

CS261-Data Structures and Algorithms

Mid-Term Final Report (Fall 2021)

**Mid-Term Project Title**

**Product-Zilla**

**Members**

|  |  |
| --- | --- |
| **2020-CS-64** | **Qazi Maaz Sajjad** |
| **2020-CS-63** | **M. Abdullah Butt** |
| **Section-B** | **Section-B** |

**Group-ID**

**10**

**Submitted To**

**Professor Samyan Qayyum Wahla**

Department of Computer Science

U.E.T Lahore, Pakistan

**Index:**

* Project Description \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_03
* Class Details \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_03
* Learning Outcome \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_04
* Data Scrapping \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_05

1. Libraries \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_05
2. Problems \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_06
3. Attributes \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_07
4. Sources \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_08
5. Algorithms \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_09

* Resource Management \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_32

1. Planning \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_32
2. Integration \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_32

* Graphical User Interface \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_33
* User Manual \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_35

**University of Engineering and Technology**

**Project Description:**

This project is all about the scrapping of data available on the internet. We are going to scrap the data from the website **Ali Express, Daraz.pk.** What we are going to do is to scrap the multiple data from the website and place them in the categories. The user will be given the complete authority in our project called **ProductZilla** to start the scrapping process anytime he wants by just a single click. Then our software will be completely reliable on to this clicking process. Once the user has pressed the button the system will start scrapping the data.

The progress of the scrapping process can be seen through the progress bar. There will also be a pause/play button that will allow the user to pause this scrapping process anytime he wants to. Once this button is pressed the title of the button will be changed. Correspondingly it will stop the scrapping process for the time being till the button is again pressed to resume this process. Till the button is pressed the progress bar will be stopped and a small values will appear indicating the percentage of the data that has been scrapped and the remaining data as well. When the system is resumed again the progress bar continues and scrapping process starts from the place where it was stopped. Once the scrapping is complete all the data will be saved in the csv file and will be displayed in the table format in the page given.

Apart from all of the above data we will also provide the user a facility to download the CSV File by just a single click. Not only this the user will be given different categories out of the products from which we are going to scrap the data. On that category data will be scrapped nothing else. Above all we will provide the algorithms to sort the column as per user demand using the **kendo UI grid.** And if the user wants then the user will be provided with the option to multi-level sorting of the data. All of these features will be completely available for user facility.

**Class:**

There is going to be only one class named “Scrapped Data”. The sole purpose of this class is to hold the data as per the entities available in the website that will be the attributes of this class. All the operations regarding the attributes will be performed in this class. Following are the attributes of this class.

* **Title:**

This will contain the title of the product that is available in the store along with a short description.

* **Price:**

This will contain the integer or float value that will be the price (excluding tax) of the corresponding product.

* **Sold Items:**

This will again contain the integer value describing the number of items that has been sold of the same product

* **Ratings:**

This attribute contains the rating (float value) of the product from the user reviews.

* **Shipment tax:**

This includes the shipment tax that will be applied to the product when it will delivered to the consumer.

* **Store Name:**

This attribute includes the name of the store from where this product will be delivered to the user.

* **Top Seller Value:**

This will include the Boolean value that will indicate whether the given product is the top seller or not

**Learning Outcome:**

Through the whole process we were able to know how to deal with a huge data of almost a million length. During the whole process we encountered a lot of problem that we dealt by reading articles through different sources. Also the process of scraping was introduces to us from a completely different vision. In the beginning we thought that we wouldn’t be able to go through the whole process but with the passage of time, once we went through the stages of this process a lot of thing were enlightened to us.

It was for the first time that were doing **scrapping** as a huge project. Although it felt difficult to us in the beginning. Then we started studying about it and then slowly we were able to digest the rules of doing scraping and finally following them we were able to scrap the data off the website and store that in csv file for later use.

Not only this had we also learned how to handle huge data and how to perform certain operations when we are dealing with such a huge amount of data. The operation also included one of the main concepts of our course called **implementation of algorithms.** We had only dealt with data of length no more than 20 before this project on which we had applied all sorting and searching algorithms but also we had never encountered any kind of problems related to string and all other data types. This was for the first time that we were using our knowledge of algorithms to implement the sorting or searching to such an enormous amount of data.

