LinkedIn Scrapper Tool



Project Supervisor

Mr. Samyan Qayyum Wahla

Project Members(**G19**)

|  |  |
| --- | --- |
| Rafaqat Hussain | 2020-CS-104 |
| Muhammad Nouman Butt | 2020-CS-110 |

Department of Computer Science

University of Engineering and Technology, Lahore

Pakistan

Table of Contents

[Project Description: 4](#_Toc87026336)

[Business need: 5](#_Toc87026337)

[End User: 5](#_Toc87026338)

[Motivation: 5](#_Toc87026339)

[Project Features: 6](#_Toc87026340)

[Project Plan 7](#_Toc87026341)

[Algorithms Detail: 9](#_Toc87026342)

[Graphical User Interface: 36](#_Toc87026343)

[Wire Frames Details: 38](#_Toc87026344)

[Graphical User Interface 39](#_Toc87026345)

[User Interface Details: 42](#_Toc87026352)

[Technology Stack: 43](#_Toc87026353)

[Classes: 43](#_Toc87026354)

[Exceptions: 45](#_Toc87026355)

[Data Storage: 46](#_Toc87026356)

# Project Description:

LinkedIn is now become one of the most famous platforms which is primarily used for professional networking and career development, and allows job seekers to post their CVs and employers to post jobs. LinkedIn has 774+ million registered members from over 200 countries and territories. LinkedIn allows members (both workers and employers) to create profiles and "connect" with each other in an online social network which may represent real-world professional relationships.

This programs “LinkedIn Profile Scrapper” is designed to scrape the details of LinkedIn profile data of different Persons from different countries in the world. The Profile data will consist of their Name, Username, Job designation/ Qualification, Company Name, Number of Connections, Residence, Number of followers, Number of recommendations

.

This will be a graphical single user desktop program where he/she will be provided with 3 screens. First screen will be available for the user to scrape the data with Start, Stop, Resume, Pause and Save Data with the table of loading data. A loading bar will also be shown indicating the amount of scrapping being done.

Second Screen will be shown with all the data scrapped and with the Sorting algorithms available. He/she can choose the algorithm of his choice and can select the column and can sort the column weather in ascending and descending order

.

Third Screen will be shown will all the scraped data and the filters of the data will also be shown. He/she can choose filter of his/her choice according to the need and can search the persons data from it.

# Business need:

LinkedIn Is the biggest Platform for the companies to find a person of their interest with the best possible skills and experience.

From this program, companies can scrape all the existing more than 774+ million profiles and then can filter the data according to their need and can hire or contact particular persons of particular skills and can have communication with them.

# End User:

Different Companies Of any type can use this Program to hire and search their desired faculty/staff. Not even companies but simple user can also use this program to search about other people having same skills to see the competition in the market.

# Motivation:

We just want to provide such a software to the industry which reduces their efforts to find persons of their interest. So that people could be hired sitting at home. In this way, we could serve the society a bit and make a healthy and peaceful environment.

# Project Features:

LinkedIn Scrapper Tool contains mainly many users and provides the following features to end user:

1. **Scrapping Window:**

* User can Select Country and scrape data of his/her preferred Country profiles.
* User can Scrape any number of profiles he/she want.
* User Can Start Scrapping Using just a single button ‘**Start**’.
* The Scrapped Data will be shown into the data at run time as new profile will be scrapped.
* A loading bar will be shown indicating the number of profiles Scrapped.
* User Can Pause/Resume the Scrapping at his/her own.
* User can also Stop the Scrapping at any time.
* User can also export the scrapped data into ‘**CSV**’ by clicking ‘**Export**’ Button
* Scrapping time and the number of profiles scrapped will also be shown below the progress bar.

1. **Sorting Window:**

* User can Import the desired CSV file from PC and the file will be shown into the table.
* User can select the column, sorting algorithm and the order detail of his/her choice from the combo boxes available at bottom to Sort the Entire data.
* Time taken while sorting will be shown also.
* User can also export the sorted data into ‘**CSV**’ by clicking ‘**Export**’ Button.

1. **Searching Window:**

* User can Import the desired CSV file from PC and the file will be shown into the table.
* User can search any entity in the Line Edit and by clicking the ‘**Search**’ button all the profiles containing that entity will be shown in the table.
* User can also export the sorted data into ‘**CSV**’ by clicking ‘**Export**’ Button.
* User can also perform advanced multi-column search by clicking ‘**Advanced Search**’ button.

1. **Advanced Searching:**

* User can Import the desired CSV file from PC and the file will be shown into the table.
* User can choose which column he/she wants to search by checking the ‘**checkbox**’ available at left side.
* Then user can type the entity and choose the operation whether he wants to apply ‘AND’ or ‘OR’ operation between the entities.
* Then the searched data will be shown into the Table by clicking the ‘**Search**’ button.

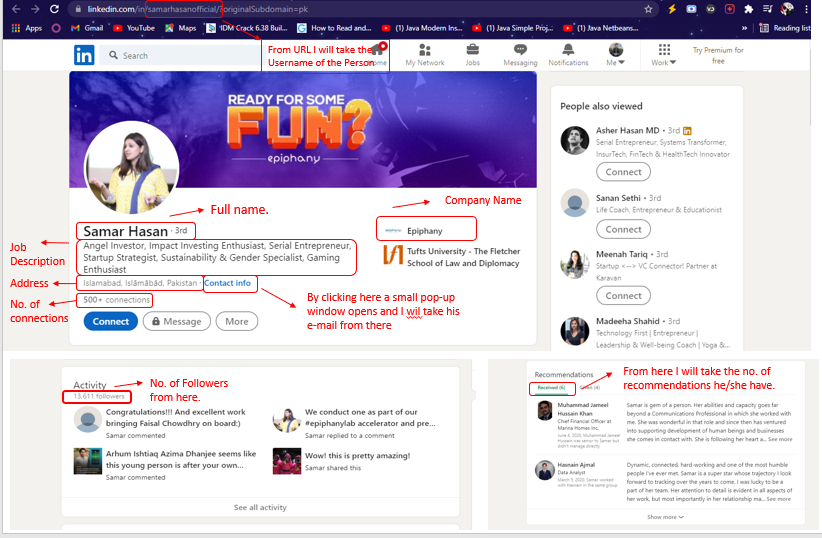
# Project Plan

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Assignment Name** | **Member Name** | **Estimated Completion Date** |
|  | Proposal Report | Rafaqat Hussain | 15 Oct 2021 |
|  | Scrapping | Nouman | 19 Oct 2021 |
|  | UI Implementation | Rafaqat Hussain | 24 Oct 2021 |
|  | Sorting Algorithms | Rafaqat Husaain | 27 Oct 2021 |
|  | Integration | Nouman | 3 Oct 2021 |
|  | View bugs and Linear searching | Rafaqat Husaain | 3 Oct 2021 |
|  | Multi Searching | Nouman | 4 Oct 2021 |
|  | Project Report | Rafaqat Hussain | 5 Oct 2021 |

**Scrapping Source:**

I am Scrapping the profiles of **LinkedIn** users from all over the World while opening their individuals’ profiles in LinkedIn and scrapping the details such as name, username, job designation, company, connections, address, country etc.

