**Group Leader**

2020-CS-53

**Proposer Details**

| Group Number | *G46* |
| --- | --- |
| Registration Number of Group Members | 1. 2020-CS-53 2. 2020-CS-79 |

**Proposal Details**

|  |  |
| --- | --- |
| ***Project*** |  |
| Proposed Project Title | App Scrapper |
| Executive Summary | *As the project name suggested that we will scrap the data of app from different types of platforms. After scrapping of data, we will store it in a csv (excel file). After the completion of the store process, we will perform different algorithm for the sorting, searching purpose.*  *In this modern era most of the world population has some kind of smart phone. According to the survey the number of peoples own smart phone or feature smart phone is* ***7.10 billion,*** *making up* ***89.90 person*** *of the world population. And every people who has smart phone use some kind of apps. There are many kinds of apps but most commons are Games, social media, E-Learning apps and many more. Let discuss about some of these kinds of Apps:*   * *Games are the most common in all over the world. According to a survey more than* ***2.2 billion*** *peoples play games. People play games for different purposes like some of them just want to kill time, some peoples like games so they play games for fun and even some peoples earn money by playing games and taking part in national and international competitions. Every person has their own reasons but all of them play games.* * *The Second most common kind of apps are social media apps. In 2020 there are* ***3.80 billion*** *peoples that use social media apps but in the last year their number increase by 9 present and not 321 million new user use social media and the number of users grow more rapidly. People use social media for different purposes but most of them use this for entertainment and be aware of the current issues occurring all over the world.* * *The third most common used apps are the E-learning apps. By using these kinds of apps, we can learn many things and study form all over the world. And with the help of these kind of apps we can teach our children while play games like they learn while play games. That’s why these kinds of apps are most useful and all persons get benefit from this.*   *By making the App Scraper we are able to scrab the kinds of app data from very famous websites and store all of that data into files so that person search their desire app and he get the link of the website form which he can download that app and get benefit from that app.* |
| ***Business Case*** |  |
| Outline the business need for the project | *The web developer’s because it will help the internet user to find which kind of the app has the best performance.* |
| End user of the product | *General public* |
| Motivation for Project | *As in the Description, we try to tell how many people use and apps. So, in this project we are going to make an apps scarper that scrap app form different websites and save information in csv files. So, whenever user is unable to find the link of their desire app S/he search that app in this project and he get all the information about that app and get the link form which website S/he is able to download that app.* |
| State the level of impact expected should the project proceed and implications of not proceeding | *It will be a strategic level because we have to scrap the data and give it to some industry so that they can make a website about the information of the app that the public wants to install.*  *On the other hand, it will also help industry to decide that is there any other app like we want to make also exist in market or not. If exist then how that app work so that we can make a better app and beat our competitors.* |
| ***Technical Details*** |  |
| Name of Entity | *Apps* |
| Attributes of Entity  (Minimum seven attributes/rows can be increased) | |  |  |  | | --- | --- | --- | | *Name* | *Data Type* | *Description* | | Name | String | Name of the app | | Size | Integer | Size of the app | | Rating | Integer | Rating of the app | | Genre (Optional) | String | Genre (action, school ...etc.) of the app | | Downloads | Integer | How many people have downloaded this app | | Developer Name | String | Name of person who developed it | | Age Restriction | Integer | Age Restriction means specific people can use it like 3+ means the people who are above 3 years old | | URL | String | URL of the website | | Description | String | Overview of the app | |
| Sample of Scrapping Source | ***Name of the app in the highlighted text***    ***Name of developers in highlighted text***    ***Genre of the app in highlighted text***      ***Age restriction in highlighted text***    ***Downloads in highlighted text***    ***Reviews in highlighted text***    ***Size in highlighted text***    ***Description in highlighted text***    ***Path of each element***  *a[@href]* |
| GitHub Repository Link | [*https://github.com/SameedAdeel/CS261F21PID46*](https://github.com/SameedAdeel/CS261F21PID46) |
| Sorting Algorithms |  |
| |  |  | | --- | --- | | **Algorithm Name** | **Description (Each algorithm in 2-3 lines)** | | Insertion Sort | Iterative algorithm | | Merge Sort | Recursive algorithm | | Selection Sort | Iterative algorithm | | Bubble Sort | Iterative algorithm | | Brute force | Iterative algorithm | | k-Select | Recursive algorithm | | Quick Sort | Recursive algorithm | | Counting Sort | Iterative algorithm | | Bucket Sort | Iterative algorithm | |  |  | |  |  | |  |  | |  |  | | |
| Searching Algorithms | *Sequential search* |
| Searching Filters for each data type | *Sequential search* |
| Multi-Level Sorting | Yes, in multilevel sorting we take 2 columns as an input and search of filter according to that we also use “AND” and “OR” operators between these two input columns so that we can filter the result more accurately. |
| Any other features | No |
| ***Interfaces for your project*** |  |
| ***U1 Login Page***  ***U2 Status***  ***U3 Check Data***   |  |  |  | | --- | --- | --- | | UI Component Name | Type of UI component | Purpose of UI Component/Other details | | Login Page(U1) | 2 Text Box and One button | The purpose of this components is that we will have a login page before the scrapping of the data | | Status (U2) | 1 Text Box 3 buttons and one progress bar | The purpose of the component is to take the url and scrap data and also have feature to stop the data and check the data button | | Check Data (U3) | 1 Table , 1 button 1 drop menu 1 Text box | All the data will be shown in this component and the sorting and search will be also done | |  |  |  | |  |  |  | |  |  |  | | |