Not forgetting about the **UI design.** It was one of the major task to perform in this project keeping in mind that we had never used python before. Implementation of this stage was quite difficult for us. But at the end, we are now quite confident to use it in future with much greater ease.

**Data Scrapping:**

|  |  |
| --- | --- |
| **Library** | All the libraries used during this process and their details |
| sys | This library will give us a wide access to a control of the function to change the current runtime environment of python. One of the basic use of this library is while ending the program we use its module .exit for finishing the program. |
| PyQt5 | This library is used in making the graphical user interface for the project like this one for interacting with the user in friendly manner |
| selenium | It can also be considered a web testing tool or web testing library using which we intact the main drive with the web and perform certain operations like scrapping. In this project we have used this library for the web driver that maintains the web page for loading and scrapping data. |
| bs4 | One of the most important library that python provides user with is bs4 for getting the data out of html or xml paths of the web pages. We also have imported one of it module called BeautifulSoup for scrapping data out of the html web page. |
| pandas | Panda library is one of the notorious libraries for providing the data frames in which we can store data manipulate data and perform other operations as well with greater ease. We have repeatedly used this library to make the csv file of the data in this project |
| time | This library’s module called “sleep” is used in this project in order to allow the system to wait for the web page to load or to halt the request on a web page for a certain time. |
| random | This library is widely used for generating the random numbers. We have used a module named randint for getting a random integer within a given range so as to implement the sleeping module on that integer mille-seconds. |
| decimal | This library is used to convert the values of other formats into special value called decimals. It was used during scrapping for our one od the attribute called “ratings”. |
| **Problems** | All the problems that we faced while the implementation and scrapping the data |
| Poor Internet Connection | The most common problem that we encountered during the whole process was our poor connection of the internet. When we were in the middle of scrapping there was a sudden loss of internet connection or sometimes signals got lost and sometimes the ping got high. Due to which either the web driver was working very slow or the web driver did not even opened. So in order to **solve** **the problem** we applied some threading in the process due to which we were able to recover few data during the run time. |
| Not Loading the page | Sometimes the web driver was not able to load the page during run time either due to poor internet connection or due to some anonymous reason. Due to this program was not picking the complete data during scrapping and leave half of the data unloaded moving to the next page. So in order to **solve this problem** we needed the help of sleep library to make the program wait till the page loads. Not only this we also had to scroll the web driver to the bottom in order to make sure that the page is completely loaded and the program is picking all the data. |
| Poor Website structure | The website that we had chosen for the scrapping had an inadequate structure due to which we faced a lot of problems while scrapping. It was that the website had repeated its data after a certain number of pages and also the pages they had mentioned at the bottom of the web page were not available. They were just for the ornamentation. Also there was no symmetry while we were choosing the classes for picking the data. We had to change multiple categories and methods to get our data scrapped. |
| Changing Website | Due to Security reasons, the website **Ali Express** had changed the format of their website and were changing their classes every hour. Due to which there was no symmetry left that we could catch to do scrapping so we had to stop further scrapping from that website and shift to the new website called **Daraz.pk.** But we chose each and every category of this website to scrap and did it by that way. |
| **Attributes** | The attributes and their details that we were scrapping. |
| Name | This is going to be the title of the product that we are scrapping and is going to help us to perform many operations most importantly the searching function. Its data type is going to be **string.** It can either be a two to three word or a complete description of the product describing the properties of the product. Multiple sorting algorithm are going to be applied to this attribute and those are going to be one of the following:   * A-Z * Z-A |
| Price | This is going to the price of the product of the relative product whose name had been scrapped and is going to be in PKR. The data type of the price is going to be **float.** This product is also going to help us in sorting as well as multilevel sorting. Following process will be applied to perform sorting on this data:   * Low-to-High * High-to-Low |
| Sold Items | This is going to be a count of the items of the product that has been sold till the present date. It will start from 0 and will move till end. The data type of this attribute is going to be **integers.** The following sorting can be applied using this attribute   * Above 200 * Below 200 |
| Ratings | This attribute is going to be a review count of a particular product that how much the users have liked this product and how well this product is selling till date.  Following sorting can be applied using this attribute:   * Above 2 * Below 2 |
| Tax | This attribute will give us the detail that apart from the price of the product how much tax the user has to pay in dollars. This data type is going to be **float** again. Its sorting can be considered the dame as in rating. |
| Store Name | This is going to be the name of the store that has collaborated with the website to provide his services and earn a handsome amount of money out of it. It is also going to be the in the **string** data type. The sorting paradigm are going to be the same as name of the product.   * A-Z * Z-A |
| Top Selling | This is the attribute that will give the detail about the product of it being one of the top selling products of the month. This is also going to be in the **String** data type but in this regard the sorting paradigm are going to change. The following sorting options will be provided to the user:   * Yes * No |
| **Sources** | The sources from which data is scrapped |
| Websites | Ali-Express and Daraz.pk |
| Website Links | * In case of Ali Express:   <https://www.aliexpress.com/>   * In case of Daraz.pk   https://www.daraz.pk/ |
| Daraz Categories | * Electronic Devices * Electronic Accessories * TV and Home Appliances * Health and Beauty * Babies and Toys * Groceries and Pets * Home and Life Style * Women’s Fashion * Men’s Fashion * Watches, Bags and Jewelries * Sports and Outdoors * Automotive and Motorbikes |
| Ali Express Categories | * Women’s Fashion * Men’s Fashion * Computer Office and Security * Home improvement and Tools * Bags and shoes * Home Appliances |

