Course Scrapping Application

CS261- Mid-Term Final Report Document



**Project Supervisor**

Mr. Samyan Qayyum Wahla

Project Members(PID-58)

|  |  |
| --- | --- |
| Awais Butt | 2020-CS-62 |
| Abdullah Yaqub | 2020-CS-72 |

**Table of Contents**

|  |  |
| --- | --- |
| **Contents** | **Page Number** |
| Project Description | 3 |
| Project Plans | 3 |
| Graphical User Interface on Pencil Tool | 4 |
| Graphical user interface on QT designer | 4 |
| Sorting Algorithms | 5 |
| Searching Algorithm | 18 |
| Use Cases | 22 |
| User Manual | 29 |
| Integration | 31 |

**Project Description:**

In this project we have to scrap data of different courses according to different professions in the world. The application is designed for the people to get a know-how of the different professions in the world. In this scenario we will have the university and course related information for the sake of sorting in order to provide user-friendly details about each and everything. We will sort our data on the basis of Course Name, Course Title, Universities Offering the specific Subject and also with the starting date of the course and ending expected date of course. We can sort the data in ascending and as well as in descending order. Also we can apply different filters on different columns such as equal to, AND, OR, is null, contains etc. This application will have all the information that a user need regarding the courses. The scrapping websites pages will be from Coursera, Udemy and Edx.

**Project Plans:**

We discussed the project in detail after it was given to us we took a brief view on scrapping first and planned a few mile stones to achieve as discussed:

|  |  |  |
| --- | --- | --- |
| **Milestones** | **Expected Date** | **Achieved By** |
| Scrapping Through Coursera | 21st of October | 2020-CS-62 |
| UI implementation | 24th of October | 2020-CS-72 |
| Sorting Algorithm | 27th of October | 2020-CS-62 & 2020-CS-72 |
| Pseudo Codes | 27th of October | 2020-CS-62 |
| PyQt Desktop Tutorials | 29th of October | 2020-CS-72 |
| Integration | 3rd of November | 2020-CS-62 & 2020-CS-72 |
| Final Report | 5th of November | 2020-CS-62 & 2020-CS-72 |

These were the dates that are provided on we pushed ourselves to made the desktop application as our course scrapping application. We also planned to use some of the concepts that we have not discussed in the class which included the threading and multithreading out of which we were able to use some of the tasks and were unable to use some of the tasks. Some things went our way and some of the things were not helping us in doing the tasks.

**Graphical User Interface in Pencil Tool:**

|  |  |
| --- | --- |
| **Description** | We made a GUI in our pencil tool to further implement it on the PyQT5. |
| **Screen Shot** | FIG: GUI IN PENCIL TOOL |

**Graphical User Interface in QT Designer:**

|  |  |
| --- | --- |
| **Description** | After the pencil tool GUI we made our implementing design in QT designer for further use and implementing the cause of our project**.** |
|  | FIG: GUI IN QT DESIGNER |

**Sorting Algorithms:**

**Insertion Sort:**

|  |  |
| --- | --- |
| **Description** | Insertion sort is a straightforward sorting algorithm that works similarly to how you arrange cards in your hands. The array is divided into two halves, one sorted and the other unsorted. The values from the unsorted component are selected and placed in the sorted part in the proper order. |
| **Pseudo Code** | INSERTION-SORT(A)  1 **for** j = 2 **to** A. *length*  2 key= A[j]  3 **//** Insert A[j] into the sorted sequence A [1. j …­­­­­­---1]  4 i = j - 1  5 **while** i > 0 and A[i]> *key*  6 A[i+1] = A[i]  7 i = i \_ 1  8 A[i+1] = *key* |
| **Code in Python** | def InsertionSort(math):  for i in range(0,len(math)):  key = math[i]  j=i-1  while j>0 and math[j]>key:  math[j+1] = math[j]  j=j-1  math[j+1] = key |
| **Time Complexity Analysis** | * The worst-case time complexity is O(n2) * The average case time complexity is O(n2) * The best-case time complexity is O(n) |
| **Proof of Correctness** | To prove Insertion Sort is correct, you will then demonstrate it for the three stages:   * Initialization - The subarray begins with the array's first element, which is (obviously) ordered by default. * Maintenance - Each loop iteration extends the subarray while keeping the sorted attribute. Only when element V is bigger than the element to the left is it added into the array. Because the items to its left have already been sorted, V is bigger than all of the elements to its left, indicating that the array is still sorted. (This was demonstrated in Insertion Sort 2 by outputting the array each time an element was correctly placed. * Termination - When V reaches the final element in the array, the sorted subarray has grown to embrace the full array, the function will terminate. The array has now been completely sorted. |
| **Dry Run** | Enter the size of Array: 5  Enter the Elements of Array: 5  Enter the Elements of Array: 4  Enter the Elements of Array: 3  Enter the Elements of Array: 2  Enter the Elements of Array: 1  Output: [1, 2, 3, 4, 5] |

