**PROJECT NAME: LAPTOPSTERS**

**MEMBERS: 2020-CS-71**

**2020 –CS-84**

Submitted to: Mr. Samyan Qayum Wahla

Date: 3/12/2021

Names of member: Esha Tur Razia

Sana Shafique



**Department of Computer Science**

**University of Engineering and Technology Lahore**

|  |
| --- |
|  |
|  |

**Table of Contents**

[1. Summary of Project 3](#_Toc164161353)

[2. Learning Outcomes 4](#_Toc164161354)

[3. Data Scrapping overview](#_Toc164161355) 5

[4. Problems Faced during Scrapping 6](#_Toc164161356)

[5. User interface 7](#_Toc164161357)

[6. UI description 8](#_Toc164161358)

[7. Quick Sort 9](#_Toc164161359)

[8. Merge Sort 12](#_Toc164161360)

[9. Bubble Sort 15](#_Toc164161361)

[10. Insertion Sort 18](#_Toc164161362)

[11. Selection Sort 21](#_Toc164161363)

[12. Counting Sort 24](#_Toc164161364)

[13. Linear Search 27](#_Toc164161365)

[14. Integration 30](#_Toc164161363)

[15. Planning of Project 31](#_Toc164161364)

[16. Collaboration 31](#_Toc164161365)

1 **Summary of the Project**

**1.1 Description**

We will scrap the data of a million laptops from amazon and save it in a comma separated values file and then load it. It will contain seven entities. We will apply searching and sorting algorithms on it. The user will be facilitated to sort the data according to their convenience either they want to sort by name or by dates etc. User can also search laptops from a particular category.

**1.2 Link to Git Hub Repository**

<https://github.com/Esha-tur-Razia/CS261F21PID41.git>

**1.3 Business benefits for project**

It will attract a vast majority of the audience who are either working in office or are developers. Students of any domain also need laptops for their studies. Gaming laptops are also wildly in demand. Laptop business is flourishing vastly in this era.

**1.4 Table listing Entities required for the project**

|  |  |  |
| --- | --- | --- |
| Name | **Data Type** | **Description** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| Name | String | Contains name of laptops |
| Price | Double | Display their international price |
| Display | Double | How many inch will be its screen |
| Core | String | Series of the laptops |
| Rating | Integer | How much audience likes it |
| Memory | Integer | How much space it can store |
| Gpu | Double | Processor for computer screen |

Table 1.4 explaining the entities required for the project

**2 Learning Outcomes**

**2.1 Algorithms**

* Help us to design and analyse simple algorithms
* Implement basic searching and sorting algorithms
* To apply algorithms in real life problems
* Implementation of sorting algorithms like
* Quick-sort
* Merge-sort
* Insertion-sort
* Bubble-sort
* Counting sort

**2.2 Industrial Benefits**

* It benefits in keeping the good quality
* By having a deep understanding of the algorithms and their analysis we can apply more efficient algorithm on given set of inputs thus increasing the quality of the product
* A given set of data can be sorted easily

**2.3 Motivation for sorting**

* Sorting algorithms are used for optimization
* Programs speed can be increased
* Least amount of memory is consumed

**2.4 Our Motivation for the Project**

* The project will enable the laptop agencies to sort their data through multiple algorithms
* Scrapping the products will also benefit these agencies

**3 Data scrapping overview**

**3.1 Basic idea**

The extraction of data from a website is referred to as web scraping. It can be then saved into spread sheet or any other form for the convenience of the user. This task can be performed manually but it takes a lot of time. While scrapping the data saves a lot of time. Web scrapping cannot always be illegal the issues faced due to scrapping are discussed in the later section.

**3.2 Working**

* First we give URL of the page to be scraped
* It loads the whole HTML contents of the page
* We move our cursor to our required details and copy the class name of the desired products
* Then we save the data into our spread sheet files

**3.3 Libraries used**

* Request library
* Selenium library
* Beautiful Soup
* LXML library
* Pandas library

**3.4 Purpose of using the library**

* To get HTML files request library allows us to get it by sending HTTP requests for getting HTML files
* Beautiful Soup is required for the parsing of the HTML files it is very simple to use and gathers the data efficiently
* We can use pandas library for saving data into CSV file

**3.5 Some function details**

We can use *findall()* function to get all contents at once and *prettify()* to get all contents in readable format.