**Algorithm Details:**

|  |  |
| --- | --- |
| Algorithm Name | Insertion Sort |
| Description | It is an efficient algorithm for sorting small number of elements. The process is carried out as we do in sorting the playing cards. We start from one element ahead than the element at the first index and then we compare it to the previous value/s unless its either the start of the array or the element is less than the element before the element. In this way moving forward we keep doing this and checking and swapping the elements unless we get our desired sorted array which is checked by the end of the array. |
| Pseudo Code | As per defined in the above description. The pseudo code to this algorithm can be defined as:  For forward = 2 to len:  Backward = forward – 1  KeyToCheck = Array[forward]  While backward >=0 and Array[backward] > keyToCheck:  Array[backward +1] = Array[backward]  Backward = backward – 1  End While  Array[i+1] = keyToCheck  End for  If we consider that Array of length len then the sorting process can be carried out as:  Algorithm: 1Psuedo Code      *Fig: Pseudo code for Insertion Sort* |
| Code | This function will simply get an array as a parameter to be sorted and then perform the algorithm of insertion to sort this particular unsorted array.  def InsertionSort(arr):  size = len(arr)  for j in range(1, len(arr)):  key = arr[j] # vAlue to be added inn the Sorted PArt  i = j - 1  while i >= 0 and arr[i] > key:  arr[i + 1] = arr[i]  i = i - 1  # print(i+1)  arr[i + 1] = key  *Fig: Python code of Insertion sort* |
| Analysis | Best Case: Ω (n)  Intermediate: Θ(C:\Users\DEll\AppData\Local\Temp\ksohtml\wps5996.tmp.jpg)  Worst case: O(C:\Users\DEll\AppData\Local\Temp\ksohtml\wps5997.tmp.jpg) |
| Correctness | We will prove the correctness of Insertion sort using loop invariants.  Prior to the first iteration the value of forward = 1 or key value = Array [1] which means that the list has only one element at that point which is already sorted. So it holds True and **Initialization** is also true.  Similarly another invariant will also be true for while loop as it will hold only one value at a time which is itself and will be present at its original position. Since the first data is copied in the keyToCheck the second loop never destroys or eliminates the data as it is available at the end of the loop ad is placed at its original position.  In **termination** process the for loop condition is checked which tells us that when the loop is to be stopped. Since it depend on the value of forward to be greater than the length of the array length. So we have to keep check on the forward value. Since it is always being incremented on each iteration which means a time will come when the value will reach to a position where the condition will be false and the loop ends.  Hence this algorithm is correct. |
| Strengths | * It is highly efficient for small set of data and is also one of the stable sorting algorithms * One of the advantage is that if the array is partially sorted then it will reduce the number of steps for the operations. In other words it is adaptive to the list. * It has much less space complexity than other sorting algorithms. |
| Weakness | * It does not deal with the huge list as well as other sorting algorithms deals with it. * The worst case time complexity is O(C:\Users\DEll\AppData\Local\Temp\ksohtml\wps5998.tmp.jpg) which is not considered a good practice when dealing with large data as compared to O(nlgn) algorithms. * In order to swap a single element we have to move a large set of elements from one position to other |