|  |  |
| --- | --- |
| **Task Division** | **Collaboration** |
| Project Proposal | 50 % Group Member 1  50 % Group Member 2 |
| Scrapping | 70 % Group Member 1  30 % Group Member 2 |
| GUI implementation | 50 % Group Member 1  50 % Group Member 2 (Pencil tool) |
| Integration | 50 % Group Member 1  50 % Group Member 2 |
| Final Report | 50 % Group Member 1  50 % Group Member 2 |

**Planning Details:**

**Functionality of .py files**

|  |  |
| --- | --- |
| **Py file Name** | **Functionality** |
| main.py | In this file we import a login.py function login |
| login.py | It is the first UI when main.py load the login function |
| scrapping.py | It the file in which user take the url from google play store and scrap the data in  a xlxs or csv file. Here googlescrapper.py function is linked |
| googlescrapper.py | In this file a function which has a scrapper is made. The function is linked to the scrapping.py by using import googlescrapper |
| checkdata.py | Display the table in which information is loaded. Sorting Algorithm is implemented by using lib.py in which all sorting functions are available |
| lib.py | All the sorting algorithm are present in the lib.py file |

**Working of file using diagram**

**Call**

**Call if use of sorting algorithm**

**Call if checkdatabutton is pressed**

**Call if start button is pressed**

**Call**

|  |  |
| --- | --- |
| **Sorting Algorithm** | **Presoude Code** |
| Insertions Sort | For a =2 to n  Key=A[a]  b=a-1  while b > A[b] > Key  A[b+1]=A[b]  b--  A[b+1] =key |
| Selection Sort | for a =1 to A.length  min=a  for j =a+1 to A.length  if A [j] < A [min]:  min=j  if min!= a :  Swap A[a],A[min] |
| Merge Sort | MergeSort (A ,low ,high )  if low < high:  m= ((low+high-1)/2)  MergeSort(A , low ,mid)  MergeSort(A ,mid+1 ,high)  Merge(A, low ,mid ,high)  Merge ( A , low , mid , high)  Create two array left and right and store the data  compare left and right element which one is small add in a new array after this done store data should be given to A. |
| Counting Sort | Count(arr )  k= range of array  output=same arr with size of arr  count = array with size k+1  for I =1 to size  j=arr[I]  count[j]+=1  for i=1 to k:  count[i]+=count[i-1]  for I = size down to 0  j = arr[I]  count[j]-=1  output[count[j]] =arr[I]  First make second array that store how much time the number occur in the original array then take sum of that array and store it. After that run reverse loop and check the place of element in second array and according to that sort it in original array. After that element is sort then decrease the value of second array by one. |
| Quick Sort | 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)  Again, in this first find pivot point and gather all the numbers less then pivot point into the left side of array and gather all the greater number into the right side of array and again call quick sort on both of these ends until the array is fully sort. |
| Bucket Sort | Bucket (arr, n):  bucket= [[] for f in range (n)]  output= []  for I =1 to n:  bucket [(floor (n\*arr[I]))]. append(arr[I])  for I in range (0, Len (bucket)):  Insertion Sort (bucket[I])  for each in bucket:  output += each  return output  This Kind of sorting is specially use for floating points. If the number is not floating point, then make it. After that make 10 buckets from 0 to 9. Multiply each number with 10 and place it in the bucket if the bucket contains more than one number then sort that bucket with insertion sort and merge all the buckets. |