**4 Problems faced**

**4.1 Websites from which we scraped**

Initially, we decided to scrape the data from Amazon website <https://www.amazon.com/> but we cannot access the whole data of the laptops. So we decided to scrap data from multiple sites. We scraped the data from the following sites

* <https://www.flipkart.com/>
* <https://www.amazon.com/>
* <https://www.daraz.pk/>
* <https://myshop.pk/?SID=b017566cb133d6e7e491940f431b57d8>
* <https://www.made-in-china.com/?gclid=Cj0KCQjww4OMBhCUARIsAILndv4-oIMrvAkLEh5HBGejKuBYU8RMNFe7EMzwVIakwvuZszp9ko5fFs4aAtS6EALw_wcB>

**4.2 Common issues**

* Before scraping we should check if website allows scraping or not by using Robots.txt if owner agree they will allow you to scrape
* Sometimes IP blocking occurs if many scrapers are scraping their data from a website it blocks their IP address.
* CAPTCHA occurs to ensure that some human is scraping not a robot
* Most common issue during scraping a large amount of data is that the website sometimes stops scraping and often fails to load the data
* Unfortunately much data can’t be scrapped

These are some issues we faced during scraping that were removed by taking some advice from the instructors and fellows.

**5 User interface of our application**

**5.1 Pencil Tool**

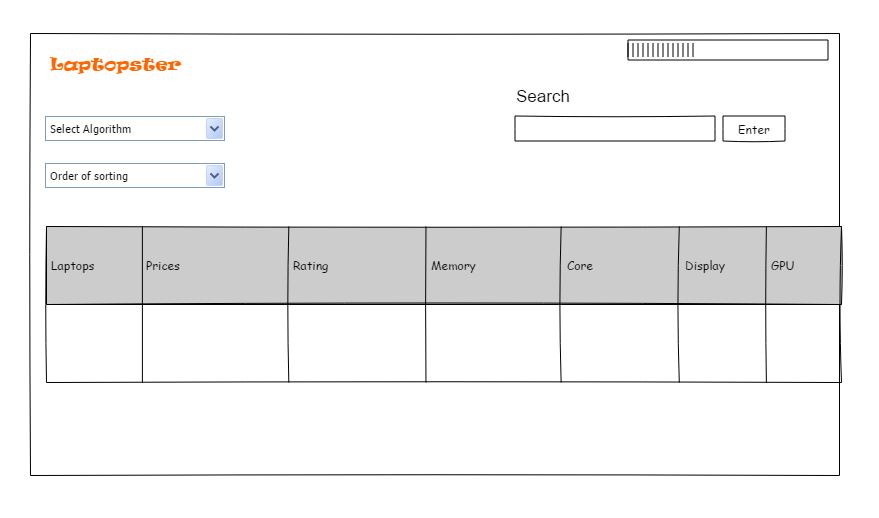
****

Image 5.1 showing the pencil tool illustration of the project

**5.2 Actual GUI**

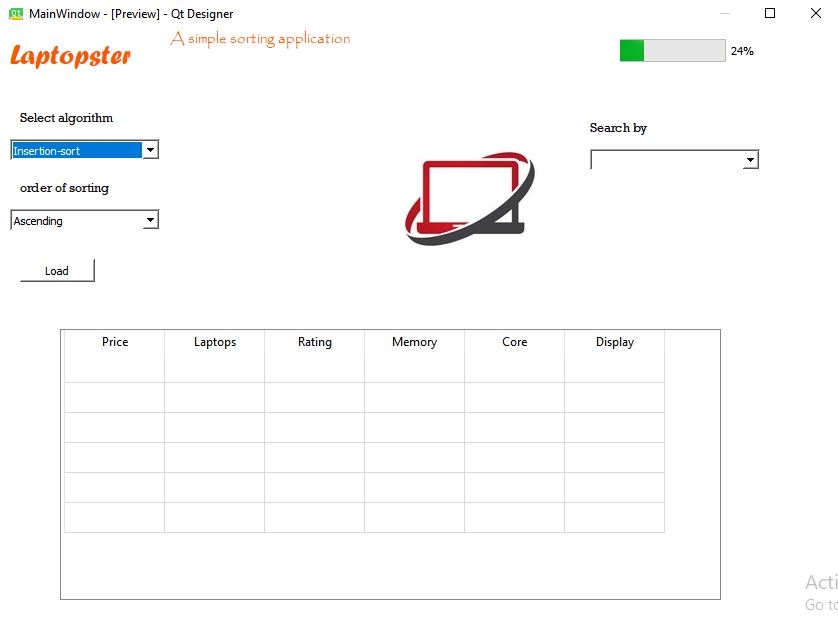
****

Image 5.2 showing the Implemented GUI using pyqt5

**6 Table Explaining UI Description**

|  |  |  |
| --- | --- | --- |
| UI Component Name | **Type of UI component** | **Purpose of UI Component/Other details** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| Laptopster | A label | To specify the application name | |
| Grid | A grid | Used to display the entities from the csv file | |
| Search | A text area | To search a specific product | |
| Enter | A button | To perform search and display result | |
| Select\_algo | A combo box | To select a desired algorithm | |
| Order\_sort | A combo box | To select the order for sorting | |
| P\_bar | A progress bar | To display the progress | |

Table 6 explaining the UI description of the project

|  |  |
| --- | --- |
| Quick Sort | |
| |  | | --- | | Pseudo code |   quickSort(list, small, large)  if (small < large)  p = partition(list, small, large)  quickSort(list, small, p - 1);  /\* prior p \*/  quickSort(list, p + 1, large);  /\* after the p \*/  partition (list, small, large)  pivotPoint = list[large]  i = (small - 1)  for (j = small; j <= large- 1; j++)  if (list[j] < pivotPoint)  i++  swap list[i] and list[j]  swap list[i + 1] and list[large]  return (i + 1) | |