| Dry Run | Let us consider that following is the input of the list that user has committed and wants to sort this list A = {5, 2, 4, 6, 1, 3}:   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 5 | 2 | 4 | 6 | 1 | 3 |   Here the j will first contain the value 1 as index for itself at which the value present is 2 which will act as key in this case. And I will hold the value 0 at which right now present value is 5. Since I is greater than -1 so the loop will not break and index at i+1 that is at 1 we will place 5 and decrement the value of I and then the loop will break. And the last statement will evaluate replacing the value of index i+1 with the key. Which means we place at index 0 we will place 2. The new Array will be:   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 2 | 5 | 4 | 6 | 1 | 3 |   Now the value of j will be 2 and key will become 4 and now I becomes 1. The loop will continue because 5is greater than key and I > -1. And 5 will be swapped with the value 4 and the new array will become like:   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 2 | 4 | 5 | 6 | 1 | 3 |   In this index there is no need of swapping because there is one condition false that is the value at I>key which is false in this case so there will be no swapping and we will move to the next index.   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 2 | 4 | 5 | 6 | 1 | 3 |   In this index we will have to run this process 4 times till the value reaches its correct position and in each iteration the value of I will be decremented by 1. And at the end the value at index i+1 will be replaced by key value which is in this case 1. The sorted array will be array as follows after this step:   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 1 | 2 | 4 | 5 | 6 | 3 |   The key value is updated to 3 and now again the inner loop will be executed 3 times until the value reaches its original positon. After sorting this key value the outer loop will run out of bound and will break the loop and the user will get the sorted array which is as follows:   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 1 | 2 | 3 | 4 | 5 | 6 | |
|  |  |
| Algorithm Name | Merge Sort |
| Description | Merge sort is a divide and conquer approach of recursive algorithms to sort the given length of arrays. It is a process in which we divide the given arrays into two sub arrays each time and keep on doing it until we reach a single element. Once we do so we then call the merge function. The purpose of the merge function is to check for the value all alone and decide which values are going to be kept first. In it we need two functions one of them is going to follow the divide approach and the other one will follow the conquer approach. The divide one will call the conquer function. The merge function will contain two arrays. One will be lft and one will be right and other than this will make a third array for our help in which we are going to evaluating value separately. The function will check both left and right sub arrays and decide which of the element is of small order and will be kept first in the array on the lower order and then increment the index of the array from which the value was taken and similarly the index of the value placed others will remain same and recursively this process goes off until we run out of elements. |
| Pseudo Code | This is the function which will be called off the main function and is going to divide the arrays into two sub arrays until the last element is found and is called recursively until then  MERGE\_SORT(Array, p, r):  If p<r:  q = [p+(r-p)]/2  MERGE\_SORT(Array, p, q)  MERGE\_SORT(Array, q+1, r)  MERGE(Array, p, q, r)  *Fig: Pseudo code of the Merge Sort*  Now the function that will perform the original operation of checking the values and then the sorting on behalf of that value is :  MERGE(Array, p, q, r):  Num1 = q-p+1  Num2 = r – q  Let Left = [num1+1] and Right = [num2 +1]  For i in 1 to num1:  Left[i] = Array[p+i-1]  For j = 1 to num2:  Right[j] = A[q+j]    Left [num1 + 1] = -999  Right[num2 + 1] = -999  i = 1  j = 1    for k = p to r:  if (Left[i] <= Right[j]):  Array[k] = Left[i]  I = i+1  Else if (Array[j] = y[k] == right[j]):  J=j+1    *Fig: Pseudo code of the Merge function* |