| Bubble Sort | Bubble Sort (A)  For a =1 to A.length  For j = 1 to A.length  if ( A[j] > A[j+1] )  swap(A[j],A[j+1])  In bubble sort only swapping procedure is to be done for the sorting purpose. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sorting Algorithm** | **Best Case**  **Time Complexity** | **Worst Case**  **Time Complexity** | **Average Case**  **Time Complexity** | **Strength of the sorting algorithm** | **Weaknesses** |
| Insertions Sort | O(n) | O(n^2) | O(n^2) | **1.** Stable algorithm means first element of duplicate come first the second duplicate element.  **2.** Perform well in small data set.  **3.** Space require is minimal. | **1.** Does not deal well for the huge list  **2.** Does not perform well as other algorithm  **3.** Useful for only small list |
| Selection Sort | O(n^2) | O(n^2) | O(n^2) | **1.** Arrange of data does not matter  **2.** Perform well in small data set  **3.** | **1.** Does not deal well for the huge list  **2.** Scanning of the whole array  **3.** unstable algorithm |
| Merge Sort | O(nlogn) | O(nlogn) | O(nlogn) | **1.**Stable algorithm means first element of duplicate come first the second duplicate element  **2.** Perform well for larger data set  **3.** Use Divide and Conqueror method | **1.** Goes through whole process if the list is sorted  **2.** Usage of memory  **3.** Slow for small data set |
| Bubble Sort | O(n) | O(n^2) | O(n^2) | **1.** In bubble sort perform only one task that is swapping because of this simple algorithm for the computer  **2.**Require less memory other than the list  **3.** | **1.** Not stable means first duplicate element will not come first  **2.**Does not deal with high numbers of list  **3.** Not to be useful for coding |
| Bucket | O(n+k) | O(n^2) | O(n+k) | **1.** Useful for floating numbers  **2.** When the elements are in bucket then every bucket works with itself and does not deal with other buckets  **3.** Stable algorithm | **1.** Performs depend on number of bucket  **2.** Cannot apply for all data type  **3.** Non Uniformly data required more time and more memory |
| Quick Sort | O(nlogn) | O(n^2) | O(nlogn) | **1.** Divide and Conquer method  **2.** In place algorithm  **3.** No extra storage required | **1.** Highly depends on pivot  **2.** In worst case time complexity equals to n^2  **3.**Mistake in implementation leads to the badly result |
| Counting Sort | O(n+k) | O(n+k) | O(n+k) | **1.** Fast algorithm  **2.** Stable algorithm  **3.** Can deal with the negative number | **1.** Not good for larger value i.e 232334  **2.** Not good for strings  **3.** |

|  |  |  |
| --- | --- | --- |
| **Sorting Algorithm** | **Input** | **Output** |
| Insertion Sort | Arr = [6,5,4,3,2,1] | Result after using Insertion Sort = [1,2,3,4,5,6] |
| Selection Sort | Arr= [6,5,4,3,2,1] | Result after using Insertion Sort = [1,2,3,4,5,6] |
| Merge Sort | Arr= [6,5,4,3,2,1] | Result after using Insertion Sort = [1,2,3,4,5,6] |
| Bubble Sort | Arr= [6,5,4,3,2,1] | Result after using Insertion Sort = [1,2,3,4,5,6] |
| Bucket Sort |  |  |
| Quick Sort | Arr= [6,5,4,3,2,1] | Result after using Insertion Sort = [1,2,3,4,5,6] |
| Count Sort |  |  |

