**Vehicle Bite (VB)**

CS261- Algorithm Detail Document



Project Supervisor

Mr. Samyan Qayyum Wahla

Group Member (G32)

|  |  |
| --- | --- |
| Nabeel Yousaf | 2020-CS-103 |
| SaifUllah | 2020-CS-102 |

Department of Computer Science

University of Engineering and Technology, Lahore, Pakistan

Table of Contents

[1 Project Details 4](#_Toc86972177)

[1.1 Project Title 4](#_Toc86972178)

[1.2 Project Statement 4](#_Toc86972179)

[1.3 Description 4](#_Toc86972180)

[1.4 Audience 5](#_Toc86972181)

[1.4.1 Problem Overview 5](#_Toc86972182)

[1.4.2 Possible Solutions 5](#_Toc86972183)

[1.4.3 How can we solve this using Computers 5](#_Toc86972184)

[1.4.4 Audience Motivation 5](#_Toc86972185)

[1.5 Requirements 5](#_Toc86972186)

[2 Technical Details 6](#_Toc86972187)

[2.1 Attributes 6](#_Toc86972188)

[2.2 Scrapping 7](#_Toc86972189)

[2.2.1 Why we use web scrapping? 7](#_Toc86972190)

[2.2.2 Websites 7](#_Toc86972191)

[3 Business Details 9](#_Toc86972192)

[3.1 Overview 9](#_Toc86972193)

[3.2 For Audience 9](#_Toc86972194)

[3.3 For Developers 9](#_Toc86972195)

[3.4 Motivation 9](#_Toc86972196)

[4 Project Planning 10](#_Toc86972197)

[4.1 Developer Details 10](#_Toc86972198)

[4.2 UI Sample 10](#_Toc86972199)

[4.3 Task Division 10](#_Toc86972200)

[4.4 Collaboration 10](#_Toc86972201)

[5 Algorithms 11](#_Toc86972202)

[6 Final Application 29](#_Toc86972203)

[6.1 UI Model 29](#_Toc86972204)

[6.1.1 Components 29](#_Toc86972205)

[6.2 Remaining 29](#_Toc86972206)

[6.3 Integration 29](#_Toc86972207)

# **Project Details**

## **Project Title**

After a detailed discussion, we titled this project with a combination of two words called **Vehicle Bite.**

## **Project Statement**

Today, we are having an enormous number of varieties in our daily life products like, in case of mobile phones and cars different brands are introducing different series of products depending upon their attributes. It is hard for a buyer to select the product and optimize it, for his need. Everyone wants to make his/her work smoother and easier. Everyone is trying to save more time. We are trying to solve their problem by getting data from different sites and make user friendly software.

This software is designed to help a customer in purchasing the automobile car that is suitable for him/her. User can easily search for a car that perfectly matches his/her desire.

## **Description**

After an observation of this problem, our team is designing software, named as **Vehicle Bite,** to solve this problem. The major approach to make this software is to save the time of every person who is willing to buy a vehicle. Many people visit different sites on internet or visit different places to buy a vehicle, which consumes a large amount of time. Now it’s our major goal to scrap data from different famous sites and then apply some sorting and searching algorithms and also we can check its time complexity.

Using this technique we will try to get millions of data on the basis of international criteria. After getting the data from website we use different algorithms to manipulate all data. User can arrange data in many orders like ascending or descending order. You can also check different types of data in different manners. User can also check the speed of sorting by implementing different algorithms on different types of data.

The data of the vehicles is too much large. So, it is difficult to enter data by single person or a couple of persons. It requires a couple weeks to enter data. On the basis of cost and time we decided to get this data from different websites. For this purpose, we are using a technique called Web-Scrapping.

Searching is also a feature of this software. User can search according to different criteria line country name, price, or model of vehicle and can choose desired vehicle. User can also check the time taken by search using different algorithms. The algorithms are of insertions sort, merge sort, selection sort and bubble sort. The user will select any one algorithm and data will be sorted according to that algorithm. Then after selecting algorithm user will enter option of run and the data will be show in the table with the attributes.

It will:

* Scrape data of multiple vehicles of different brands and series from many websites.
* Show the time taken by data scrapping including the stop and play functionality
* Provide an option to sort lists depending upon different attributes like engine power, passenger capacity, etc.
* Provide the functionality of searching a particular vehicle.
* Allows selection of algorithms for sorting and searching and accordingly shows time taken.

### **Problem Overview**

It’s an era of Internet and Communication Technologies. Everyone is buying products from different **Online Platforms.** It’s our target to approach those persons who are willing to buy a vehicle. If a person is buying a car, he/she will check some websites and then contact the seller. The 2nd approach is to visit physically that place where the people sell their vehicles. Time is much important and these solutions are time consuming. 1st one is contains time on visiting different and trustable websites and 2nd is physical visit which also more time consuming than 1st one. Now, our team is trying to find the maximum possible solutions and then we will implement in possible best solution.

### **Possible Solutions**

1. Visit different websites
2. Physical visit of market
3. Buy a vehicle from a relative
4. Get data from a single source

### **How can we solve this using Computers**

In our Possible Solutions, two of them (i.e. a. and d.) are solving this using computer. We are selecting 4th solution (i.e. d), in which we will get data from different websites, combine it and then load it in our software.

### **Audience Motivation**

Life became too fast that we are travelling from one place another in minutes and hours instead of days and months. In past, if we cover a distance in some days, now we cover same distance in hours. All of us want vehicles according to our daily routine. Some are using multiple vehicles for different purposes (i.e. for personal use, for business etc).

## **Requirements**

|  |  |
| --- | --- |
| Programming language | Python 3 |
| Version | 3.9.7 |
| IDE for Python 3 | There are many IDE’s of Python. Some of them are given below:   * PyCharm * Spyder * Jupyter(Anaconda) * VS Code   From all these IDEs, we choose VS Code for Python 3. |
| Library | * pandas * peautifulsoup * time * pyttsx3 * requests |

# Technical Details

## Attributes

|  |  |  |
| --- | --- | --- |
| **Name** | **Data Type** | **Description** |
| Title | String | In websites, the main content writtern for each entity is the title. It is the main content of the entity which has been written by seller. It contains the name and some other important information of the vehicle.  For example,    [In this entity , “Toyota Yaris 2021 ATIV X CVT 1.5 for Sale” is the title of this entity.](https://www.pakwheels.com/used-cars/toyota-yaris-2021-for-sale-in-lahore-4633963" \o "Toyota Yaris  2021 ATIV X CVT 1.5" \t "_blank) |
| City | String | This contains the information about city of the vehicle. This entity will help users to buy vehicle accrding to their desired city.  For Example, in previous picture, city is Lahore. |
| Model | String | It tells us the model of vehicle of a specified entity. In the picture(placed in tite), the model is “2021.” |
| Country | String | It contains the information of a car that where it is located. This entity will help users to buy vehicle accrding to their desired country.  For Example, in previous picture, country is Pakistan. |
| Engine Capacity | String | It is the maximum engine capacity of vehicle that has been fixed by company. It contain that perticular information.  For example,    In this picture, “1300cc” is the engine capacity. |
| Meter Reading | String | It contains the information about the vehicle that how much kilometers the vehicle has been drived. It helps the uer to buy a suitable and better vehicle.  For exmple, in previous picture, this car drived 13km. |
| Auto/Manuel | String | It contains the information about the vehicle is automatic or manuel. |
| Fuel Type | String | There are two types of fuel, one is petrol and second diesel. This attribute contains that pertucular information in which user can know which type of fuel he will use after buying this vehicle. |
| Identity | String | It is the name/identity that has been entered by the user. It maybe the special name of the seller. Sometimes, some sellers cannot enter this information so it maybe empty sometimes. |
| Price | String | It contains the information of the amout that the owner of the vehicle demands. You can deal with user and both of you can also decide a custom price after contacting to each other. |
| Contact | String | It contains a specific information through which we can contact with seller. |

## **Scrapping**

### **Why we use web scrapping?**

As mentioned earlier, it’s our target to collect maximum data as we can. It is too much time consuming to enter data of all these vehicles. So, we use web scrapping to get data from different websites and then load it to our page. In short, scrapping is used to collect large amount of data from different websites.