I search about profiles on Bing and get the data of each page and open profiles one by one and retrieve their profiles in this way. An Example of a LinkedIn profiles is shown Below and the entities are highlighted.

**

# Algorithms Detail:

**Selection Sort:**

|  |  |
| --- | --- |
| **Description** | Selection sort is one of the simplest procedures of sorting. It is a comparison-based Algorithm. In the first iteration, it first finds the smallest element of the given array and then swaps it with the first element of the array and then the array divides into 2 arrays. One is sorted which contains 1 element and the other is sorted which contains all other elements. Then in the next iteration, it finds the smallest element in the unsorted array and swaps it with the first element of the unsorted array and it becomes part of the sorted Array. Similarly, it continues to sort the given elements, and at the end sorted array is returned. |
| **Pseudo Code** | list: array of items  n: Length of list  for i = 1 to n - 1  min = i  for j = i+1 to n  if list[j] < list[min]  min = j  if indexMin != i  swap list[min] and list[i] |
| **Code** | def Selection(arr,size):      for i in range(0,size-1):          min = i          for j in range(i+1,size):              if(arr[j]<arr[min]):                  min=j          if(min !=i):              arr[i],arr[min]=arr[min],arr[i]      return arr |
| **Time Complexity** | |  |  | | --- | --- | | **Cases** | **Time Complexity** | | Best Case | n2 | | Average Case | n2 | | Worst Case | n2 | |
| **Proof of Correctness** | Proof of correctness using loop invariance is shown below:  The elements in ‘array’[1….i-1], are the smallest of the array in sorted order  **Initialization:** At i = 0, the array is empty, since there are no smallest elements in the array and the condition holds.  **Maintenance:** There are 2 sub-arrays, one consists of [0 … i] which is sorted and other consists of [i+1 … n] elements order.  At i = i+1, the min\_index is equal to the value in A[i]. The for loop then finds the smallest element in the next sub-array, and if it does, it swaps with the A[i].  In the next iteration, the sub array ‘array’ [0...i] does indeed consist of the smallest numbers in sported order, as now the sub-array becomes [1 … i].  **Termination:** The loop invariant terminates when i = n +1, i.e., i > n  Therefore, selection Sort is correct. |
| **Three Strengths** | 1. It works very well for a small number of inputs.  2. It will perform excellently on the array that is already sorted, as no element would be swapped in this case.  3. It does not require a lot of space, as it works in the original array and no other new array is used. |
| **Three Weakness** | 1. It would not perform well for a large number of inputs.  2. The time complexity is n^2 in the worst case and it will consume more time.  3. Its efficiency decreases with the increase in a number of inputs.  4. It is not a stable sort. |
| **Dry Run** |  |

**Insertion Sort:**

|  |  |
| --- | --- |
| **Description** | Insertion sort is also a comparison-based sorting algorithm. It is inspired by the way in which we sort playing cards.  Before starting the implementation, we divide the given array into a sorted array (which contains only the first element of the array) and an unsorted array (which contains the remaining elements of the array considered to be unsorted). Now the first iteration will start from the second element of the array considered to be the first element of the unsorted array and take it as “Key element” and compare it with the sorted array elements and place it at its right place by comparison with sorted elements. Then similarly second-time first element of the unsorted array will be taken and will be placed in a sorted array and at the end, a sorted array will be returned. |
| **Pseudo Code** | INSERTION-SORT(A)  for i = 1 to n  key =A [i]  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(A):        for i in range(1, len(A)):          key = A[i]          j = i - 1            # Compare key with each element on the left of it until an element smaller than it is found          # For descending order, change key<array[j] to key>array[j].          while j >= 0 and key < A[j]:              A[j + 1] = A[j]              j = j - 1            # Place key at after the element just smaller than it.          A[j + 1] = key |
| **Time Complexity** | |  |  | | --- | --- | | **Cases** | **Time Complexity** | | Best Case | n | | Average Case | n2 | | Worst Case | n2 | |
| **Proof of Correctness** | Proof of correctness using loop invariance is shown below:  **Initialization:** As j=2 therefore, A [1 … j-1] = A [1] array of single element is always sorted.  **Maintenance:** For a particular j, A [1 … j-1] is sorted while loop -> A [j] in the correct position A [1 … j] is now sorted. At the beginning of next iteration j becomes j+1 A [ 1 … j] is sorted -> which means loop invariant holds.  **Termination:** The ‘for’ loop terminates when j > n (i.e., j = n+1) The subarray is A [1 … n] by definition is in sorted order.  Therefore, Insertion sort is correct. |
| **Three Strengths** | * It works very well for a small number of inputs. * It would not spend much time if the array is already sorted, * It does not require a lot of space, as it works in the original array and no other new array is used * It is stable sort |
| **Three Weakness** | * It works very well for a small number of inputs. * Its time complexity of the worst case is n^2 * It would take more time if the array is entirely unsorted and it has to do n-1 swapping. |
| **Dry Run** |  |