| Time complexity | Big-O(n log(n)) |
| |  | | --- | | Description |   In this, we randomly select a pivot point and array is divided on the basis of that pivot and then sort it. It works on the principle of divide and conquer. | |
|  | |
| Strength | |
| * Its time complexity is big-o(n log(n)) * Its faster than merge sort * Additional memory is not required | |
| Weakness | |
| * Not stable * Depending on number of elements of array running time differs * If array is sorted running time might degrades | |
| Proof of correctness | |
| Picking a pivot value randomly can help to avoid worst case scenarios like (perfectly sorted array). | |

|  |
| --- |
| Code in python |
| def quickSort(arr,small,large)  if len(arr) == 1  return arr  if low < high  p = partition(arr, small, large)  quickSort(arr, small, p - 1)  quickSort(arr, p + 1, large)  else  p = 0  def partition(arr, small, large)  i = (small - 1)  pivot\_point = arr[large]  for j in range(small,large)  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)  if \_\_name\_\_ == '\_\_main\_\_':  arr = [4, 1, 3, 9, 7, 8]  n = len(arr)  quickSort(arr, 0, n-1)  print("Sorted array is:")  for i in range(n):  print("%d" arr[i])n"), |

|  |  |  |
| --- | --- | --- |
| Merge Sort | | |
| |  | | --- | | Pseudo code |   def mergeSort( arr )  if ( n == 1 )  return arr  array1 = arr[0] to arr[n/2]  array2 = arr[n/2+1] to arr[n]  array1 = mergeSort( array1 )  array2 = mergeSort( array2 )  return merging( array1, array2 )  def merging( arr\_a, arr\_b )  arr\_c  while ( arr\_a and arr\_b contains members )  if ( arr\_a[0] > arr\_b[0] )  adding arr\_b[0] at the ending of arr\_c  then remove arr\_b[0] from arr\_b  else  adding arr\_a[0] at the ending of arr\_c  then remove arr\_a[0] from arr\_a  while ( arr\_a contains members )  adding arr\_a[0] at the ending of arr\_c  then remove arr\_a[0] from arr\_a  while ( arr\_b contains members )  adding arr\_b[0] at the ending of arr\_c  then remove arr\_b[0] from arr\_b  return arr\_c | | |
| Time complexity | Big-O(n log(n)) | |
| |  | | --- | | Description |   In this we simply split the given array into two parts and then sort them recursively until they are sorted and then merge them at the end. | | |
|  | | |
| Strength | |
| * Merge sort is much faster than straight forward methods of sorting * Stable algorithm * It’s a recursive algorithm that calls itself over and over | |
| Weakness | |
| * Sub elements use more space * Slower than others | |
| Proof of correctness | |
| It requires big-o(n) space on arrays and linked lists, it demands a constant space. | |