### **Websites**

|  |  |
| --- | --- |
| PakWheels | **PakWheels** is Pakistan’s website which contains data of vehicles.  **Front Display:**    **How entity looks:**    It contains all data that we required  **No. Of vehicles:** approximately 300k to 400k+ vehicles.  **Link:** https://www.pakwheels.com/ |
| Carvago | **Carvgo** is an international website which contains a large amount of data and provides proper details.  **Front display:**    **How entity looks:**    It contains all data that we required.  **No. Of vehicles:** approximately 400k to 600k+ vehicles.  **Link:** <https://www.carmax.com/> |

# **Business Details**

## **Overview**

This software is used generally for business point of view as one can sell or buy any type of vehicle from such types of store which are available online 24/7.It provides a large variety of cars from different models and of different year whether it’s new or used. This software opens a new gateway for online users as they can sell or buy new or old cars without visiting the store. It has merits as well as some demerits as one cannot see the physical appearance of vehicle. It helps the user to see any type of vehicle from all over the country and can see different features of specified vehicles.

## **For Audience**

This software solves a lot of problem of the one who wants to buy or sell their vehicles specially cars. Without this software the audience may find any customer and then has to show or see the specified car which is quite time taking task. It has saved the time of both buyer or seller as all the features are uploaded online with pictures of the specified car. The only work left is to go and deal with the dealer. This software might help both buyers and sellers as it can be modified easily so such cars which are already sold or purchased can be eliminated from the software.

## **For Developers**

By implementation of this software developers can be entertained as the main in this software is of developers as they have to upgrade features of specified car. Developers can be hired for adding or removing unwanted or irrelevant data in case if they are sold out of purchased. By the span of time data has to be changed as different features like distance covered etc. are changing with time. They can upgrade any additional options to the software in order to entertain their customers which helps their company to spread all over the country as well as can be used for foreign dealing of cars.

## **Motivation**

This software motivates other dealers as they can also start their work online as well as developers as they can adopt an initial idea in order to design any other software. This software can be created to develop interest of such people who feel shy while buying something online as they are frighten to be fooled by the dealers. In short this software can increase interest in young developers to design something relevant to the respected software.

## **Conclusion**

This software as merits as well as demerits depends on it’s use if used honestly can easily develop confidence of the users to buy or sell cars online but if not checked wisely then can cause devastating results.

## **Project Planning**

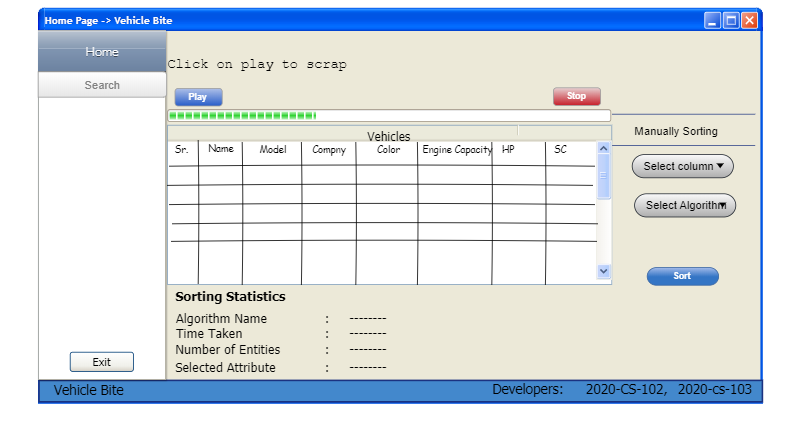
## **Developer**

* SaifUllah Sheikh(2020-cs-102)
* Muhammad Nabeel Yousaf(2020-cs-103)

## **UI Sample**

After a long discussion, we made a same UI model that we proposed in “Detailed Project Proposal”. Sample of UI model is given below:

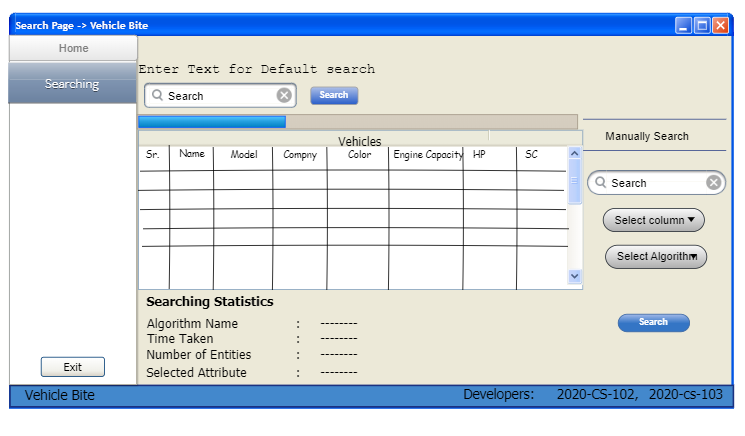
### **Sorting page:**



#### **Component Details:**

|  |  |  |
| --- | --- | --- |
| UI Component Name | Type of UI component | Purpose of UI Component/Other details |
| Home | Menu Button | To reach on home screen |
| Search | Menu Button | To reach on search screen |
| Products tabe | Table | To show vehicle |
| Exit Button | Button | To exit from system |
| Play button | Button | To start or resume scrapping |
| Stop button | Button | To stop scrapping |
| Information lables | Lables | To show Sorting Statistics |
| Select column | Combo box | To select column for sorting |
| Select algorithm | Combo box | To select Algorithm for sorting |
| Sort | Button | Start Sort |
| Progress | Progress Bar | To show scrapped vehicles |

### **Searching Algorithm:**



#### **Component Details**

|  |  |  |
| --- | --- | --- |
| UI Component Name | Type of UI component | Purpose of UI Component/Other details |
| Home | Menu Button | To reach on home screen |
| Search | Menu Button | To reach on search screen |
| Products tabe | Table | To show searched vehicle |
| Exit Button | Button | To exit from system |
| Information lables | Lables | To show Searching Statistics |
| Select column | Combo box | To select column for searching |
| Select algorithm | Combo box | To select Algorithm for searching |
| Sort | Button | Start Sort |
| Progress | Progress Bar | To show scrapped vehicles |
| Search | Text Box | To Enter text for search |
| Search button | Button | Start Search |
|  |  |  |

#### 

## **Task Division**

After assigning the groups, we were exploring different ideas to make a proper project. We rejected many topics due to different reasons. 2020-cs-102 suggested the topic upon vehicles and after exploring different sites we proposed this project.