**Merge Sort:**

|  |  |
| --- | --- |
| **Description** | Merge sort is a famous sorting algorithm. It uses a divide and conquer paradigm for sorting. It divides the problem into sub-problems and solves them individually and then the sub-problems are further divided into more sub-problems and are solved. It then combines the results of sub-problems to get the solution to the original problem. e.g., we have divided a given array into 2 arrays A, B and then we will merge it into a new array C in a sorted manner. It is also a stable Sort. |
| **Pseudo Code** | mergeSort (array, low,high)  If size =1  Return array  Else  m= (low+ high)/2  mergeSort (array, low, m)  mergeSort(array , m+1 , high);  merge (array ,low, m , high)  def Merge(A,a,m,b):  R= [] of size m+1-a  L= [] of size B-m  for i in range a to m+1:  L.append(A[i])  for j in range m+1 to n+1  R.append(A[j])  i=0  j=0  for k in range a to b+1  if(i<len(L) and j<len(R) and L[i]<R[j]):  A[k]=L[i]  i=i+1  elif(j<len(R) and i<len(L)and L[i]>=R[j]):  A[k]=R[j]  j=j+1  elif(i<len(L)):  A[k] =L[i]  i=i+1  elif(j<len(R)):  A[k]=R[j]  j=j+1  return A |
| **Code** | def merge(X,a,m,b):        left\_copy = X[a:m + 1]  # Copy first half into one Array      right\_copy = X[m+1:b+1] # Copy Second Half into Second Array        # Necessary Variables      left\_ind = 0      right\_ind = 0      sort\_ind = a        # This Loop will Copy the Element from Both Left and Right Array in a sorted Manner      while left\_ind < len(left\_copy) and right\_ind < len(right\_copy):          if left\_copy[left\_ind] <= right\_copy[right\_ind]:              X[sort\_ind] = left\_copy[left\_ind]              left\_ind = left\_ind + 1          # Opposite from above          else:              X[sort\_ind] = right\_copy[right\_ind]              right\_ind = right\_ind + 1          # Regardless of where we got our element from          # move forward in the sorted part          sort\_ind = sort\_ind + 1        # We ran out of elements either in left\_copy or right\_copy      # so we will go through the remaining elements and add them      while left\_ind < len(left\_copy):          X[sort\_ind] = left\_copy[left\_ind]          left\_ind = left\_ind + 1          sort\_ind = sort\_ind + 1      while right\_ind < len(right\_copy):          X[sort\_ind] = right\_copy[right\_ind]          right\_ind = right\_ind + 1          sort\_ind = sort\_ind + 1 |
| **Time Complexity** | |  |  | | --- | --- | | **Cases** | **Time Complexity** | | Best Case | n \* log n | | Worst Case | n \* log n | |
| **Proof of Correctness** | Proof of correctness using loop invariance is shown below:  **Initialization**: Prior to the first iteration, k = a, so the subarray X[ a … k-1] is empty and k-a = 0 elements. I = j = 1, no left\_copy[i] and right\_copy[j] contain smallest elements not copied back into X.  **Maintenance**: Lets consider left\_copy[i] <= right\_copy[j]  Left\_copy[i] -> smallest element not copied into A  X[a … k-1] contain k-a smallest elements.  Now X[k] = Left\_copy[i] therefore, X[a … k] contains k-a+1 smallest elements.  Now I is incremented, k is incremented in the for loop.  This help re-establish the loop invariant.  **Termination**: k = r+1. By definition of loop invariant. A [a … k-1] which is A[a … r] contains k-a = r-a+1 smallest elements in sorted order  Therefore, Merge Sort is correct. |
| **Three Strengths** | * It works very well for a larger number of inputs. * It has better time complexity which is n log n. * It has a constant running time. * It is a stable sort. * It is a recursive sorting algorithm. |
| **Three Weakness** | * Merge sort is comparatively slow for small number of inputs. * More memory is used to store elements of sub arrays * It will perform the whole process even if the array is already sorted. |
| **Dry Run** |  |

**Bubble Sort:**

|  |  |
| --- | --- |
| **Description** | Bubble sort is another comparison-based and easiest sorting algorithm to implement. It doesn’t use any extra space while sorting. It uses multiple passes through an array. In each pass, it compares the next element of itself and then swaps the pair if they are in the wrong order and this process goes on. And it continues until a full scan is passed without any swapping of any element means the given array is sorted. |
| **Pseudo Code** | procedure bubbleSort (list: array of items)  loop = list.count;  for i = 0 to loop-1:  swapped = false  for j = 0 to i:  if list[j] > list[j+1]:  swap (list[j], list[j+1])  swapped = true  if (not swapped)  break |
| **Code** | def Bubble(A,size):      n=size-1      for i in range(0,n):          swap=0          for j in range(i):              if(A[j]>A[j+1]):                  temp=A[j]                  A[j]=A[j+1]                  A[j+1]=temp                  swap=1          if(swap==0):              break      return A |
| **Time Complexity** | |  |  | | --- | --- | | **Cases** | **Time Complexity** | | Best Case | n | | Average Case | n2 | | Worst Case | n2 | |
| **Proof of Correctness** | Proof of correctness using loop invariance is shown below:  **Initialization:** At i = 0, the array is empty, since there are no smallest elements in the array and the condition holds.  **Maintenance:** There are 2 sub-arrays, one consists of [0 … i] which is sorted and other consists of [i+1 … n] elements order.  At i = i+1, the algorithm keeps traversing the array, swapping any two unsorted element together.  In the next iteration, the sub array ‘array’ [0...i-1] does indeed consist of the smallest numbers in sported order, as now the sub-array becomes [1 … i].  **Termination:** The loop invariant terminates when i = n +1, i.e., i > n  Therefore, Bubble Sort is Correct. |
| **Three Strengths** | * It is also Stable sort. * It can detect whether the list is already sorted or not. * In the case of the sorted array the time complexity is O (n). * It also works better for a small number of inputs |
| **Three Weakness** | * It works poorly for a larger number of inputs. * It has O (N^2) time complexity. * It would use more memory space. |
| **Dry Run** |  |

**Quick Sort:**

|  |  |
| --- | --- |
| **Description** | Quick Sort is a famous sorting algorithm. It is also a comparison-based algorithm but involves a divide and conquer approach as well. It also follows a recursive algorithm. It divides the given array into 2 arrays using a partitioning element also known as a pivot and the division is done in a way that all the elements on the left side of the pivot are smaller than the pivot and all the elements on the right side of the pivot are greater than the pivot. Also, pivot reaches its original position. |
| **Pseudo Code** | quickSort(arr [], low, high)  if (low < high)  pi = partition (arr, low, high)  quicksort (arr, low, pi - 1)  quicksort (arr, pi + 1, high)  partition (arr [], low, high)  pivot = arr[high]  i = (low - 1)  for (j = low; j <= high- 1; j++)  if (arr[j] < pivot)  i++  swap arr[i] and arr[j]  swap arr [i + 1] and arr[high])  return (i + 1) |
| **Code** | def quickSort(A,low,high):      if( low < high):          # pi is partitioning index, arr[pi] is now          # at right place          pi = partition(A, low, high)          quickSort(A,low, pi -1)          quickSort(A, pi + 1, high)  def partition(A,low,high):      # pivot (Element to be placed at right position)      pivot = A[high]      i = (low - 1) # Index of smaller element and indicates the                    # right position of pivot found so far      # If current element is smaller than the pivot      for j in range(low,high):          if(A[j] < pivot):              i += 1              # increment index of smaller element              A[i], A[j] = A[j], A[i]        A[i+1], A[high] = A[high], A[i+1]      return (i+1) |
| **Time Complexity** | |  |  | | --- | --- | | **Cases** | **Time Complexity** | | Best Case | n\*logn | | Worst Case | n2 | |
| **Proof of Correctness** | Proof of correctness using loop invariance is shown below:  **Initialization:** At the start of each Iteration, I = low-1 and j = low, it creates 3 conditions:   * If low<= k <= low-1, then array[k] <= x, true because no value of k satisfies the equation * If low <= k <= low-1, then array[k] > pivot, true because no value of k satisfies the equation * If k = high, the array[k] = pivot, but since no changes have been made to pivot, this condition is also true   **Maintenance:** At the next Iteration, if A[j] <= pivot A[j] will be swapped with the first element of the right sub-array and the index of last element of left sub-array is increased as I = I +1.  And if A[j] > pivot, then the only change is the last index of the right sub-array and the conditions remains valid because the last index is greater then pivot.  **Termination:** when j = high and therefore the array ‘A’ has been partitioned into 3 sub-arrays, one contains elements less than pivot. 2nd contain greater elements than pivot and 3rd contain element that are equal to pivot.  Therefore, Quick Sort is Correct. |
| **Three Strengths** | * It works very well for a larger number of inputs. * It has better time complexity which is n log n. * No additional storage is required in case of quicksort * If the array split is half then there will be O(n\*(lg\*n)) |
| **Three Weakness** | * It is recursive and if recursion is not available to us then the implementation would be more difficult. * Its time complexity in the worst case is n\* n if the array divides into arrays of 1 and (n-1) . * It has a In-consistent running time. * It is not a stable sort. |
| **Dry Run** |  |