| Code | The function of **MERGE\_SORT** can be defined in the code as :    *Fig: Python code for Merge Sort*  def mergeSort(arr,l,r):  if r>l:  m = l+(r-l)//2  mergeSort(arr,l,m)  mergeSort(arr,m+1,r)  merge(arr,l,m,r)  Also the function of **MERGE**  can be defined as:  *Fig: Python code for Merge Function*  def merge(arr,l,m,r):  helper = []  for i in range (l,r+1):  helper.append(arr[i])  i=0  j= m+1-l  k=l  while i<=m-l and j<=r-l:  # if i>=m:  # break  #  # if j>=r:  # break  if helper[i] <= helper[j]:  arr[k] = helper[i]  i+=1  else :  arr[k] = helper[j]  j+=1  k+=1  # print(i, j)  while i<=m-l:  arr[k] = helper[i]  i+=1  k+=1  while j<=r-l:  arr[k] = helper[j]  j+=1  k+=1 |
| Analysis | Best Case: Ω (nlgn)  Intermediate Case: Θ(nlgn)  Worst Case: O(nlgn) |
| Correctness | The correctness can be checked by verifying each and every step of the function recursively. First we need to prove the correctness of the merge function as it is iterative. Again we will take the help of the loop invariants to prove the correctness of this proof. For this we have to take care of the temporary array. For one iteration of k we will be dealing with two indices of the array that are I and j. In such case our loop invariant is going to be temp [k] <= array [1, 2… m]. and temp[k] <= array[j, … q].What we want to say is that we are simply copying the minimum of the remaining elements in the array. Once the temp array is filled from left to right then all the elements left are later on added to the array. It means that after iteration the array will be sorted between p and q only. Since array [i] <= array [i+1, .. m] and array[j] <= array[j+1,..q]. since array[i] and array[j] are the smallest elements off the array so the complement invariant is true.  The other part can be proved by the use of **Mathematical Induction.** If we suppose that we call a merge-sort on an array of size n. Then It will recursively tries to call this method on two arrays of size n/2. For inductive hypothesis we assume that the call will sorts the arrays correctly. Since we have already proved that the mere works correctly. Hence the array will be sorted correctly between indices p and q hence it proves our hypothesis |
| Strengths | * It is quicker for sorting even a large amount of the data list * It is one of the stable sorting algorithms available * It has consistent running time. |
| Weaknesses | * It uses a lot of memory space extra approximately equal to N * Goes through the whole process even if the list is completely sorted. * It is less efficient than other sorting algorithms. |
| Dry Run | Let the user has inputted the array A={9,7,5,2,1} and wants to sort this using merge sort and this can be explained using recursive call and is given below:   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 9 | 7 | 5 | 2 | 1 |   First we select the complete array an then divide into two parts in this case it can be done as left array and right array will be :   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | Left part   |  |  |  | | --- | --- | --- | | 9 | 7 | 5 | | Right Part   |  |  | | --- | --- | | 2 | 1 | |   In this process again merge sort will be called on the left array and agin we will obtain two arrays   |  |  |  |  |  | | --- | --- | --- | --- | --- | | Left part   |  |  | | --- | --- | | 9 | 7 | | Right Part   |  | | --- | | 5 | |   Now again the Merge sort will be called off on to the two arrays firstly on left array and is going to be :   |  |  |  |  | | --- | --- | --- | --- | | Left part   |  | | --- | | 9 | | Right Part   |  | | --- | | 7 | |   Since the array of length needs no sorting which means this array is ready for the merge function to be called on it. Merge the selected arrays in reverse and sorted order. Select the minimum of the two values and then add it to the selected array of above note. When one of the list becomes empty copy all of the value of the other array into this noted list.   |  |  | | --- | --- | | 7 | 9 |   Array of the length 1 needs no sorting again it is ready for the merge function therefore again we check for the smallest value from the starting index of the bot left and right list and place it first in the sorted noted list.  And the process goes on. Until we obtain the complete sorted array.   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 1 | 2 | 5 | 7 | 9 | |