After project we divide different tasks. A little summery of our decided work is in the following table:

|  |  |
| --- | --- |
| Task | Assigned to |
| Searching websites for scrapping | Both 2020-cs-102 and 2020-cs-103 |
| Scrapping of data | 2020-cs-103 |
| Making design | Both 2020-cs-102 and 1010-cs-103 |
| Project description | 2020-cs-103 |
| Business case | 2020-cs-102 |
| Executive summery | 2020-cs-103 |
| Technical details | 2020-cs-103 |
| Algorithm details | Both 2020-cs-103 and 2020-cs-102 |
| Implementation of design in Pencil Tool | 2020-cs-103 |

# **Algorithms**

|  |  |
| --- | --- |
| **Insertion Sort** |  |
| Description | In insertion sort, we assume that previous array is already sorted and start checking from 2nd element of the array. Is the current element is smaller than previous number then replace it with previous and doing it until it reaches to its actual position. It is just like the card sorting. In normal usage we use this type of sort.    Sorting by insert has many advantages. It is easy to implement and efficient enough for small data sets, especially when they are mostly ordered. It has low overhead and can sort the list as it receives data. Another benefit of insert sorting is that you only need one constant disk space for the entire process. It is more efficient than other similar algorithms, e.g. Bubble sorting or selection sorting.  In vehicle bite, our target is to sort data of vehicles in different manners. There are 11 attributes as we mentioned in previous report. Is we apply this of our data there is different ways to apply it. For example, if we apply it on engine capacity data will sort easily by comparing it the integer value of engine capacity. If we apply it on any alphabetic attribute, then we have to compare is with alphabetic characters and sort in ascending or descending according to alphabetic manner. |
| Pseudo Code | Function insertionSort(array) do:      for j = 1 to array.length  do:          key = array[j]          i = j-1  if(type == "Ascending"):              while i>-1 and array[i] > key:                  array[i+1] = array[i]                  i = i-1  end while loop  end if          else:              while i>-1 and array[i] < key:                  array[i+1] = array[i]                  i = i-1  end while loop  end else            array[i+1] = key  end for  end function |
| Code in Python | def insertionSort(array,type):      for j in range(1,len(array)):          key = array[j]          i = j-1          if(type == "Ascending"):              while i>-1 and array[i] > key:                  array[i+1] = array[i]                  i = i-1          else:              while i>-1 and array[i] < key:                  array[i+1] = array[i]                  i = i-1          array[i+1] = key |
| Code for Different attributes | def insertionSort(self,array,index,type):          for j in range(1,len(array)):                key = array[j]              i = j-1          #     print(array[j])                if(type == "Ascending"):                    while i>-1 and str(array[i][index]) > str(key[index]):                        array[i+1] = array[i]                      i = i-1                  #     print(array[i])              else:                  while i>-1 and str(array[i][index]) < str(key[index]):                        array[i+1] = array[i]                      i = i-1              array[i+1]= key          return array |
| Time Complexity Analysis | |  |  |  | | --- | --- | --- | | **Code:**  def insertionSort(array,type):      for j in range(1,len(array)):          key = array[j]          i = j-1          if(type == "Ascending"):              while i>-1 and array[i] > key:                  array[i+1] = array[i]                  i = i-1          else:              while i>-1 and array[i] < key:    array[i+1] = array[i]                  i = i-1          array[i+1] = key | **Cost:**  C1  C2  C3  C4  C5  C6  C7  C8  C9  C10  C11  C12 | **Time:**  n  n -1  n - 1  n- 1  ∑nj=2 tj  ∑nj=2 (tj -1)  ∑nj=2 (tj -1)  n-1  ∑nj=2 tj  ∑nj=2 (tj -1)  ∑nj=2 (tj -1)  n -1 | |  |  |  |   **For Ascending Order:**  T(n) = c1 + c2(n-1) + c3(n-1) + c4(n-1) + c5(∑nj=2 tj) + c6(∑nj=2 (tj -1)) + c7(∑nj=2 (tj -1)) + c12(n-1)  T(n) = c1 + c2(n-1) + c3(n-1) + c4(n-1) + c5( -1) + c6() + c7() + c12(n-1)  After further solving this equation, following equation will become  T (n) = (c5/2 + c6/2 + c7/2)n2 + (c1 + c2  + c4 + c5/2 - c6/2 - c7/2 + c12)n – (c2 + c4 + c5 + c12)  **T(n) = O(n2 )**  **For Descending Order:**  T(n) = c1 + c2(n-1) + c3(n-1) + c4(n-1) + c9(∑nj=2 tj) + c10(∑nj=2 (tj -1)) + c11(∑nj=2 (tj -1)) + c12(n-1)  T(n) = c1 + c2(n-1) + c3(n-1) + c4(n-1) + c9( -1) + c10() + c11() + c12(n-1)  After further solving this equation, following equation will become  T (n) = (c9/2 + c10/2 + c11/2)n2 + (c1 + c2  + c4 + c9/2 - c10/2 - c11/2 + c12)n – (c2 + c4 + c9 + c11)  **T(n) = O(n2 )**  **Best case:** Ω(n)  **Worst case:** O(n2)  **Average case:** Θ(n2) |
| Proof of Correctness | **Initialization:**  Before the loop j = 2 ⇒ A [1 .. j1] = A [1] which only contains the elements A [1 .. j1] (of which there is only one) and since there is only one element, they are trivial orderly .  **Maintenance :**  The outer for loop selects the A [j] element and places it correctly in A [1..j1] via the while loop. Since array A [1..j1] has been sorted, place element A [j] at the correct position produces A [1..j] in order (and contains the first j elements).  **Termination:**  The loop ends when j = n + 1 ⇒ A [1 .. j1] = A [1 .. (n + 1) 1] = A [1 .. n] which since the array remains sorted after each iteration gives A [1 .. n] is sorted when the loop ends and contains all of the original elements, the entire original array is sorted. |
| Three Strengths | 1. It is stable sort. 2. Efficient for small data set. 3. It become fast if data is already sorted because its best case is O(n) |
| Three Weakness | 1. For large data sets, it contains large time because its time complexity is O(n2). 2. It iterates array again and again, so time increases 3. It is not efficient as other algorithms 4. If we add new element it checks almost all emenents if element is larger. |
| Dry Run | |  |  |  |  |  | | --- | --- | --- | --- | --- | | **j** | **key** | **i** | **A[j]** | **A** | |  |  |  |  | **{**1,2,4,9,7,6**}** | 2 | 4 | 9 | 7 | 6 | | 2 | 2 | 1 | 2 | **{**1,2,4,9,7,6**}** | | 3 | 4 | 2 | 4 | **{**1,2,4,9,7,6**}** | | 4 | 9 | 3 | 9 | {1,2,4,9,7,6} | | 5 | 7 | 4 | 9 | {1,2,4,9,7,6} | | 5 | 7 | 3 | A[4] = 7 | {1,2,4,7,9,6} | | 6 | 6 | 5 | 9 | {1,2,4,7,6,9} | | 6 | 6 | 4 | A[5] = 7 | {1,2,4,7,7,6} | | 6 | 6 | 3 | A[4] = 6 | {1,2,4,6,7,9} | |