**Tim Sort:**

|  |  |
| --- | --- |
| **Description** | The **Tim sort** algorithm is considered a **hybrid** sorting algorithm made with the help of merge sort and insertion sort. It takes advantage of common patterns of data and solve the large-Scale real-world data.  Insertion sort is the best method to sort when data is already or partially sorted or the length of run is smaller than MIN\_RUN and merge sort is best when the input is large. MIN\_RUN is a value which is a power of 2 not more than 32 (or 64).  Tim Sort applies insertion sort on the small subarrays whose length is less than 32 or 64. And then merge these sorted arrays. |
| **Pseudo Code** | def TIMSORT(array)  minRun = 24  size1 = len(array)  for i in range(0,len(arr),RUN)  arr[x:x+RUN] = self.InsertionSort\_ascend(arr[x:x+RUN],col)  size2 = minRun  while size2 < size1  for i in range(0,size1,size2\*2)  mid = i + size2-1  end = min((i + size2\*2-1,(size1-1)))  left = array[i:mid]  right = array[mid+1:end+1]  arr = merge(left,right)  size2 = size2 \* 2  return array |
| **Code** | def timsort\_ascending(self,arr,col):          RUN = 2          for x in range (0,len(arr),RUN):              arr[x:x+RUN] = self.InsertionSort\_ascend(arr[x:x+RUN],col)          r = RUN          while r < len(arr):              for x in range(0,len(arr),2\* r):                  arr[x:x+ 2\*r] = self.merge\_asccending(arr[x:x+r],arr[x+r:x+2\*r],col)              r = r\*2          return arr |
| **Time Complexity** | |  |  | | --- | --- | | **Cases** | **Time Complexity** | | Best Case | n | | Worst Case | n \* log n | |
| **Proof of Correctness** | Proof of correctness using loop invariance is shown below:  Tim sort divides the array into chunks of 32 elements. Then apply insertion sort into these chunks and then use merge sort in these sorted chunks to make a single sorted array  **Initialization**: Consider each chunk as separated arrays and subarray [0 to i] of each array. Each sub array is sorted because initially i is equal to 0 and the array containing only one element is considered to be sorted.  **Maintenance**: with each iteration a key value (array[i+1]) is compared with the elements of the sorted part and placed in its correct position followed by an increment of i by one. Thus, maintaining the loop invariant.  **Termination**: when i approaches the size of original array, the array is completely sorted meaning array [0 to i] is sorted  After termination each of the chunks are merged using merge sort into a single sorted array  Therefore, Tim Sort is correct. |
| **Three Strengths** | * It works very well for a larger number of inputs. * It has better time complexity which is n log n. * It is a stable sort. |
| **Three Weakness** | * More memory is used to store elements of sub arrays * It will perform the whole process even if the array is already sorted. |

**Shell Sort:**

|  |  |
| --- | --- |
| **Description** | Shell sort is the advanced version of Insertion Sort. It Sorts the element based on the interval basis. It avoids large shifts as used to happens in Insertion Sort. For example, if we take the interval of 4 then it will compare the first element with the 5th element and then reduces the interval as the iterations continues. |
| **Pseudo Code** | shellSort()  A : array of items    while interval < A.length /3 do:  interval = interval \* 3 + 1  end while    while interval > 0 do:  for outer = interval; outer < A.length; outer ++ do:  valueToInsert = A[outer]  inner = outer;  while inner > interval -1 && A[inner - interval] >= valueToInsert do:  A[inner] = A[inner - interval]  inner = inner - interval    A[inner] = valueToInsert  interval = (interval -1) /2; |
| **Code** | n = len(array)              gap = n//2              while gap >= 1 :                for i in range(gap,n):                  temp = array[i]                  j = i - gap                  while j >= 0 and array[j][col] > temp[col] :                    array[j+gap] = array[j]                    j -= gap                  array[j+gap] = temp                gap = gap//2              return array |
| **Time Complexity** | |  |  | | --- | --- | | **Cases** | **Time Complexity** | | Best Case | N \* log N | | Average Case | N\* log N | | Worst Case | n2 | |
| **Proof of Correctness** | Proof of correctness using loop invariance is almost same as insertion sort as insertion sort is proved right then its conditions is also almost same:  **Initialization:** As j=2 therefore, A [1 … j-1] = A [1] array of single element is always sorted.  **Maintenance:** For a particular j, A [1 … j-1] is sorted while loop -> A [j] in the correct position A [1 … j] is now sorted. At the beginning of next iteration j becomes j+1 A [ 1 … j] is sorted -> which means loop invariant holds.  **Termination:** The ‘for’ loop terminates when j > n (i.e., j = n+1) The subarray is A [1 … n] by definition is in sorted order.  Therefore, Shell sort is correct. |
| **Three Strengths** | * This algorithm is quite efficient for medium-sized data sets * It would not spend much time if the array is already sorted, * It does not require a lot of space, as it works in the original array and no other new array is used * It is stable sort |
| **Three Weakness** | * It is not a stable sort. * Its time complexity of the worst case is n^2 |

**Cock Tail Sort:**

|  |  |
| --- | --- |
| **Description** | Cocktail Sort is also known as Bi-directional Bubble Sort. It extends by appling bubble sort by sorting the given array from both sides. |
| **Pseudo Code** | Cocktail\_Sort(A : list of sortable items) is  do  swapped := false  for each i in 0 to length(A) − 2 do:  if A[i] > A[i + 1] then  swap(A[i], A[i + 1])  swapped := true  end if  end for  if not swapped then  // we can exit the outer loop here if no swaps occurred.  break do-while loop  end if  swapped := false  for each i in length(A) − 2 to 0 do:  if A[i] > A[i + 1] then  swap(A[i], A[i + 1])  swapped := true  end if  end for  while swapped // if no elements have been swapped, then the list is sorted |
| **Code** | isSwapped = True              start = 0              end = len(A) - 1              while (isSwapped == True):                  isSwapped = False                  i = start                  while ( i < end ):                      if (A[i][col] > A[i + 1][col]):                          A[i], A[i + 1] = A[i + 1], A[i]                          isSwapped = True                      i += 1                  if (isSwapped == False):                      break                  isSwapped = False                  i = end - 2                  while(  i > start - 1):                      if (A[i][col] > A[i + 1][col]):                          A[i], A[i + 1] = A[i + 1], A[i]                          isSwapped = True                      i = i - 1                  start = start + 1              return A |
| **Time Complexity** | |  |  | | --- | --- | | **Cases** | **Time Complexity** | | Best Case | N2 | | Average Case | n2 | | Worst Case | n2 | |
| **Proof of Correctness** | Proof of correctness using loop invariance is almost the same as bubble sort because it is derived from it. is shown below:  **Initialization:** At i = 0, the array is empty, since there are no smallest elements in the array and the condition holds.  **Maintenance:** There are 2 sub-arrays, one consist of [0 … i] which is sorted and other consists of [i+1 … n] elements order.  At i = i+1, the algorithm keeps traversing the array, swapping any two unsorted element together.  In the next iteration, the sub array ‘array’ [0...i-1] does indeed consist of the smallest numbers in sported order, as now the sub-array becomes [1 … i].  **Termination:** The loop invariant terminates when i = n +1, i.e., i > n  Now reverse all the conditions for next for loop.  Therefore, CockTail Sort is Correct. |
| **Three Strengths** | * It is also Stable sort. * It can detect whether the list is already sorted or not. * It also works better for a small number of inputs |
| **Three Weakness** | * It works poorly for a larger number of inputs. * It has O (N^2) time complexity. * It would use more memory space. |