**Merge Sort:**

|  |  |
| --- | --- |
| **Description** | Merge Sort Algorithm works on the principle of Divide and Conquer strategy and also divided the bigger problem set into pieces or smaller part. In this algorithm we have to find the solution of the smaller problems and then after finding the solution of those smaller problems we sub-merges the sub-problems and then find the solution of bigger problem. |
| **Pseudo Code** | 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]  L[n1+1] = ∞  R[n2+1] = ∞  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 in Python** | def Merge(arr , s, m, e):  left = arr[s:m+1]  right = arr[m+1:e+1]  i=0  j=0  k=s  while i<len(left) and j<len(right):  if left[i]<right[j]:  arr[k]=left[i]  i=i+1  else:  arr[k]=right[i]  j-j+1  while i<len(left):  arr[k]=left[i]  i=i+1  k=k+1  while j<len(right):  arr[k]=right[j]  j=j+1  k=k+1  def MergeSort(arr,s,e):  if e>s:  m = (s+(s-e)//2)  MergeSort(arr,s,m)  MergeSort(arr,m+1,e)  Merge(arr,s,m,e)  math[j+1] = key |
| **Time Complexity Analysis** | * The worst-case time complexity is O(nlogn) * The average case time complexity is O(n) * The best-case time complexity is O(nlogn) |
| **Proof of Correctness** | The algorithm can be proven by the divide and conquer rule in which all the array is divided into parts and when the array size is equal to 1 then the array is already sorted. So, we can prove the correctness of the algorithm through induction first we will have the base case equal to 1 and then we will confirm it for some arbitrary value “k” and if it is true then it will be true for all the values. |
| **Dry Run** | Enter Size of Array: 6  Enter Elements of Array:4  Enter Elements of Array:3  Enter Elements of Array:2  Enter Elements of Array:1  Enter Elements of Array:4  Enter Elements of Array:1  **Output: [1, 2, 3, 4, 4, 5]** |

**Selection Sort:**

|  |  |
| --- | --- |
| **Description** | Selection sort algorithm is an efficient algorithm that is based on the swapping of the element when it is compared with the minimum element. If the element is not equal to the minimum then it is swapped and at the result of that we get our array in the sorted form |
| **Pseudo Code** | for i to i-1:  min=i  for j=i+1 to n  if list[j]<list[min]  min=j  if indexMin!=i  swap last[min] and list[i] |
| **Code in Python** | def SelectionSort(arr):  for i in range(0,len(arr)-1):  min=i  for j in range(i+1, len(arr)):  if arr[j]<arr[min]:  min = j  if min != i:  temp=arr[min]  arr[min]=arr[i]  arr[i]=temp |
| **Time Complexity Analysis** | * The worst-case time complexity is O(n2) * The average case time complexity is O(n2) * The best-case time complexity is O(1) |
| **Proof of Correctness** | We will have the proof of correctness of algorithm by defining the algorithm for array with size 1. It return the sorted array and afterwards we will our array with the size n and it will perform n2 steps and will return our sorted array then we will increase the size by n+1 the array will have the sorted array with (n+1)2 operations then we will say that our algorithm is correct. We have proven it by induction process. |
| **Dry Run** | Enter Size of array:7  Enter the element of array:4  Enter the element of array:3  Enter the element of array:5  Enter the element of array:8  Enter the element of array:9  Enter the element of array:2  Enter the element of array:4  Output: [2,3,4,4,5,8,9] |