|  |  |
| --- | --- |
| **Selection Sort** |  |
| Description | Sorting by selection is a simple sorting algorithm. This sorting algorithm is a comparison-based algorithm in which the list is divided into two parts, the ordered part on the far left and the unordered part on the far right. The ordered part is initially empty and the unsorted part is the entire list.    The smallest element is selected from the unordered array and swapped with the leftmost element, and that element becomes part of the ordered array. This process continues by moving the boundary of the unordered array one element to the right.  In vehicle bite, we can use this type of sort to manage data in ascending or descending order. We can apply it on prices of vehicles because it is already inefficient for large amount of data and we also have data in alphabetic characters. If we apply it on alphabetic attributes, maybe it will increase time complexity too high.  This algorithm is not suitable for large data sets because its average and worst-case complexity is Ο (n2), where n is the number of elements. |
| Pseudo Code | function selectionSort(array, size):        for j to size:          min = j          for i = j + 1 to size:                if array[i] < array[min]:                  min = i          swap(array[j], array[min]) |
| Code in Python | def selectionSort(array, size):        for j in range(size):          min = j          for i in range(j + 1, size):                if array[i] < array[min]:                  min = i            array[j], array[min] = array[min], array[j] |
| Code for different Attributes | def selectionSort(self,array,index,type):          size = len(array)          for j in range(size):              min = j              if(type == "Ascending"):                  for i in range(j + 1, size):                        if str(array[i][index]) < str(array[min][index]):                          min = i              else:                  for i in range(j + 1, size):                        if str(array[i][index]) > str(array[min][index]):                          min = i                array[j], array[min] = array[min], array[j]          return array |
| Time Complexity Analysis | |  |  |  | | --- | --- | --- | | **Code** | **Cost** | **Time** | | def selectionSort(array, size):        for j in range(size):          min = j          for i in range(j + 1, size):                if array[i] < array[min]:                  min = i            array[j], array[min] = array[min], array[j] | C1  C2  C3  C4  C5  C6 | n+1  n  ∑ni=j+1 (ti +1)  ∑ni=j+1 ti  ∑ni=j+1 ti  n |   T(n) = (n+1) +n + (∑ni=j+1 (ti +1) ) + (∑ni=j+1 ti ) + (∑ni=j+1 ti ) + n  T(n) = 3n + + (∑ni=j+1 (ti +1) ) + (∑ni=j+1 ti ) + (∑ni=j+1 ti ) + 1  After solving this equation we will reach at the following solution:  T(n) = O(n2)  **Best case:** Ω(n2)  **Worst case:** O(n2)  **Average case:** Θ(n2) |
| Proof of Correctness | **Initialization**:  The loop invariant is true at the beginning of the loop. The beginning of the loop is when j = i+1 and min = i, so the loop invariant states that array[i] is the smallest which is true.  **Maintenance:**  The invariant is preserved in the loop body (including the increment of the loop index j). We know that array[min] is the smallest of array[i..j] between lines 4 and 5. The body of the loop checks if array[j] is smaller than array[min] and sets min to j if it is, thus we know between lines 7 and 8 that array[min] is the smallest of array[i...(j+1)]. The loop invariant will still be true between lines 4 and 5 when j is incremented during the loop.  **Termination:**  As a result, we can deduce that the invariant is true once the loop is completed, which occurs when j=len (array). We know that on line 9 array[min] is the smallest from among L[i..len(array)] by replacing j=len(array) into the loop invariant. |
| Three Strengths | 1. It performs very well if data set is small. 2. It become fast if data is already sorted. 3. Too large space is not required because it is in-place algorithm and only one space required for an extra/temporal variable. |
| Three Weakness | 1. For large data set it is inefficient because time complexity will increase a lot. 2. It is unstable sort. 3. Other algorithms, such as quicksort, have better performance compared to the selection sort. |
| Dry Run | Let the input array A is   |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 0 | 1 | 2 | 3 | 4 | 23 | 10 | 16 | 11 | 20 |   When=1:   |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 0 | 1 | 2 | 3 | 4 | 10 | 23 | 16 | 11 | 20 |   After i=1:   |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 0 | 1 | 2 | 3 | 4 | 10 | 11 | 16 | 23 | 20 |   After i=2:   |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 0 | 1 | 2 | 3 | 4 | 10 | 11 | 16 | 23 | 20 |   After i=3:   |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 0 | 1 | 2 | 3 | 4 | 10 | 11 | 16 | 20 | 23 |   After i=4:   |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 0 | 1 | 2 | 3 | 4 | 10 | 11 | 16 | 20 | 23 | |

|  |  |
| --- | --- |
| **Merge Sort** |  |
| Description | Merge Sort is a divide and conquer algorithm. It works by recursively dividing a problem into two or more identical or related subproblems, until they become simple enough to be solved directly. The solutions of the subproblems are then combined to give a solution to the original problem.    In vehicle bite, we can apply this sort to arrange alphabets because it is faster than other sorting algorithm and performs better than other sorting algorithms. We can apply this method to sort data of different types because its time complexity is consistent and it is a stable sort.  It is an efficient algorithm. It sort data set with average time complexity O(nlog n ). |
| Pseudo Code | Function merge(arr, l, m, r):      n1 = m - l + 1      n2 = r - m      L = array of size n1      R = array of size n2      for i = 0 to n1          L[i] = arr[l + i]      for j = to n2          R[j] = arr[m + 1 + j]      i = 0      j = 0      k = l      while i < n1 and j < n2:          if L[i] < 0              arr[k] = L[i]              i =i+ 1          else              arr[k] = R[j]              j =j+ 1          k = k + 1      while i < n1          arr[k] = L[i]          i = i+ 1          k = k+ 1      while j < n2          arr[k] = R[j]         j =j + 1        k =k + 1  end function  function mergeSort(arr, l, r):      if l < r          m = l+(r-l)/2          mergeSort(arr, l, m)          mergeSort(arr, m+1, r)          merge(arr, l, m, r)  end function |
| Code in Python | def merge(arr, l, m, r):      n1 = m - l + 1      n2 = r - m      L = [0] \* (n1)      R = [0] \* (n2)      for i in range(0, n1):          L[i] = arr[l + i]      for j in range(0, n2):          R[j] = arr[m + 1 + j]      i = 0      j = 0      k = l      while i < n1 and j < n2:          if L[i] < 0:              arr[k] = L[i]              i += 1          else:              arr[k] = R[j]              j += 1          k += 1      while i < n1:          arr[k] = L[i]          i += 1          k += 1      while j < n2:          arr[k] = R[j]          j += 1          k += 1  def mergeSort(arr, l, r):      if l < r:          m = l+(r-l)//2          mergeSort(arr, l, m)          mergeSort(arr, m+1, r)          merge(arr, l, m, r) |
| Code for different attributes | def merge(self,arr, l, m, r,index):          n1 = m - l + 1          n2 = r - m          L = [0] \* (n1)          R = [0] \* (n2)          for i in range(0, n1):              L[i] = arr[l + i]          for j in range(0, n2):              R[j] = arr[m + 1 + j]          i = 0          j = 0          k = l          while i < n1 and j < n2:              if str(L[i][index]) <= str(R[j][index]):                  arr[k] = L[i]                  i += 1              else:                  arr[k] = R[j]                  j += 1              k += 1          while i < n1:              arr[k] = L[i]              i += 1              k += 1          while j < n2:              arr[k] = R[j]              j += 1              k += 1      def mergeSort(self,arr, l, r,index):          if l < r:              m = l+(r-l)//2              self.mergeSort(arr, l, m,index)              self.mergeSort(arr, m+1, r,index)              self.merge(arr, l, m, r,index) |
| Time Complexity Analysis | **Best case:** Ω(n log n)  **Worst case:** O(n log n)  **Average case:** Θ(n log n) |
| Proof of Correctness | **Initialization**:  We have k = p prior to the first iteration of the loop, hence the subarray A[p... k - 1] is empty. Since I = j = 1, both L[i] and R[j] are the smallest items of respective arrays that have not been cloned back into A, this empty subarray includes the k - p = 0 smallest elements of L and R.  **Maintenance:**  Let us first assume that l[i] = R[j] so that each iteration maintains the loop invariant. The smallest element not yet copied back into A is L[i]. Because A[p... k - 1] contains the k - p smallest items, the subarray A[p... k] will include the k - p + 1 smallest elements after line 14 duplicates L[i] into A[k]. The loop invariant is re-established for the following iteration by incrementing k(in the for loop update) and i(in line 15). If, instead, L[i] > R[j], lines 16-17 take the necessary steps to keep the loop invariant.  **Termination:**  k = r + 1 when the programme ends. The subarray A[p... k - 1], which is A[p... r], comprises the k - p = r - p + 1 smallest items of L[1... n1 + 1] and R[1... n2 + 1], in sorted order, thanks to the loop invariant. Together, the arrays L and R have n1 + n2 + 2 = r - p + 3 items. All but two of the largest items have been cloned back into A, and the sentinels are the two largest elements. |
| Three Strengths | 1. It is stable algorithm. 2. It contains similar time because its time complexity if same i.e. n logn. 3. It is faster than insertion and selection sort for huge data sets. |
| Three Weakness | 1. It is slower than other sorting algorithms if there is a small data set. 2. No of steps and time is same even the array is sorted or unsorted. 3. More memory is required because this method splits array recursively and divides it in two arrays. So more memory is required if data is increasing or no of recursive call will increase. |
| Dry Run | |  |  |  | | --- | --- | --- | | **Unsorted list** | **Divide** | **Sorted list** | | {12,23,2,43,51,35,19,4} | {12,23,2,43}  {51,35,19,4} | {} | | {12,23,2,43}  {51,35,19,4} | {12,23} {2,43}  {51,35} {19,4} | {} | | {12,23} {2,43}  {51,35} {19,4} | {12,23} {2,43}  {35,51} {4,19} | {12,23} {2,43}  {35,51} {4,19} | | {12,23} {2,43} {35,51} {4,19} | {2,12,23,43} {4,19,35,51} | {2,12,23,43} {4,19,35,51} | | {2,12,23,43} {4,19,35,51} | {2,4,12,19,23,35,43,51} | {2,4,12,19,23,35,43,51} | | {} | {} | {2,4,12,19,23,35,43,51} |   **Unsorted array:** {12,23,2,43,51,35,19,4}  After applying merge sort:  **Sorted array:** {2,4,12,19,23,35,43,51} |