|  |  |
| --- | --- |
| **Sorting Algorithm** | **Proof of Correctness using loop invariant** |
| Insertion Sort | **Initialize:**  Take a certain key  **Maintain:**  Using the i loop and comparing key with A[i-1] for swapping the values in order to have a sorted array  **Terminate:**  When i reaches A.length it means array A is sorted |
| Selection Sort | **Initialize:**  Inner loop begin with j=i+1 and min=i means A[i] is smallest element in the A [i…i+1]  **Maintain:**  After the begin of j loop A[j] compare itself with the A[min] that it is less then the A[min] then value of min=j and j++ after the j loop end then values are swapped  **Terminate:**  When i reaches A.length it means array A is sorted |
| Merge Sort | **Initialize:**  Divide the array into 2 parts i.e. left and right using midpoint left array will consist element from 0 to m while right will consist from m+1 till A.length  **Maintain:**  After division if L[i] <= R[j] then store the value of L[i] in new array tem[k] if not then store value of R[j] in tem[k]  **Terminate:**  After all procedure is done array is sorted |
| Bubble Sort | **Initialize:**  Both loops starts with 0 index  **Maintain:**  In inner loop each element is compare with all the elements in that array with condition A[j] > A[j+1] swap both it means j+1 is small the j  **Terminate:**  After all procedure is done array is sorted |
| Bucket Sort | **Initialize:**  Make the buckets according to size of the array to be sorted  **Maintain:**  Multiply each element by size so that each element in the right place after this insertion sort is called so sorting the buckets individually  **Terminate:**  After all procedure is done array is sorted |
| Quick Sort | **Initialize:**  Make a pivot using partition  **Maintain:**  Recall the quicksort with 0 to pivot -1 and pivot+1 to A.length.  **Terminate:**  After all procedure is done array is sorted |
| Count Sort | **Initialize:**  Range, count which will is range+1,output of same size of input  **Maintain:**  Count which value comes how many time in A using for loop after this loops start from the A. length in which output[count[A[i]] mean output [which index] = A[i].  **Terminate:**  After all procedure is done array is sorted |

|  |  |
| --- | --- |
| **Sorting Algorithm** | **Code in python** |
| Insertion Sort | def InsertionSort (Arr):      for i in range (1, len (Arr)):          key=Arr[i]          j=i-1          while j>=0 and Arr[j] > key:              Arr[j+1] =Arr[j]              j=j-1          Arr[j+1] = key |
| Selection Sort | def SelectionSort (Arr):      #min=0      for i in range (len (Arr)):          min=i          for j in range (i+1, len(Arr)):              if Arr[j] < Arr[min]:                  min=j          if min! = i:              Arr[i], Arr[min]=Arr[min], Arr[i] |
| Merge Sort | def Merge (A, low, m, high):      L= []      R= []      o=m+1      for i in range (low, m+1):          L.append(A[i])        for j in range (o, high+1):          R. append(A[j])        i=0      j=0      x= []      while i< len(L) and j < len(R):          if(L[i]<R[j]):              x.append(L[i])                i=i+1          else:              x.append(R[j])              j=j+1      if i! = len(L):          while i<len(L):              x.append(L[i])              i=i+1      if j! = len(R):          while j<len(R):              x.append(R[j])              j=j+1      k=low      for each in x:          A[k]=each          k=k+1  def MergeSort (arr, l, h):      if l < h:          m=int((l+h-1)/2)          MergeSort (arr, l, m)          MergeSort (arr, m+1, h)          Merge (arr, l, m, h) |
| Bubble Sort | def BubbleSort(arr):      count=len(arr)      swapped=False      for i in range(0,count-1):          swapped=False          for j in range (0, count-1):              if arr[j]> arr[j+1]:                  arr[j], arr[j+1] =arr[j+1], arr[j]                  swapped=True          if (not swapped):              break |
| Bucket Sort | def Bucket (arr, n):      bucket= [[] for f in range (n)]      output= []      for i in range (0, n-1):          bucket[(floor(n\*arr[i]))]. Append (arr[i])        for i in range (0, len(bucket)):          InsertionSort(bucket[i])      for each in bucket:          output += each      return output |
| Quick Sort | 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  def Quicksort (arr, low, high):      if low < high:          pi= partition (arr, low, high)          QuickSort (arr, low, pi-1)          QuickSort (arr, pi+1, high) |
| Counting Sort | def ConuntingSort(arr):      k= max(arr)-min(arr)      size=len(arr)      count= [0] \* (k+1)      output= [0] \* size      index= -1      P= []      N= []      for i in range (size):          j= arr[i]          count[j]+= 1        for i in range (1, k+1):          count[i]+=count[i-1]        for i in range (size-1, -1, -1):          j = arr[i]          count[j]-=1          output[count[j]] = arr[i]      for i in range (0, size):          if output[i] >= 0:              P. append (output[i])          else:              N. append (output[i])      return N+P |

|  |  |
| --- | --- |
| Difficulties | Explanation |
| Data Scrapping | Data was coming put in the saving procedure data got duplicate. |
| Missing Attribute in some pages | Face the issue in the scrapping that when one element is missing it gives some error. Resolve this issue using try except statement |
| Data Loading | Whole Data is not loading from the csv file |
|  |  |
|  |  |
|  |  |