**Bubble Sort:**

|  |  |
| --- | --- |
| **Description** | Bubble sort algorithm is a sorting algorithm for integers and strings as well. What makes is efficient is iteration through the array and swapping each and every element of the array which is minimum to the other and doing that makes the array sorted in the case. |
| **Pseudo Code** | BUBBLESORT(A)  for i = 1 to A.length-1  for j = A.length downto i+1  if A[j]<A[j-1]  exchange A[j] with A[j-1] |
| **Code in Python** | def BubbleSort(math):  temp=0  for i in range (0,len(math)-1):  swapped = False  for j in range(0,len(math)-1):  if math[j]==0:  var = int(math[j])  math[j]=math[j+1]  math[j+1]=var  swapped=True  if not swapped:  break |
| **Time Complexity Analysis** | * The worst-case time complexity is O(n2) * The average case time complexity is O(n2) * The best-case time complexity is O(n) |
| **Proof of Correctness** | We will have the proof of correctness of algorithm by defining the algorithm for array with size 1. It will return the sorted array and afterwards we will our array with the size n and it will perform **n2**steps and will return our sorted array then we will increase the size by n+1 the array will have the sorted array with **(n+1)2** operations then we will say that our algorithm is correct. We have proven it by induction process. |
| **Dry Run** | Enter the size of Array: 6  Enter Elements of Array: -8  Enter Elements of Array: -7  Enter Elements of Array: 2  Enter Elements of Array: 4  Enter Elements of Array: -1  Enter Elements of Array: 4  Output: [-8, -7, -1, 2, 4, 4] |

**Heap Sort:**

|  |  |
| --- | --- |
| **Description** | Heap sort algorithm uses the concept of binary search tree for the sorting of integers and string as well and in this case we assume the root node of the tree as the largest element and through which after comparison the array which is returned is a sorted one by the traversal of all the elements of search tree. Where the minimum element found it just swaps the elements |
| **Pseudo Code** | Max-Heapify(A, i)  l = LEFT(i)  r = RIGHT(i)  if l≤A.heap-size and A[l]>A[i]  largest = l  else largest = i  if r≤A.heap-size and A[r]>A[largest]  largest = r  if largest != i  Exchange A[i] with A[largest]  Max-Heapify[A,largest] |
| **Code in Python** | def heapify(arr, n, i):  largest = i  l = 2\*i+1  r = 2\*i+2  if l<n and arr[i]<arr[l]:  largest = r  if r<n and arr[largest]<arr[r]:  largest = r  if largest != i:  arr[i],arr[largest]=arr[largest],arr[i]  heapify(arr, n, largest)  def heapsort(arr):  n=len(arr)  for i in range(n//2-1,-1,-1):  heapify(arr, n, i)  for i in range(n-1, 0, -1):  arr[i],arr[0]=arr[0],arr[i] |
| **Time Complexity Analysis** | * The worst-case time complexity is O(nlogn) * The average case time complexity is O(nlogn) * The best-case time complexity is O(n) |
| **Proof of Correctness** | We will have the proof of correctness of algorithm by defining the algorithm for array with size 1. It return the sorted array and afterwards we will our array with the size n and it will perform **n(log(n))operations** and will return our sorted array then we will increase the size by n+1 the array will have the sorted array with **(n+1(log(n+1)))** operations then we will say that our algorithm is correct. We have proven it by induction process. |
| **Dry Run** | Enter Size of Array:7  Enter Element of Array:-3  Enter Element of Array:-1  Enter Element of Array:-10  Enter Element of Array:3  Enter Element of Array:5  Enter Element of Array:99  Enter Element of Array:56  Output: [-10,-3,-1,3,5,56,99] |