|  |  |
| --- | --- |
| **Bubble Sort** |  |
| Description | The bubblesort iteratively traverses a list. It compares things that are nearby and swaps those that are out of order. The next maximum value is placed in the appropriate position each time the list is traversed. Each object "bubbles" up to its proper spot. A bubble sort is one of the different sorting algorithms. It is taught to us as a basic sorting algorithm. This is simple to learn and has no learning curve. It is simple to include into the code, which is quite helpful for novices. However, it is the poorest method for sorting the items in any except because it verifies whether the array is sorted or not every time.  We must construct a bubble sort() method that accepts list1 as an input. We constructed two for loops within the function: the first for loop iterates the whole list, and the second for loop iterates the list and compares the two entries in each outer loop iteration. When the for loop reaches the end, it will be stopped. In the inner for loop, we defined the condition: if the first index value is bigger than the second index value, swap their locations. We gave a list to the function, which iterated and returned the sorted list.  Bubble sort is mostly used in education to assist students comprehend the fundamentals of sorting. This is used to determine if the list has previously been sorted. The complexity of bubble sort is only O(n) when the list is already sorted (which is the best-case situation). in worst case it, iterates over n-1 iterations, looking at n-1 pairs of neighbouring entries. O(n2) is regarded as a terrible sorting algorithm. |
| Pseudo Code | Function bubbleSort(array)      for i = 0 to array.length      for j = 0 to array.length - i - 1        if array[j] > array[j + 1]          swap(array[j], array[j+1] )         end if  end for  end for  end function |
| Code in Python | def bubbleSort(array):      for i in range(len(array)):      for j in range(0, len(array) - i - 1):        if array[j] > array[j + 1]:          array[j], array[j+1] = array[j+1], array[j] |

|  |  |
| --- | --- |
| **Quick Sort** |  |
| Pseudo Code | Function partition(array , low , high):      pivot = array[high]      i = low-1      for j = low to high          if array[j] < pivot              i = i+1              swap(array[i], array[j])      end if  end for      swap(array[i+1] , array[high] )      return i+1  end function  Function quickSort(array,low , high)      if low <high          pi = partition(array,low,high)          quickSort(array,low,pi-1)          quickSort(array,pi+1,high)  end if  end function |
| Code in Python | def partition(array , low , high):      pivot = array[high]      i = low-1      for j in range(low,high):          if array[j] < pivot:              i = i+1              array[i], array[j] = array[j], array[i]        array[i+1] , array[high] = array[high],array[i+1]      # print(array)      return i+1  def quickSort(array,low , high):      if low <high:          pi = partition(array,low,high)          quickSort(array,low,pi-1)          quickSort(array,pi+1,high) |

|  |  |
| --- | --- |
| **Heap Sort** |  |
| Description | Heap sorting is the type of algorithm in which comparision based sorting is used. In this type of sorting one value of array is taken as reference point and then it is compared with the whole array. If the value is small then the reference point then it changes the reference point an if the value is greater then the reference point then it moves to next value. It uses binary tree for completion of the array. Any value is taken from the array which might be greatest of all values which is kept as the base of tree then it is noted that the the branches of the respected tree has less value then the main tree  POINTS TO BE NOTED:  Heap sort is a type of sorting which is used in-place.  Heap sorting is not stable algorithm but can be made stable.  **TIME AUXILIARY**:  The time auxiliary used for heap sorting is O(logn)  **WORST CASE SPACE :**  The worst case space for heap sorting is O(0) or O(1)  **BEST CASE:** The best case for heap sorting is O(nlogn).  ADVANTAGES: Heap sorting is easy to understand and is simple. It is more effective then any other sorting and is time saving.  **EXAMPLE:**  23,6,43,1,78,9,87,3 In the given example by using heap sorting the biggest value of array is taken as the reference point and is kept at the top and then others value are checked according to the reference point. The given example after heap sorting is as follow  1,3,6,9,23,43,78,87  **USE OF PRIORITIES:**  Heap sorting can also be used for priority as it as time complexity of O(logn). . Binomoial Heap and Fibonacci Heap are also used in heap sorting. |

|  |  |
| --- | --- |
| **Counting Sort** |  |
| Description | Counting sorting is the type of sorting algorithm in which sorting of collection of data is done in order in small positive number in an array. Integars are computed using counting sorting.  Its running time is linear in the number of items and the difference between the maximum key value and the minimum key value, so it is only suitable for direct use in situations where the variation in keys is not significantly greater than the number of items.  WORST CASE PERFORMANCE: The worst case performance of counting sorting is O(n+k) in which k is the range of non-negative numbers.  **WORST CASE SPACE COMPLEXITY:**  The worst case space complexity of counting sorting is O(n+k).  **TIME COMPLEXITY:**  The time complexity of counting sort is O(n+r).  **EXAMPLE:**  Consider the following example of array having values as follow  2,2,54,1,34,33,4,33,43,54 In the given example by using counting sorting the sorting will start from the smallest number which is 1 and then it will find next small number and then arrange them in an ascending order. The array after sorting of counting will be as follow  1,2,2,4,33,33,34,43,54,54  **POINT TO BE NOTED:** Counting sort is efficient if the range of input data, k*k*, is not significantly greater than the number of objects to be sorted, n*n*.  **STABILITY:** Counting sort is an stable type of sorting algorithm in which the space complexity is O(n+K).  **CONCLUSION:** Counting sorting is used in daily life as it is stable as well as it is an accurate type of sorting algorithm. |

|  |  |
| --- | --- |
| **Shell Sort** |  |
| Description | Shell sorting is combination of both bubble and insertion sorting. It is an in-comparsion sorting which can use exchange(bubble sort) or insertion(insertion sort) to assemble the required data. It is also known as diminishing increment sort. It is one of the oldest sorting used in sorting algorithm.  **AVERAGE TIME COMPLEXITY:** The average time complexity of shell sorting is near to O(n).  **TIME COMPLEXITY**: The time complexity of shell sorting depends on the gap sequence.  **BEST TIME COMPLEXITY:** The best time complexity of shell sorting is O(n\*logn).  **WORST TIME COMPLEXITY:** The worst time complexity of shell sorting is O(n\*log^2\*n).  **STABILITY:** The shell sorting is not a stable sorting as relative value of the given number may change.  **POINT TO BE NOTED:** It is an in-place sorting algorithm as it requires no additional scratch space.  **EXAMPLE:**  Consider the given example of array havine following numbers  34,55,23,76,87,54,54,55,76,1 In the given example by using shell sorting it can be arranged by either bubble or insertion sorting. It will arrange them in an increasing order. The example after shell sorting will be as follow:  1,23,34,54,54,55,55,76,76,87  **ADVANTAGES:** Shellsort performs more operations and has higher cache miss ratio than quicksort. However, since it can be implemented using little code and does not use the call stack, some implementations of the qsort function in the C standard library targeted at embedded systems use it instead of quicksort.  **CONCLUSION:** Shell sorting is useful in different cases but it is no stable as insertion and merge sorting. |