**Comb Sort:**

|  |  |
| --- | --- |
| **Description** | Comb Sort is the advanced form of bubble Sort. It is also a comparison-based sorting algorithm. Comb Sort uses a gap of size more than 1. The gape in the comb sort starts with the larger value and then shrinks by a factor of 1.3. It means that after the completion of each iteration, the gap is divided by the shrink factor of 1.3. The iteration continues until the gap is 1. At the end the array is sorted and returned. |
| **Pseudo Code** | combsort(array input)  gap := input.size  loop until gap = 1 and swaps = 0  gap := int(gap/1.3)  if gap < 1  //minimum gap is 1  gap := 1  end if  i := 0  swaps := 0  loop until i + gap >= input.size  if input[i] > input[i+gap]  swap(input[i], input[i+gap])  swaps := 1 //Flag a swap has occurred, so the  //list is not guaranteed sorted  i := i + 1 |
| **Code** | def divideNum(self,n):            n = n / 1.3            return 1 if n < 1 else int(n)      def CombSort\_ascend(self,A,col):          if (col != 5):              n = len(A)              swap = True              while not n == 1 or swap == True:                  n = self.divideNum(n)                  swap = False                  for i in range(0, len(A) - n):                        if A[i][col] > A[i + n][col]:                            A[i], A[i + n] = A[i + n], A[i]                          swap = True              return A |
| **Time Complexity** | |  |  | | --- | --- | | **Cases** | **Time Complexity** | | Best Case | N | | Average Case | n2 | | Worst Case | n2 | |
| **Proof of Correctness** | Proof of correctness using loop invariance is almost the same as bubble sort because it is derived from it. is shown below:  **Initialization:** At i = 0, the array is empty, since there are no smallest elements in the array and the condition holds.  **Maintenance:** There are 2 sub-arrays, one consists of [0 … i] which is sorted and other consists of [i+1 … n] elements order.  At i = i+n, the algorithm keeps traversing the array, swapping any two unsorted element after the gap number.  In the next iteration, the sub array ‘array’ [0...i-1] does indeed consist of the smallest numbers in sported order, as now the sub-array becomes [1 … i].  **Termination:** The loop invariant terminates when i = n +1, i.e., i <= 1  Therefore, COMB Sort is Correct. |
| **Three Strengths** | * It is also Stable sort. * It can detect whether the list is already sorted or not. * It also works better for a small number of inputs |
| **Three Weakness** | * It works poorly for a larger number of inputs. * It has O (N^2) time complexity. * It would use more memory space. |

**Searching Algorithms:**

Searching algorithms are designed to check for an element or retrieve an element from any data structure where it is stored. There are many types of searching algorithms just as linear, binary and interval but in this project, I have implemented the following search features with just linear search:

**Single Column**

|  |  |
| --- | --- |
| **Description** | In this searching, User will input any entity he/she wants to, then the function will iterate through the whole data linearly and where it will find the profile containing the searched text then the whole profile will be returned and shown into the table. In this way whole data will be shown in table. |
| **Code** | def search(self,data,key):          final = []            for i in range(len(data)):              tempp = []\*len(data[0])              for k in range(len(data[0])):                  if ( search(key,data[i][k])):                      tempp = data[i]              if len(tempp) != 0:                  final.append(tempp)          return final |
| **Time Complexity** | |  |  | | --- | --- | | **Cases** | **Time Complexity** | | Best Case | N2 | | Average Case | n2 | | Worst Case | n2 | |