**Quick Sort:**

|  |  |
| --- | --- |
| **Description** | Quick sort algorithm is an efficient algorithm that follows the divide and conquer approach for sorting. In the case the array is divided on the basis of the pivot that is an arbitrary value from the array and on the base of that pivot point the whole array is divided into right and left part and then compared and the whole is merged, the merged array we get is in the sorted form. |
| **Pseudo Code** | QuickSort(A,p,r)  If p<r  q=PARTITION(A,p,r)  QuickSort(A,p,q-1)  QuickSort(A,q+1,r) |
| **Code in Python** | def QuickSort(arr,low,high):  if low<high:  part = Partition(arr,low,high)  QuickSort(arr, low, part-1)  QuickSort(arr,part+1,high)  def Partition (arr,low,high):  pivot=arr[high]  i=low-1  for j in range(low,high):  if arr[j]<pivot:  i=i+1  arr[i],arr[j]=arr[j],arr[i]  arr[i+1],arr[high]= arr[high],arr[i+1]  return i+1 |
| **Time Complexity Analysis** | * The worst-case time complexity is O(nlogn) * The average case time complexity is O(nlogn) * The best-case time complexity is O(n) |
| **Proof of Correctness** | We will have the proof of correctness of algorithm by defining the algorithm for array with size 1. It return the sorted array and afterwards we will our array with the size n and it will perform **n(log(n))operations** and will return our sorted array then we will increase the size by n+1 the array will have the sorted array with **(n+1(log(n+1)))** operations then we will say that our algorithm is correct. We have proven it by induction process. |
| **Dry Run** | Enter size of Array:7  Enter Elements of Array:-3  Enter Elements of Array:-1  Enter Elements of Array:-10  Enter Elements of Array:3  Enter Elements of Array:5  Enter Elements of Array:99  Enter Elements of Array:56  Output: [-10,-3,-1,3,5,56,99] |

**Hybridized Algorithm:**

|  |  |
| --- | --- |
| **Description** | A hybrid algorithm is the one which is based on the number of inputs defined. It used the efficient algorithm on the basis of input. If the input is large it will use the merge sort algorithm and if the input is small it will apply the insertion sort algorithm which makes the algorithm efficient. |
| **Pseudo Code** | INSERTION-SORT(A)  **for** j = 2 **to** A. *length*  key= A[j]  **//** Insert A[j] into the sorted sequence A [1. j …­­­­­­---1]  i = j – 1  **while** i > 0 and A[i]> *key*  A[i+1] = A[i]  i = i \_ 1  A[i+1] = *key*  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]  L[n1+1] = ∞  R[n2+1] = ∞  i=1  j=1  for k=p to r  if L[i]≤R[j]  A[k] = L[i]  i=i+1 |
| **Code in Python** | def InsertionSort(math):  for i in range(0,len(math)):  key = math[i]  j=i-1  while j>0 and math[j]>key:  math[j+1] = math[j]  j=j-1  math[j+1] = key  def Merge(arr , s, m, e):  left = arr[s:m+1]  right = arr[m+1:e+1]  i=0  j=0  k=s  while i<len(left) and j<len(right):  if left[i]<right[j]:  arr[k]=left[i]  i=i+1  else:  arr[k]=right[i]  j-j+1  while i<len(left):  arr[k]=left[i]  i=i+1  k=k+1  while j<len(right):  arr[k]=right[j]  j=j+1  k=k+1  def MergeSort(arr,s,e):  if e>s:  m = (s+(s-e)//2)  MergeSort(arr,s,m)  MergeSort(arr,m+1,e)  Merge(arr,s,m,e) |
| **Time Complexity Analysis** | It will have **O(n2)** for larger inputs and **O(nlogn)** for smaller inputs |
| **Proof of Correctness** | We will have the proof of correctness of algorithm by defining the algorithm for array with size 1. It return the sorted array and afterwards we will our array with the size n and if the size of array n is equal to the value at which merge sort preforms better then we will perform **n(log(n))operations otherwise n2** and will return our sorted array then we will increase the size by n+1 the array will have the sorted array with **(n+1(log(n+1))) or (n+1)2** depending on the value of n operations then we will say that our algorithm is correct. We have proven it by induction process. |