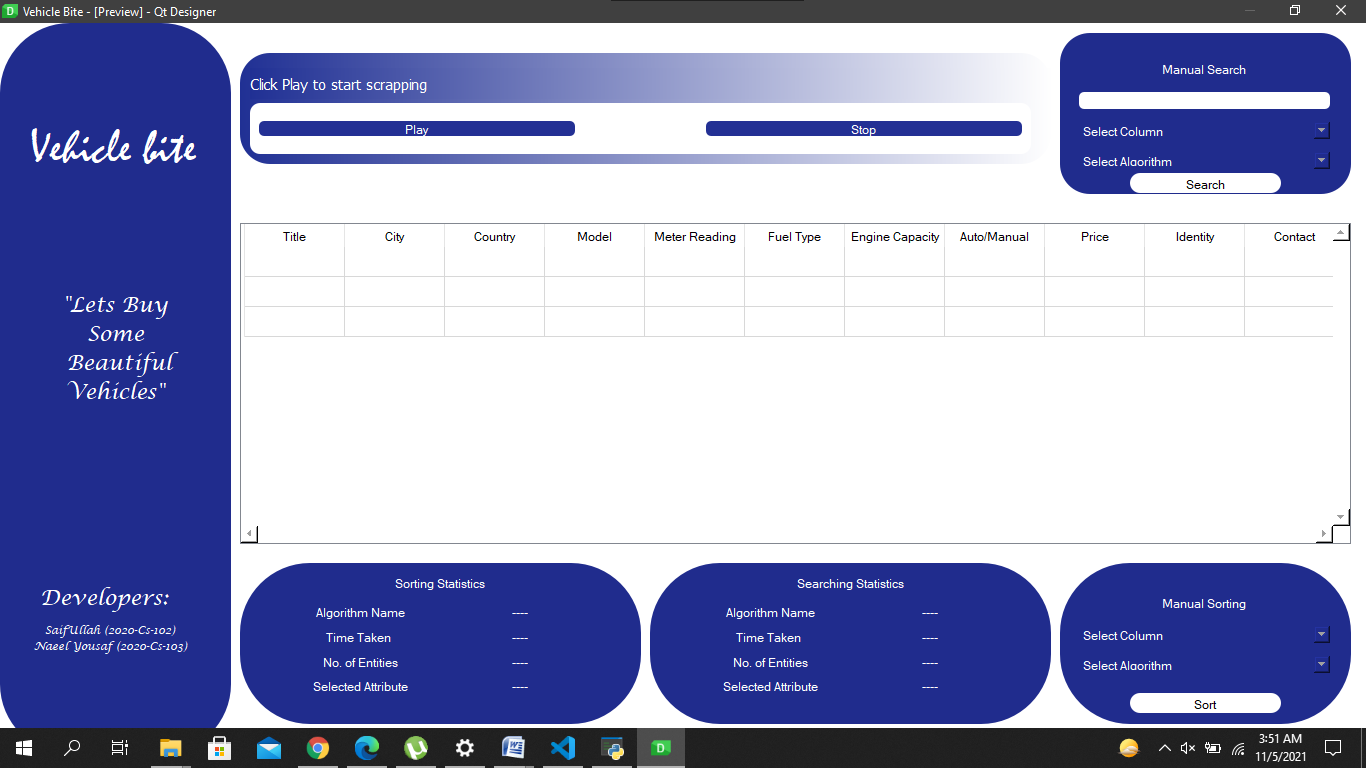
# **Final Application**

After a long procedure, we reached at a particular point where our project is just reached to its completion.

## **UI Model**

### **First UI Model:**

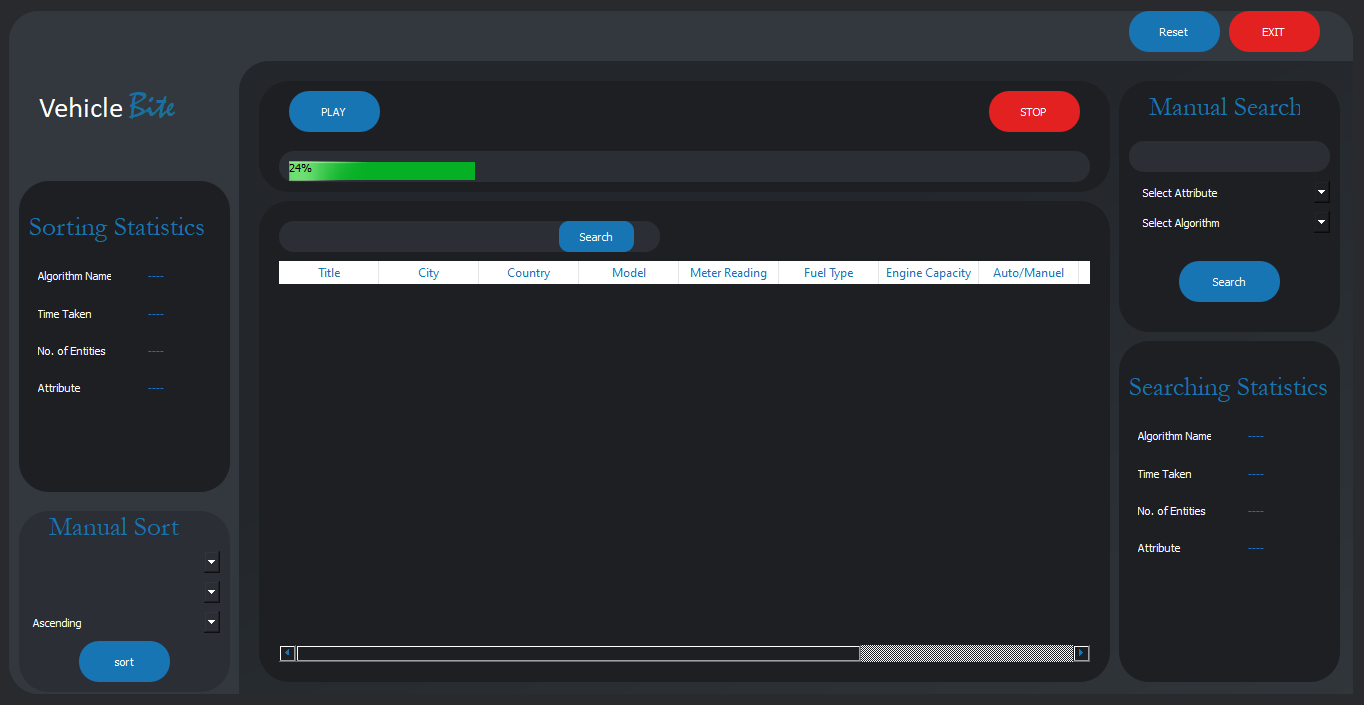
Firstly a model has been created by us, but it has been rejected due to some issues. The snapshot of the model is given below:



**Error**: There is a bug in this model. It was stuck due to some unexpected errors and the scrollbars of this model was not working properly. So we recreated a new User Interface to complete our work.

### **Final UI Model**

Snapshot of final model is given below:



### **Components of UI**

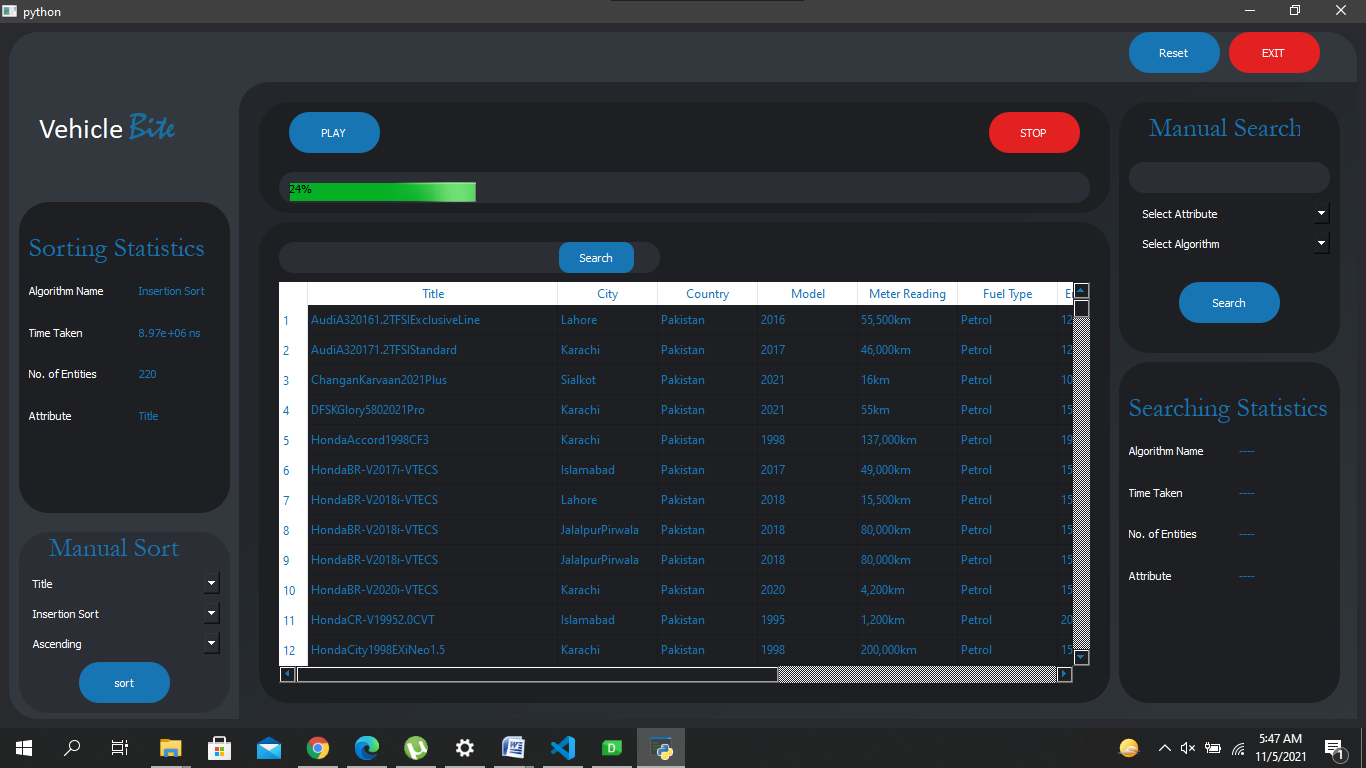
The components of final model are given below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Component Name** | **Component Type** | | **Object Name** | **Purpose with snapshot** |
| Vehicle Bite | Label | | label1 and lable\_2 | It reflects the name of our software. It acts like a logo or a sign which represents the name any company. |
| Play | PushButton | | Play | When we start the push button, it will start the scrapping and load data in table and you can also see the change in progress bar. |
| Stop | PushButton | | Stop | When we stop the push button, it will stop the scrapping and load data in table and progress bar will also stop. |
| Type Search | editLine | | searchLine | We can write here the text that we want to search. |
| Search | pushButton | | searchDef | It will search the text that we have entered in the “editLine” , just before the Button |
| Progress Bar | progressBar | | progressBar | It will show how any data has been scrapped out of 1.5 million and when we star scrapping, it will run and show the number of scrapped data. |
| Exit | pushButton | | Exit | After pressing this key, our system will close. |
| Reset | pushButton | | reset | After pressing this key, all the data will reset and every label will reset as its actual or initial position. |
| **Sorting Statistics** | | | | |
| Algorithm Name | label | | sort\_algoName\_2 | It will show the name of selected algorithm for sorting. |
| Time Taken | label | | sort\_TimeTaken\_2 | It will show the time taken by the selected algorithm for sorting. |
| No. Of Entities | label | | sort\_noOfEntities\_2 | It shows that how much entities have been sorted. |
| Attribute | label | | sort\_SelectedAttribute\_2 | It reflects that which attribute is selected as a base or a key for sorting. |
| Snapshot |  | | | |
| **Manual Sort** | | | | |
| Select Attribute | ComboBox | sort\_attributeCombo | | It is used to select attribute as a base or key for sorting. |
| Select Algorithm | ComboBox | sort\_AlgoCombo | | It selects the algorithm for sorting. |
| Mode | ComboBox | sort\_ModeCombo | | It selects the mode i.e. Ascending or Descending. |
| Snapshot |  | | | |
| **Manuel Search** | | | | |
| Select Attribute | ComboBox | search\_attributeCombo | | It is used to select attribute as a base or key for searching. |
| Select Algorithm | ComboBox | search\_AlgoCombo | | It selects the algorithm for searching. |
| Type for Search | EditLine | searchEditLine | | We can write here the text that we want to search from selected attribute. |
| Search | pushButton | mSearchButton | | It will search the text that we have entered in the “editLine” , just before the Button, from selected attribute using selected algorithm. |
| Snapshort |  | | | |
| **Searching Statistics** | | | | |
| Algorithm Name | label | search\_algoName\_2 | | It will show the name of selected algorithm for searching. |
| Time Taken | label | search\_TimeTaken\_2 | | It will show the time taken by the selected algorithm for searching. |
| No. Of Entities | label | search \_noOfEntities\_2 | | Shows the result taken from how many entities. |
| Attribute | label | search\_SelectedAttribute\_2 | | It reflects that which attribute is selected as a base or a key for searching. |
| Snapshots |  | | | |
| **Table** | | | | |
| Snapshot | tableWidget | table | | It is used to display all data. |
| Snapshots |  | | | |