**Multi-Column Search**

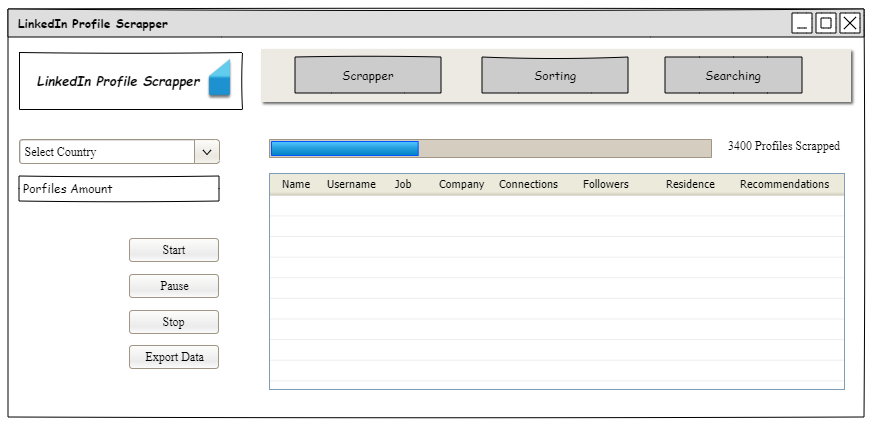
|  |  |
| --- | --- |
| **Description** | In this searching, User has been given the option to search based on 5 whole columns using AND and OR operators.  Checkboxes are available to let the user choose weather he/she wants to search the data using these columns or not. User will select and the function will be called for searching. It will search the result the appended new list will be returned and displayed into the table.  If the search is on two column bases then the time complexity is O(n) otherwise O(n2) |
| **Code** | def Search\_ad(self,data,name,name\_a,name\_col,user,user\_a,user\_col,comp,comp\_a,comp\_col,count,count\_a,count\_col,connec,connec\_col):    final = []          size = len(data)          for i in range (size):              a = data[i][connec\_col].replace("+","")              a = a.replace(",","")              data[i][connec\_col] = a          if ( name\_a == 'AND' ):              for i in range(size):                  if ( search(name,data[i][name\_col]) and search(user,data[i][user\_col]) ):                      final.append(data[i])          elif ( name\_a == 'OR'):              for i in range(size):                  if ( search(name,data[i][name\_col]) or search(user,data[i][user\_col]) ):                      final.append(data[i])          if ( user\_a == 'AND' ):              if( len(final) == 0):                  for i in range(size):                      if ( search(user,data[i][user\_col]) and search(comp,data[i][comp\_col]) ):                          final.append(data[i])              else:                  for j in range(len(final)):                      item = final[j][user\_col]                      for i in range(size):                          if ( search(item,data[i][user\_col]) and search(comp,data[i][comp\_col]) ):                              if (final[j] != data[i]):                                  final.append(data[i])          elif ( user\_a == 'OR'):              if( len(final) == 0):                    for i in range(size):                      if ( search(user,data[i][user\_col]) or search(comp,data[i][comp\_col]) ):                          final.append(data[i])              else:                  for j in range(len(final)):                      item = final[j][user\_col]                      for i in range(size):                          if ( search(item,data[i][user\_col]) or search(comp,data[i][comp\_col]) ):                              if (final[j] != data[i]):                                  final.append(data[i])          if ( comp\_a == 'AND' ):              if( len(final) == 0):                  for i in range(size):                      if ( search(comp,data[i][comp\_col]) and search(count,data[i][count\_col]) ):                          final.append(data[i])              else:                  for j in range(len(final)):                      item = final[j][comp\_col]                      for i in range(size):                          if ( search(item,data[i][comp\_col]) and search(count,data[i][count\_col]) ):                              if (final[j] != data[i]):                                  final.append(data[i])          elif ( comp\_a == 'OR'):              if( len(final) == 0):                  for i in range(size):                      if ( search(comp,data[i][comp\_col]) or search(count,data[i][count\_col]) ):                          final.append(data[i])              else:                  for j in range(len(final)):                      item = final[j][comp\_col]                      for i in range(size):                          if ( search(item,data[i][comp\_col]) or search(count,data[i][count\_col]) ):                              if (final[j] != data[i]):                                  final.append(data[i])          if ( count\_a == 'AND' ):              #print(str(count)+" "+str(connec)+ " " +str(count\_a))              if( len(final) == 0):                    for i in range(size):                      if ( search(count,data[i][count\_col]) and search(connec,data[i][connec\_col]) ):                          final.append(data[i])              else:                  for j in range(len(final)):                      item = final[j][count\_col]                      for i in range(size):                          if ( search(item,data[i][count\_col]) and search(connec,data[i][connec\_col]) ):                              if (final[j] != data[i]):                                  final.append(data[i])          elif ( count\_a == 'OR'):              if( len(final) == 0):                    for i in range(size):                      if ( search(count,data[i][count\_col]) or search(connec,data[i][connec\_col]) ):                          final.append(data[i])              else:                  for j in range(len(final)):                      item = final[j][count\_col]                      for i in range(size):                          if ( search(item,data[i][count\_col]) or search(connec,data[i][connec\_col]) ):                              if (final[j] != data[i]):                                  final.append(data[i])          return final |
| **Time Complexity** | |  |  | | --- | --- | | **Cases** | **Time Complexity** | | Best Case | N | | Average Case | n2 | | Worst Case | n2 | |

# Graphical User Interface:

Graphical User Interfaces are provided to make your program look better into the users eyes and to make it a bit user friendly so that more and more people use it and there would be no difficulty while doing work.

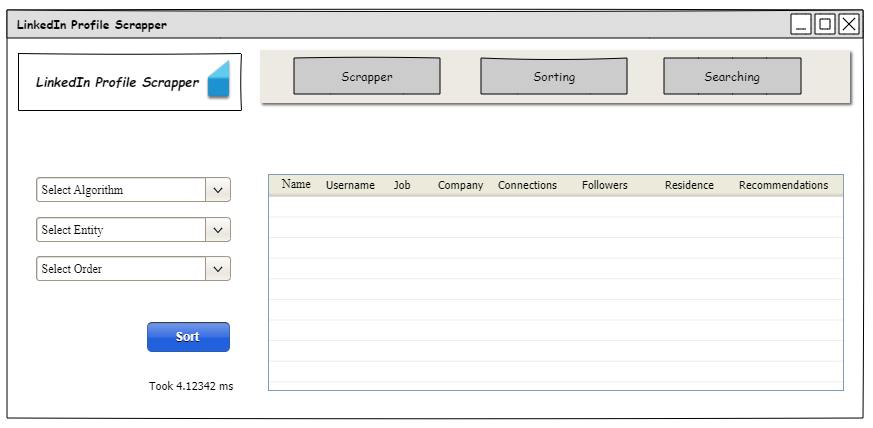
The Graphical User Interface I proposed for the project is shown below:

**Wire Frames**

***UI # 01*****

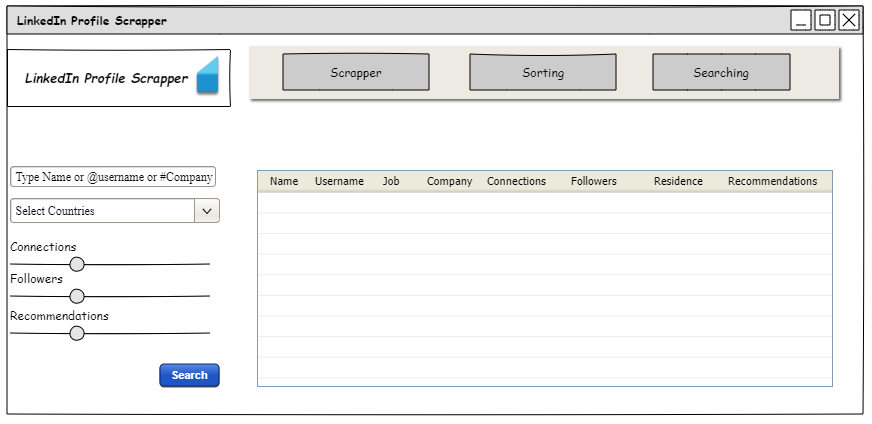
|  |  |  |
| --- | --- | --- |
| **UI Component Name** | **Type of UI component** | **Purpose of UI Component/Other details** |
| Logo | Pic | Its just a logo of Program will be in top left corner. |
| Navigation Bar | Buttons | It is the navigation bar which will direct pages of the program from one to another. |
| Scraping bar | Loading bar | This bar will indicate that how much profiles have been scrapped. |
| Profile’s table | Table | This table will show the scrapped profiles from the LinkedIn. |
| Select Country | Drop-Down menu | This will contain all the countries |
| Profiles Amount | Text | This will ask user to enter the amount of profiles data to scrape. |
| 4 Buttons | Button | These buttons will stop, start resume or load the scraped data into the PC. |

***UI # 02:***

**

|  |  |  |
| --- | --- | --- |
| **UI Component Name** | **Type of UI component** | **Purpose of UI Component/Other details** |
| Logo | Pic | It’s just a logo of Program will be in top left corner. |
| Navigation Bar | Buttons | It is the navigation bar which will direct pages of the program from one to another. |
| Profile’s table | Table | This table will show the scrapped profiles from the LinkedIn. |
| Select Algorithm | Drop-Down | User will select available algorithm from here. |
| Select Entity | Drop-Down | User will select which entity he want to sort accordingly. |
| Select Order | Drop-Down | User will select how to sort it e.g ascending or descending. |