|  |
| --- |
| Code in python |
| def Merge\_Sort(arr):  first = []  second = []  if len(arr) > 1:  mid = len(arr)//2  first = arr[:mid]  second = arr[mid:]  Merge\_Sort(first)  Merge\_Sort(second)  i = 0  j = 0  k = 0  while i < len(first) and j < len(second):  if first[i] < second[j]:  arr[k] = first[i]  i = i + 1  else:  arr[k] = second[j]  j = j + 1  k = k + 1      while i < len(first):  arr[k] = first[i]  i = i + 1  k = k + 1    while j < len(second):  arr[k] = second[j]  j = j + 1  k = k + 1  def print\_List(arr):  for i in range(len(arr)):  print(arr[i], end=" ")        if \_\_name\_\_ == '\_\_main\_\_':  arr = [4, 1, 3, 9, 7]  print("Given array is", end="\n")  print\_List(arr)  Merge\_Sort(arr)  print("Sorted array is: ", end="\n")  print\_List(arr) |

|  |  |
| --- | --- |
| Bubble Sort | |
| |  | | --- | | Pseudo code |   def BubbleSort( arr )  looping = arr.count;  for i = 0 to looping-1  do:  swap = false  for j = 0 to looping-1  do:  /\* check elements which are adjacent to each other and compare them\*/  if arr[j] > arr[j+1] then  swap( arr[j], arr[j+1] )  swap = true  if(! swapped)  break  return list | |
| Time complexity | Big-O(n) |
| |  | | --- | | Description |   In this, we simply compares the consecutive elements from the given array and then swap them until they are sorted. | |
|  | |
| Strength | |
| * Use less space * Few lines of code | |
| Weakness | |
| * Complexity is big-o(n^2) * Take much time for sorting | |
| Proof of correctness | |
| As it takes a large time, it is not a preferred sorting algorithm. | |

|  |
| --- |
| Code in python |
| def bubble\_Sort(arr):  for i in range(len(arr) - 1):  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]  return arr  if \_\_name\_\_ == '\_\_main\_\_':  arr = [5, 8, 9, 3, 5, 7, 1, 3, 4, 9, 3, 5, 1, 8, 4]  print("Given array: ")  bubble\_Sort(arr)  printArr(arr) |

|  |  |
| --- | --- |
| Insertion Sort | |
| |  | | --- | | Pseudo code |   for i = 1 to n  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 | |
| Time complexity | Big-O(n) |
| |  | | --- | | Description |   In this we split the input in two parts one is sorted and the other is unsorted part then take any element from the unsorted part and replace it at the desired position. | |
|  | |
| Strength | |
| * Efficient * Take less usage of space | |
| Weakness | |
| * It takes time despite array being sorted * Fails on huge sets of information | |
| Proof of correctness | |
| In the best case scenario (already sorted) every insert requires a constant time. | |

|  |
| --- |
| Code in python |
| def Insertion\_sort(arr):  # for loop iteration in python from j to the length of array  j = 1  for j in range(len(arr)):  key = arr[j]  i = j-1  while i > 0 and arr[i] < key:  arr[i+1] = arr[i]  i = i-1  arr[i+1] = key    def print\_List(arr):  for i in range(len(arr)):  print(arr[i], end=" ")    arr = [5, 7, -8, 9, 10, 4, -7, 0, -12, 1, 6, 2, 3, -4, -15, 12]  print("Given array is", end="\n")  print\_List(arr)  Insertion\_sort(arr)  print("Sorted array is: ", end="\n")  print\_List(arr) |

|  |  |  |
| --- | --- | --- |
| Selection Sort | | |
| |  | | --- | | Pseudo code |   def selection (arr)  for i = 1 to n - 1  /\* current element will be converted to min\*/  min = i  /\* checking if the element is minimum \*/  for j = i+1 to n  if arr[j] < arr[min]  then min = j;  if index of min != i  then swapping arr[min] and arr[i] | | |
| Time complexity | Big-O(n^2) | |
| |  | | --- | | Description |   In this smallest element is selected from the initially given input and then it is swapped with the left most element and the process is repeated until the sorted array is formed. | | |
|  | | |
| Strength | |
| * Data arrangement is not affected * Fewer operations are used | |
| Weakness | |
| * Large amount of time is required | |
| Proof of correctness | |
| Even if the array is already sorted it requires time to scan the entire array. | |