**Searching Algorithms:**

**Linear Search:**

|  |  |
| --- | --- |
| **Description** | In searching algorithms, we will have our element searched in the whole array through the array and it iterates every element where find it will break the loop. |
| **Pseudo Code** | Search(n)  for i=1 to n  if find[i]==n  end if  end for |
| **Code in Python** | def Search(arr,x):  for i in range(len(arr)):  if arr[i]==x:  return i  return -1 |
| **Time Complexity Analysis** | The time complexity is **O(1).** |
| **Proof of Correctness** | Simply it will iterates through the loop and we will have our index at which element is located. |
| **Dry Run** | **Not Found Case:**  **[5,4,2,1,3,1]**  Enter the Element to find: 6  Not Found and i= -1  **Found Case:**  **[5,4,2,1,3,1]**  Enter the Element to find: 5  Element Found at 0 |

**Binary Search:**

|  |  |
| --- | --- |
| **Description** | In the case the element to be found is searched recursively by dividing the array into the single element and it is the element that needs to be found. |
| **Pseudo Code** | Procedure binary\_search  A is sorted array  n is size of array  x is value to be searched  set lowerbound=1  set upperbound=n  while x not found  if upperbound<lowerbound  EXIT: x doesnot exist  set midpoint = lowerbound+(upperbound-lowerbound)//2  if A[midpoint]<x  set lowerbound = midpoint+1  if A[midpoint]>x  set upperbound = midpoint-1  if A[midpoint] = x  EXIT: x found at location midpoint  End while  End procedure |
| **Code in Python** | def binary\_search(arr,low,high,x):  if high >= low:  mid = (high +low)//2  if arr[mid]==x:  return mid  if arr[mid]>x:  return binary\_search(arr,low,mid-1,x)  if arr[mid]<x:  return binary\_search(arr,mid+1,high,x)  else:  return -1 |
| **Time Complexity Analysis** | The time complexity of the algorithm will be **O(log n)** |
| **Proof of Correctness** | It is true for all the values because it just follows the divide and conquer rule at the simplest level.  We will have the proof of correctness of algorithm by defining the algorithm for array with size 1. It return the sorted array and afterwards we will our array with the size n and it will perform **n2**steps and will return our sorted array then we will increase the size by n+1 the array will have the sorted array with **(n+1)2** operations then we will say that our algorithm is correct. We have proven it by induction process.  Simply it will iterates through the loop and we will have our index at which element is located. |
| **Dry Run** | **Not Found Case:**  **[66, 1, 7, 1, -9, -27]**  Enter the Element to find: 4  Not Found and i= -1  **Found Case:**  **[66, 1, 7, 1, -9, -27]**  Enter the Element to find: 66  Element Found at 0 |

**Use Cases:**

**Use Case #1 (Main Window Design)**

|  |  |
| --- | --- |
| **Description** | The first one is our main window which will be displayed to user when he will open the application and will found as the main GUI page to implement its features. |
| **GUI Feature/Features** | This includes the following Features:   * A start button to start scrapping * A stop button to stop scrapping * A resume button to resume scrapping * A progress bar * A Menu for Sorting, help, about and providing the filters |
| **Screen Shot** | FIG: MAIN WINDOW OF GUI |

**Use Case #2 (Sorting Menu)**

|  |  |
| --- | --- |
| **Description** | The sorting menu provides the sorting algorithms in order to sort the values displayed in the table. On click of the menu it will ask to select the column to sort the selected columns based on the attributed scrapped. |
| **GUI Feature/Features** | This includes the following Features:   * Quick Sort * Insertion Sort * Selection Sort * Bubble Sort * Heap Sort * Merge Sort * Hybridized Algorithm |
| **Screen Shot** | FIG: SORTING MENU    FIG: COLUMN SELECTION MENU |

**Use Case #3 (Filters Menu)**

|  |  |
| --- | --- |
| **Description** | The filters menu provides the filters in order to sort the values displayed in the table. On click of the menu it will ask the keywords to filter |
| **GUI Feature/Features** | This includes the following Features:   * Contains * Less than * Greater than * Greater than equal to * Is Null * Is not Null * And * Or |
| **Screen Shot** | FIG: FILTERS MENU    FIG: INPUT FOR FILTERS MENU |