***UI # 03:***



|  |  |  |
| --- | --- | --- |
| **UI Component Name** | **Type of UI component** | **Purpose of UI Component/Other details** |
| Logo | Pic | Its just a logo of Program will be in top left corner. |
| Navigation Bar | Buttons | It is the navigation bar which will direct pages of the program from one to another. |
| Profile’s table | Table | This table will show the scrapped profiles from the LinkedIn. |
| Type Name or Username | Text | This will take name or username or Company name and will search according to it. |
| Select Country | Drop-Down | Countries in scrapped data will show here and user can select it here. |
| 3 Amount Selectors | Scale | This will be used by user to select the amount of entity to search. |

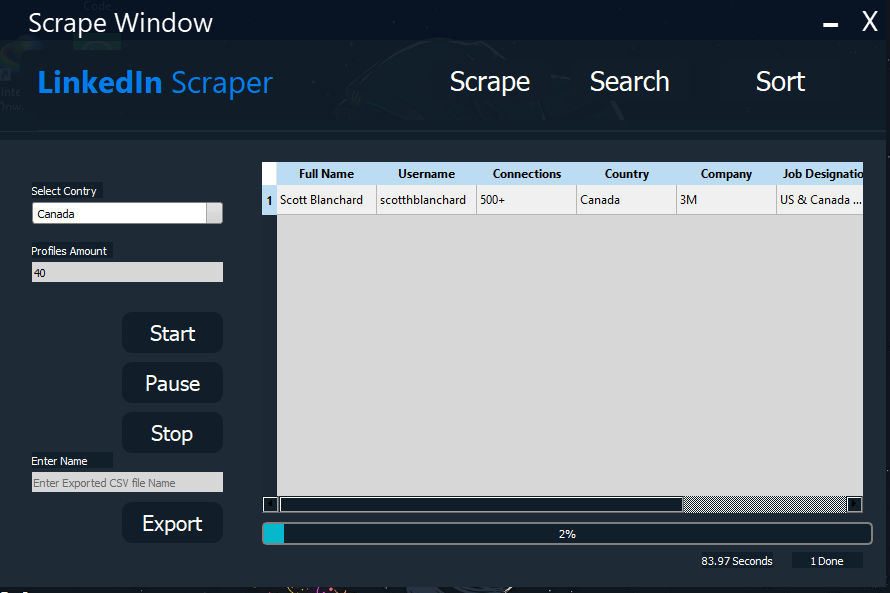
# Wire Frames Details:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Use Case Id | Text Edits | Dropdown | Table | Buttons | Sliders | Checkbox | Menu | Labels | Progress Bar |
| U01 | 1 | 1 | 1 | 7 | 0 | 0 | 0 | 0 | 1 |
| U02 | 0 | 3 | 1 | 4 | 0 | 0 | 0 | 1 | 0 |
| U03 | 1 | 1 | 1 | 4 | 3 | 0 | 0 | 5 | 0 |
|  |  |  |  |  |  |  |  |  |  |

# Graphical User Interface

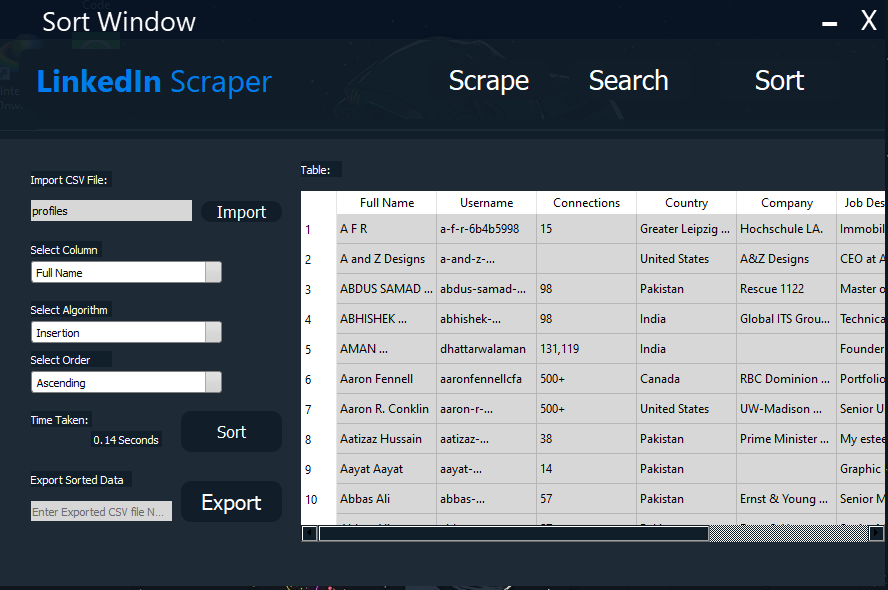
# This is the user Interface which is made for the users to use. All the backhand programming has been applied to it.

# *UI # 01*



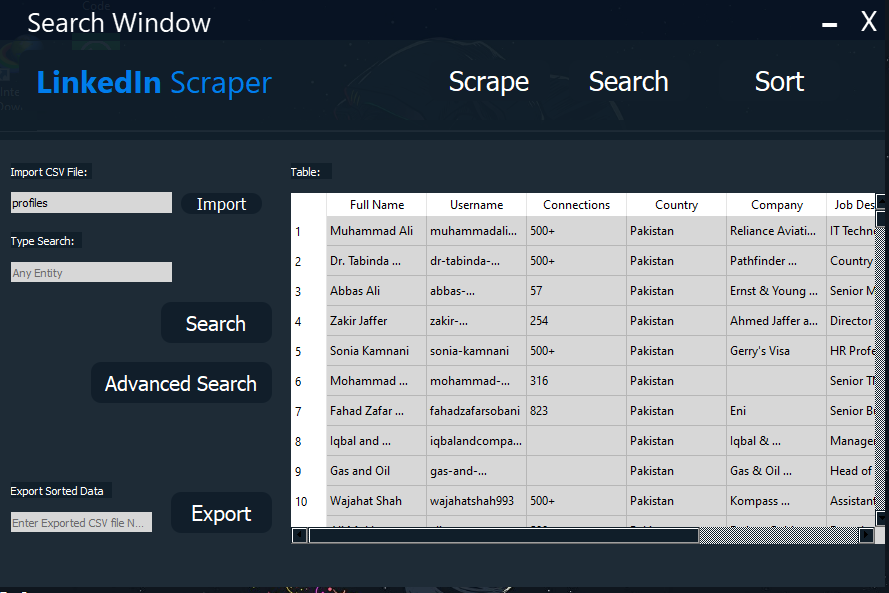
|  |  |  |
| --- | --- | --- |
| UI Component Name | Type of UI component | Purpose of UI Component/Other details |
| Logo | Label | The label represents the program main purpose all the time at the top left corner with some CSS style sheet. |
| Navigation Bar | Buttons (Scrape, Search, Sort) | It is the navigation bar which will direct pages of the program from one to another. |
| Scraping bar | Loading bar | This bar will indicate that how much profiles have been scrapped. |
| Profile’s table | Table | This table will show the scrapped profiles from the LinkedIn. Just as it is showing right now. |
| Select Country | Drop-Down menu | This will contain all the countries to choose it. |
| Profiles Amount | Line Edit | This will ask user to enter the amount of profiles data to scrape. |
| 4 Buttons | Button | These buttons will stop, start resume or load the scraped data into the PC. And will export the scraped data into back to PC. |