|  |  |
| Algorithm Name | Selection Sort |
| Description | Selection sort is one of the stable in-place comparison sorting algorithms. In this we have a key value index and then we iterate that value through the next index to check if there is any value that is smaller than that value if there exist such a value then we simply swap it other wise that is considered the smallest value. |
| Pseudo Code | *Fig: Pseudo code for Selection sort*  Selection-Sort (Array):  N = length (Array)  For I = 1 to n-1:  Min = i  For j = i+1 to n  If Array[j] < Array[min]:  Min = j  End if  End inner for  If min != i:  Swap Array[min] with Array[i]  End if  End for |
| Code | The simplest code for this process is as follows:  def SortingFunction(list):  n = len(list)  for i in range(0,n-1):  min = i  for j in range(i+1,n):  if list[j]< list[min]:  min = j  if min != i:  list[min],list[i] = list[i],list[min]  *Fig: Python code for Selection Sort* |
| Analysis | Best Case: Ω (C:\Users\DEll\AppData\Local\Temp\ksohtml\wps59AA.tmp.jpg)  Intermediate: Θ(C:\Users\DEll\AppData\Local\Temp\ksohtml\wps59AB.tmp.jpg)  Worst case: O(C:\Users\DEll\AppData\Local\Temp\ksohtml\wps59AC.tmp.jpg) |
| Strengths | * The main advantage is it performs very well on small list * It in-place sorting algorithm so space usage is minimal * It performs well on items that have already been sorted. |
| Weaknesses | * It scans all the elements even if the list is sorted * Poorly efficient when dealing with large data. * It requires n squared steps to sort n length array. |
| Dry Run | |  |  |  |  |  | | --- | --- | --- | --- | --- | | 14 | 33 | 27 | 10 | 35 |   First we take the first value as the key value and then we look for the list for a value smaller than the key value. Once we encounter the value that is 10 we shift the min index to 10 and then we look for more. Once the loop ends since the min index is not of value I that is 14 so we replace the ith value with jth value.   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 10 | 33 | 27 | 14 | 35 |     Then we take key value of i+1 index that is 33 and then again we move forward and look for the value smaller than it. Since 27 is smaller than 33 then again it will set value to min. But 14 is also smaller than 27 so again min index will be 14 value index. After the termination of the loop 33 is replaced with 14.   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 10 | 14 | 27 | 33 | 35 |   Since for the further values min index is the current index so it is already the sorted array. |
|  |  |
| Algorithm Name | Bubble Sort |
| Description | The other name of this algorithm is sinking algorithm. It simply and iteratively lists through each and every element of the list and then compares them and swaps them if they are placed in a wrong order. There are two loops required for this process and a Boolean variable which tells us that whether the given array is being sorted or not. Once the Boolean variable it false the list is completely sorted and we simply call off the function to terminate. |
| Pseudo Code | *Fig: Pseudo code for Bubble sort*  Bubble Sort(Array):  n = length (list)  for i = 0 to n-1:  swapped = false  for j = 0 to n-1 do:  if list[j] > list[j+1]:  swap( list[j], list[j+1] )  swapped = true  end if  end for  if(!swapped):  break  end if  end for |
| Code | The Simplest code for Bubble sort is :  def BubbleFunction(arr):  size = len(arr)  for i in range(0, size - 1):  swapped = False  for j in range(0, size - 1):  if arr[j]>arr[j+1]:  list[j],list[j+1] = list[j+1],list[j]  swapped = True  if (not swapped):  break |
| Analysis | Best Case: Ω (n)  Intermediate: Θ(C:\Users\DEll\AppData\Local\Temp\ksohtml\wps59AE.tmp.jpg)  Worst case: O(C:\Users\DEll\AppData\Local\Temp\ksohtml\wps59AF.tmp.jpg) |
| Strengths | * It is one of the popular and easy to use sorting algorithms * No additional memory is used in this case * Code is very simple and takes only few lines and time to write and implement it. |