**Use Case #1 (Execution of Algorithms)**

|  |  |
| --- | --- |
| **Description** | After selecting the algorithm, the user will select the column and execute the algorithm on column. |
| **GUI Feature/Features** | The GUI will show a message box that the sorting has started for the selected column and will show the total execution time for different algorithms. |
| **Screen Shot** | FIG: FILTERS MENU    FIG: TIME OF EXECUTION OF QUICK SORT    FIG: TIME OF EXECUTION OF MERGE SORT    FIG: TIME OF EXECUTION OF HEAP SORT    FIG: TIME OF EXECUTION OF HYBIRDIZED ALGORITHM    FIG: TIME OF EXECUTION OF SELECTION SORT |

**Use Case #4 (Start and Resume Button)**

|  |  |
| --- | --- |
| **Description** | On clicking the start button the GUI will respond and will have our scrapping started and it will append the csv file as data scrapped in the file. |
| **GUI Feature/Features** | It will display the message box that will let the user know that the scrapping has started. And will show the progress through a progress bar. |
| **Screen Shot** | FIG: DISPLAYS SCRAPPING HAS STARTED    FIG: DISPLAYS THE CONTENTS IN TABLE |

**Use Case #5 (Progress Bar)**

|  |  |
| --- | --- |
| **Description** | On clicking the start button the GUI will respond and will have our scrapping started and it will append the csv file as data scrapped in the file. The progress bar will be displayed |
| **GUI Feature/Features** | It will display the message box that will let the user know that the scrapping has started. And will show the progress through a progress bar. |
| **Screen Shot** | FIG: DISPLAYS PROGRESS BAR |

**User Manual:**

**Name:**

The application name is **COURSE SCRAPPING APPLICATION**.

**Intended Use:**

The basic purpose of the application is that it scrap the data of online courses from the internet. The data is shown in the Graphical User interface(GUI) to the user (Fig2). The user can start, stop, resume this process. The attribute that will show in the graphical interface are the:

* Course Title
* University Name (University offering that course)
* Course Type (Category of Course)
* Course Ratings
* Number of Students (Enrolled in course)
* Course Level (Difficulty Level)
* Student Reviews

**Features:**

* **Functionality:**

The main function of this application is discussed earlier in the above topic. The other functionality of the application contains the Sorting and Filtering of the data shown in the table in GUI.

1. **Sorting:**

The function of the drop down of Sorting(Fig2) in the Menu bar is that it will sort data according to the selected algorithm on the particular column of the data. The selected column will be sorted according to the working of that algorithm. The pop-up will show when the sorting will started. The time taken by the machine for the sorting will be shown after the sorting has been completed.

Algorithms:

* Merge Sort
* Insertion Sort
* Selection Sort
* Hybridized Algorithm
* Bubble Sort
* Heap Sort
* Quick Sort

The above-mentioned algorithms are used in the sorting process

II. **Filtering:**

The process of Filtering in the application is that it will filter the data according the selected filter. The filter are:

* AND
* OR
* Is Equal to
* Is not Equal to
* Is less than
* Is Greater than
* Is Greater than equal to
* Is Null
* Is not Null
* Contains
* Starts with
* Ends With
* **Table:**

The table contains the data that will be scraped from the internet. The data will be according to the data scraped from the internet. After that, user can apply the sorting and filtering according to the need. The sorting and filtering process is described above.

* **Buttons:**

The buttons are placed in the footer of the app that will offer user to perform the actions to the scrapping method. The buttons used are as follow:

* **Start:**

The Start button will start the process of scrapping and will load the data in the table. When the button will pressed the pop-up will show that the scrapping has started. After the complete data is scraped in the table the pop-up will show that the data has been scrapped.

* **Stop:**

The stop button will stop the scrapping process and show the data in the table scrapped up to that action.

* **Resume**:

The resume button will resume the scrapping process. The scrapping will be resumed from the point where the scrapping has stopped.

* **Progress Bar:**

The progress bar will show the progress of the completed process how much scrapping has been done till now.

**Integration:**

For the sake of integration, we have used the pyqt5 and pyqt6 libraries to link our front end with the back-end python program through which we have done integrated our algorithms with the massive number of inputs. We have developed a clear understanding about the analysis and time execution of algorithms by implementing the given code.