# *UI # 02*



|  |  |  |
| --- | --- | --- |
| **UI Component Name** | **Type of UI component** | **Purpose of UI Component/Other details** |
| Logo | Label | The label represents the program main purpose all the time at the top left corner with some CSS style sheet. |
| Navigation Bar | Buttons (Scrape, Search, Sort) | It is the navigation bar which will direct pages of the program from one to another. |
| Profile’s table | Table | This table will show the profiles present in CSV file import from PC. |
| Select Algorithm | Drop-Down | User will select available algorithm from here. |
| Select Entity | Drop-Down | User will select which entity he wants to sort accordingly. |
| Select Order | Drop-Down | User will select how to sort it e.g. ascending or descending. |
| Export Button | Button | This button will take file name from line Edit and will create CSV file of the sorted data in the table into PC. |
| Time Taken | Label | This label will show the Time Taken in the sorting. |

# *UI # 03:*



|  |  |  |
| --- | --- | --- |
| **UI Component Name** | **Type of UI component** | **Purpose of UI Component/Other details** |
| Logo | Label | The label represents the program main purpose all the time at the top left corner with some CSS style sheet. |
| Navigation Bar | Buttons (Scrape, Search, Sort) | It is the navigation bar which will direct pages of the program from one to another. |
| Profile’s table | Table | This table will show the profiles present in CSV file import from PC. |
| Press any entity | Line Edit | User can enter any entity here to search from the table. |
| Advanced Search | Button | User click on this button to do advanced search in the program. Such as multi column searching. |
| Export Button | Button | This button will take file name from line Edit and will create CSV file of the sorted data in the table into PC. |

# *UI # 03:*

# 

|  |  |  |
| --- | --- | --- |
| **UI Component Name** | **Type of UI component** | **Purpose of UI Component/Other details** |
| Logo | Label | The label represents the program main purpose all the time at the top left corner with some CSS style sheet. |
| Navigation Bar | Buttons (Scrape, Search, Sort) | It is the navigation bar which will direct pages of the program from one to another. |
| Profile’s table | Table | This table will show the profiles present in CSV file import from PC. |
| Searching entities | Line- edits | User will enter all the entities of all columns here to search into the column. |
| And or combo | Drop-Down | User Will select which operation to done between entities while searching for entities. |
| Checkboxes | Check box | User will select which columns to search. |

# User Interface Details:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Use Case Id | Text Edits | Dropdown | Table | Buttons | Sliders | Checkbox | Menu | Labels | Progress Bar |
| U01 | 2 | 1 | 1 | 7 | 0 | 0 | 0 | 5 | 1 |
| U02 | 2 | 3 | 1 | 6 | 0 | 0 | 0 | 5 | 0 |
| U03 | 3 | 0 | 1 | 7 | 0 | 0 | 0 | 5 | 0 |
| U04 | 6 | 4 | 1 | 5 | 0 | 4 | 0 | 7 | 0 |

# Technology Stack:

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

# Classes:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Class Name** | **Software/ Domain** | **Is Abstract (Yes/No)** | **Is Singleton (Yes/No)** | **Is the class will has parametrized constructor(Yes/No)** |
| Scrapee | Domain | NO | NO | NO |
| Searching | Domain | NO | No | NO |
| sorting\_algorithms | Domain | No | No | NO |
| Scrape | Software | NO | NO | Maybe(Auto Generated) |

**Problems Arrived During Implementation:**

**Scarping:**

* We wanted to do something different so we choose LinkedIn as our main scraping source.

At the beginning, I tried to do scrapping through google as searching on google this “https

//www.linkedin.com/in/” shows you Billions of profiles of LinkedIn users. We made whole scraping

program but when we implemented then google detected that it’s an auto controlled and made

reCAPTCHA for us to solve. And we could scrape from google.

* Then we shifted to LinkedIn Official website and again from zero made whole program and the scrapping started but when I started scrapping then after 1000 profiles scrapped, we came to know that there is a limit for a single account to view 1000 profiles in one month and we have completed it. And any new account which is freshly made, you can’t view people’s profiles from there so that source also dropped
* Then we used Bing as our source using a lot of time delays then finally, we scrapped almost 1200 profiles and started our project.

**Threading:**

* When we integrated scraping into our program and started then only one functions keep running and the whole python program stop working and “Program not responding” comes and program breaks.

But then after spending 4-5 hours of searching we came to know about threading and it was just 2 lines of code. Then the code run in its usual state.

**Multi-Column Searching:**

* We were trying to allow user to search based on all of 5 entities applying ‘AND’ and ‘OR’ operators as well but after a lot of hard work it was working fine just for 2-3 columns and 5 columns were giving a bit of wrong output.

**Sorting:**

* Sorting Algorithms were working fine but the algorithms which were not comparison-based, we was not able to implement them for strings such as radix, bucket or counting and many more.
* Other sorting algorithms who were working for strings was not working for integers so we had to made them work for both in ascending and descending others.
* Multi-Column searching was being tried but there was not much time for it so it is left behind (Leftover)

# Exceptions:

|  |  |  |
| --- | --- | --- |
| **Type of Exception** | **Why this exception will occur** | **How you will handle the exception** |
| Invalid Convertions | While scraping if the entity was empty then the string was null and while computation converting to integer value gives an Error. | If there is nothing into the string then place ) in the string using Try and catch. |
| File Opening exception | When file is not found | By making a new file or putting the right name. |
| File reading exception | When file is empty | By writing data in it. |
| Out of bound Exceptions | When index out of size of list is accessed | By making a few changes wight now don’t even know |
| Import Error | This Error occurs when you are trying to import a module of other class which doesn’t exist anymore. | Remove that import or make that module. |
| Type Error | When you try to concatenate an integer with string | Convert integer to string first. |
| Missing argument in fuctions | When you pass less or more arguments into function calls | Check and pass Correct Number of Arguments. |
| Circle imports | When you create object of other class In one class and then create object of same class in other class | Copy the functions of other class and make only One class. |

# Data Storage:

**1:**

This will contain all the profiles scrapped from the web and will be available for the user to add to the program rather than just scrapping all the data again.

|  |  |
| --- | --- |
| **File Name** | profiles.csv |
| **Type** | Excel (CSV) File |
| **Data format** | Full name | Job Designation | Company | Address | Country | NO. of Connections | Username |

**2:**

At the Run Time, User can also create his/her own CSV of the sorted, searched or scraped data into the PC using the name of his/her own choice

|  |  |
| --- | --- |
| **File Name** | Name.csv |
| **Type** | Excel (CSV) File |
| **Data format** | Full name | Job Designation | Company | Address | Country | NO. of Connections | Username |