|  |
| --- |
| Code in python |
| def Selection\_Sort(arr):  for i in range(len(arr) - 1):  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]  return arr  if \_\_name\_\_ == '\_\_main\_\_':  arr = [5, 8, 9, 3, 5, 7, 1, 3, 4, 9, 3, 5, 1, 8, 4]  print("Given array: ")  Selection\_Sort(arr)  printArr(arr) |

|  |  |
| --- | --- |
| Counting Sort | |
| |  | | --- | | Pseudo code |   def CountingSort(arr)  for i = 0 to length(arr) - 1  do  j = key(arr[i])  num[j] += 1  for i = 1 to k  do  num[i] += num[i - 1]  for i = length(arr) - 1 down till zero  do  j = key(arr[i])  num[j] -= 1  result[num[j]] = arr[i]  return result | |
| Time complexity | Big-O(n+k) |
| |  | | --- | | Description |   This works by counting presence of each element in given input. | |
| Strength | |
| * Less time complexity | |
| Weakness | |
| * For large range of values it requires more space | |
| Proof of correctness | |
| Non negative key values have range of k. | |

|  |
| --- |
| Code in python |
| import random,math  def getKey(n):  return n  def countingSort(list,k,getKey):  count = [0]\*k  for n in list  count[getKey(n)] = count[getKey(n)] + 1  for i in range(k)  if i ==0:  count[i] = count[i]  else:  count[i] += count[i-1]  output = [None]\*len(list)  for i in range(len(list)-1, -1, -1)  sortkey = getKey(list[i])  output[count[sortkey]-1] = list[i]  count[sortkey] -=1  return output  random.seed(0)  arr = [random.randint(0,20) for n in range(10)]  print("Unsorted array")  print(arr)  print("\nSorted array using basic counting sort")  output = countingSort(arr, max(arr) +1, getKey)  print(output) |

|  |  |
| --- | --- |
| Linear Search | |
| |  | | --- | | Pseudo code |   def linearSearch(arr)  while x < arr.length  do  if(arr[x] == r]  return x  else  x = x + 1  return -1 | |
| Time complexity | Big-O(n) |
| |  | | --- | | Description |   It simply compares the element in a specific sequence until the element is found or matched. | |
|  | |
| Strength | |
| * Small and medium can be searched quickly * List does not required to be sorted * Not affected by deleting and inserting | |
| Weakness | |
| * Time consuming for large arrays * Linear search algorithm is worst case * Less efficient | |
| Proof of correctness | |
| * It searches the element by sequentially comparing the array elements. | |

|  |
| --- |
| Code in python |
| def linearSearch(arr, n, k):   for i in range(0, n):          if (arr[i] == k):              return i   return -1      arr = [1 ,3, 5, 4, 7, 9]  k = 7    n = len(arr)  ans = linear\_Search(arr, n, k)  if(ans == -1):      print("Element not found")  else:      print("Element found at index: ", ans) |

**Integration:**

**Libraries used:**

* QT core
* QT widgets
* QTGUI

**Purpose:**

* **QT core:**

Used for storing binary files.

* **QT widgets:**

Used for creating interfaces.

* **QT GUI:**

It provides class for image, font and text etc.

**Loading data into table**

In this, we loaded the data from csv file by linking it with button named load data .by clicking this all data from csv file moves to grid and progress bar shows the percentage of loaded data.

**Sorting Algorithms:**

After loading of data we apply sorting and searching algorithms which will sort all the data loaded in the grid.

**Problems faced:**

1. We face problems in loading data from csv file to grid. We added code of a button for loading data but the UI was running same so data was not being loaded.
2. Also instead of changing components in PY code the UI was running same i.e. no change occurs in UI.

**Planning**

|  |  |
| --- | --- |
| **Mile Stones** | **Completed Date** |
| Project Proposal | 14 October,2021 |
| Scraping | 19 October,2021 |
| UI implementation | 23 October,2021 |
| Sorting Algorithms | 27 October,2021 |
| Integration | 2 November,2021 |
| Project Report | 3 November,2021 |

**Collaboration**

Collaboration between the members occurred by

* Both members were from hostel so most of the communication relating the project occurred in person through physical presence
* During weekends or holidays communication regarding project occurred through online settings such as zoom and Microsoft teams
* Remaining communication occurred via whatsapp and through messages.