 - Detail of Searching Section

Detail of UI components used in this section along with their description is explained by using a table given below.

|  |  |  |
| --- | --- | --- |
| **UI Component Name** | **UI Component Type** | **Description** |
| Search | Combo Box | To select the entity column from which user want to search |
| Name | Edit Text | To get input from user |
| Start with | Radio btn | It is used to get all entities of that column start with entered item |
| End | Radio Btn | It is used to get all entities of that column End with entered item |
| Time | Label | To Display time on searching |
| OK | Button | To start the searching process |

# 4. – Technology Stack

This section explain all the IDEs and Programming language used in this project.

|  |  |
| --- | --- |
| Language(C#/JAVA) | Python |
| Platform | Desktop |
| Frontend Technology | Pyqt5 |
| IDEs | VS Code, QT Designer |

# 5. – Classes

Following table contain detail about the classes used in the project

|  |  |  |  |
| --- | --- | --- | --- |
| **Class Name** | **Domain/Software** | **Abstract** | **Singleton** |
| Scrap the Movies | Domain | **No** | **No** |
| Searching | Domain | **No** | **No** |
| Sorting | Domain | **No** | **No** |
| Scrap the Movies.ui | Software | **No** | **No** |

# 6. – Problems Faced During Project

**During Scraping**

During scrapping firstly I start scraping YTS.max and also I was doing it by using web driver but the problem I faced in it was that the YTS website is not as good as imbd and secondly by using web driver it take extra time in opening window in web driver so then to cover this I shifted to imbd.

Another issue while scraping was that the after 100 pages the sequence of the address to move on next page changes and it again start scraping from page one and I noticed it when I scraped 2 lac movie so then I have to change the method to get the address of next page.

**Threading**

During scraping a major problem I have to face is that at that time pc has to do multi tasks at the same time i.e. first scrap the data from website and then display it in the table in our project so whenever we start the scrapping process “Python stopped working”. So for resolve this problem we used the threading process.

**'utf-8' can't decode bytes** was also the issue came at the time when I was trying to save the csv file. It was not creating csv file because utf-8 was not decoding the format of the release date which contain some characters as well so then I have to update release year format and it resolved easily.

# 7. – Data Storage

In this project csv file is used to store all the scrapped movies data and also he can save the csv file at the run time.

|  |  |
| --- | --- |
| **File Name** | ScrappedMovies.csv |
| **Type** | Excel(CSV)files |
| **Data Format** | Title, Duration, Releasing Year, Genre, Rating, Director, Cast, Synopsis |