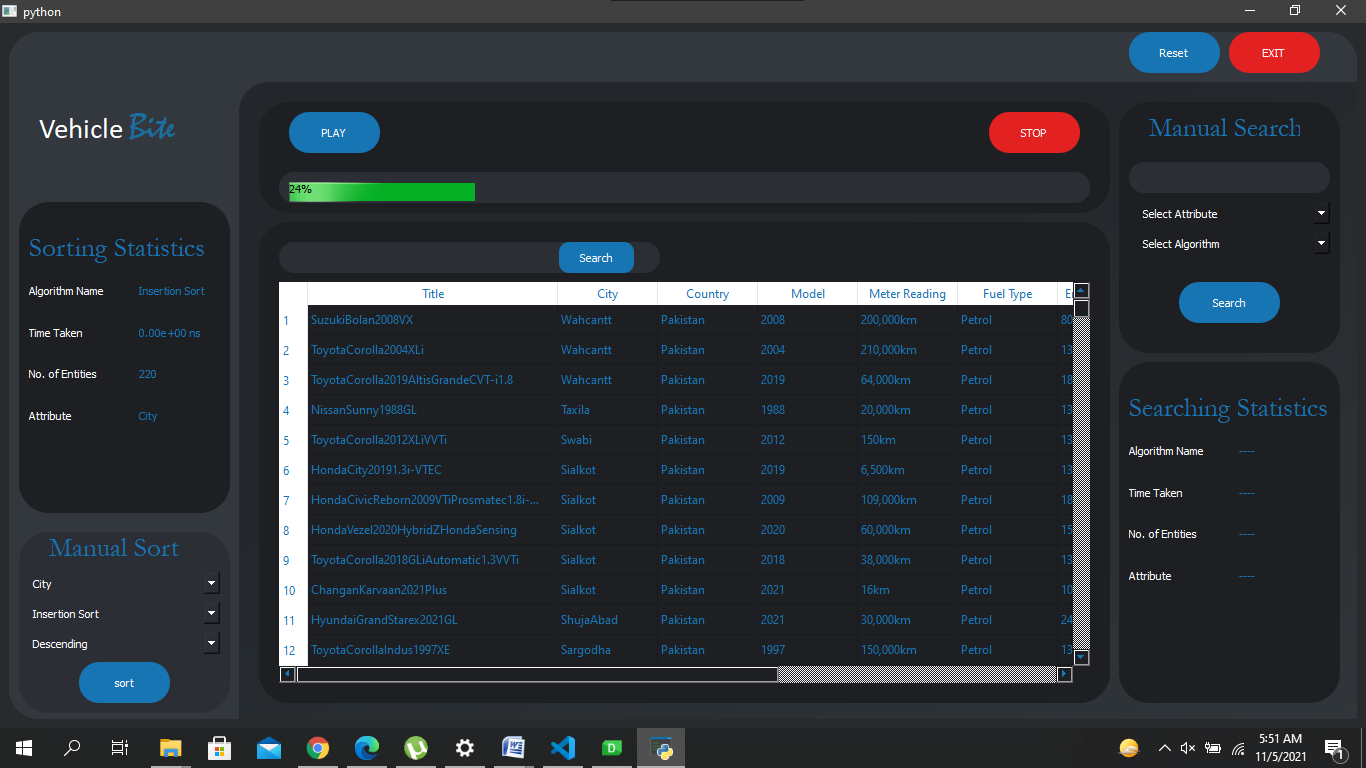
## **Final Testing**

### **Insertion sort Using different attributes:**

**Using Title in Ascending order:**

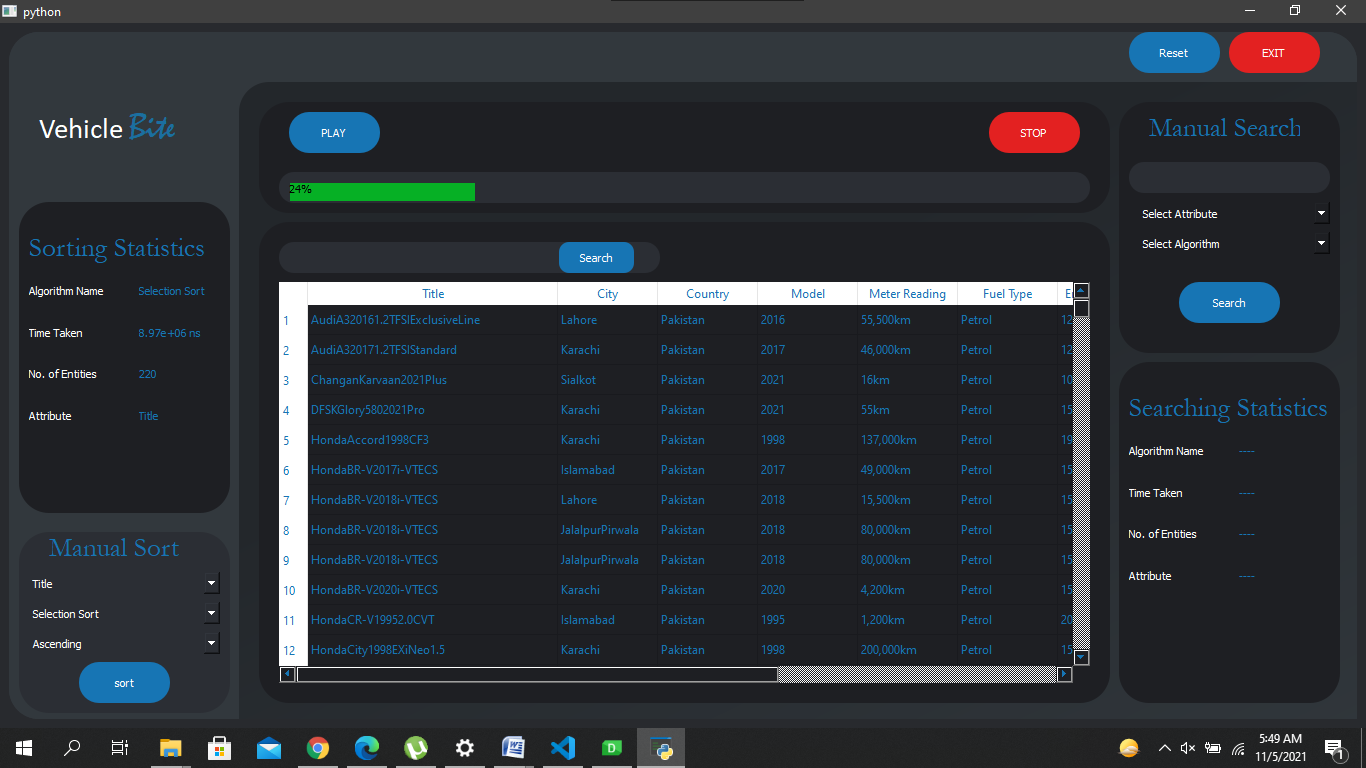
****

**Using City in descending mode:**

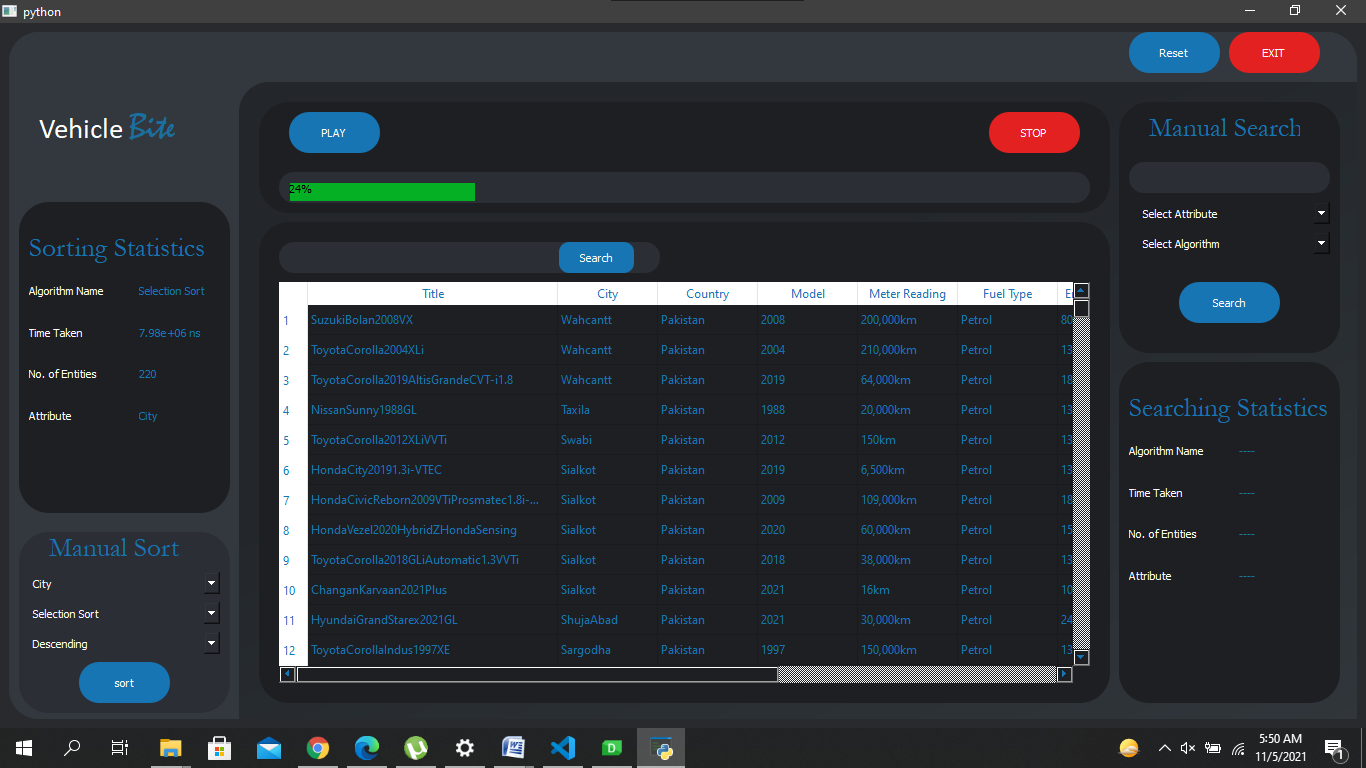
****

### **Selection sort Using different attributes:**

**Using Title in Ascending order:**

****

**Using City in Descending order:**

****

### **Merge Sort:**

### 

# \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_