|  |  |
| --- | --- |
| Weaknesses | * It does deal well with the list containing maximum number of elements. * Requires n squared processing steps to sort n elements * It is suitable only for teaching but not for real life. |
| Dry run | Let user has given us the following input to sort the array A={9,8,7,6,5}.   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 9 | 8 | 7 | 6 | 5 |   In this sorting we constantly move from left to right comparing each and every adjacent values and swapping on each case if needed. First we compare the first two value since the 9 is greater than 8 we swap both of them.   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 8 | 9 | 7 | 6 | 5 |   Now again comparing the next two values. Since the value 9 is again greater than 7 we again swap them ad this whole process continues and finally a time comes when there will be no need to swap the value =s and that will be the time when the values will be at their original position and we get a sorted array   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 5 | 6 | 7 | 8 | 9 | |
|  |  |
| Algorithm Name | Quick Sort |
| Description | Quick sort is also like the merge sort and it also applies the divide and conquer technique for sorting of the data. The main function of the quick sort is to pick element as a pivot and then divide the list into two halfs based on that pivot point such that the elements less than pivot are on left and others are on right. Well keeping it mind that it is also an in place sorting algorithm. It partitions an array and then calls itself to sort the resulting subarrays |
| Pseudo Code | F*ig: Pseudo code for Quick Sort*  quickSort(arr, low, high):  if (low < high):  pi = partition(arr, low, high)  quickSort(arr, low, pi - 1)  quickSort(arr, pi + 1, high)  The partition function can be defined as :  *Fig: Pseudo Code for the Partition method*  partition (arr[], low, high):  pivot = arr[high]  i = (low – 1)  for (j = low to high- 1)  i++  swap arr[i] and arr[j]  swap arr[i + 1] and arr[high])  return (i + 1) |
| Code | The simplest code for the Quick sort function is :  def quickSort(Arr,low,high):  if low<high:  pi = partition(Arr,low,high)  quickSort(arr,low,pi-1)  quickSort(arr,pi+1,high)  *Fig: Function of Quick sort*  Now the partion function must also be defined which is given below:  def partition(Arr,low,high):  pivot = Arr[high]  i = low-1  for j in range(low,high):  if Arr[j] < pivot:  i+=1  Arr[i],Arr[j] = Arr[j],Arr[i]  Arr[i+1],Arr[high] = Arr[high],Arr[i+1]  return i+1  *Fig: Python Code for Partition Function* |
| Analysis | Best Case: Ω (nlgn)  Intermediate Case: Θ(nlgn)  Worst Case: O(C:\Users\DEll\AppData\Local\Temp\ksohtml\wps59C1.tmp.jpg) |
| Strengths | * It can be implemented as an in place sorting algorithm and uses only the auxiliary stack * Its memory usage is minimal * It has an extremely short inner loop |
| Weaknesses | * Its worst case performance is similar to average performance of any sorting algorithms * It is fragile a simple mistake may lead to a completely different result. * If recursion is not available the implementation is near to impossible |
| Dry run | Let the inputted array be A={2,8,1,7}   |  |  |  |  | | --- | --- | --- | --- | | 2 | 8 | 1 | 7 |   In this case for partitioning we will have the key as 7 and then we will enter the loop by taking value as first index.Here the I value will be first index -1 which is -1. Now we compare it with the partiton value since it is less than the key value so we increment in the value of I that is 0. And then we place the the value 2 at 0th index which is already there. And then we move to the next index here the value is 8. Since it does no satisfy any kind of relation so it simply passes the value.  Now the current index will hold value 1 since it is less than the key value so it simply increment tha value of I which becomes 1 and now replaces the value at I with value at j and the array becomes.   |  |  |  |  | | --- | --- | --- | --- | | 2 | 1 | 8 | 7 |   Now here the first loop breaks and it then increments the value of I which becomes 2 now and place he key value there by swapping.   |  |  |  |  | | --- | --- | --- | --- | | 2 | 1 | 7 | 8 |   Now it will be called on left array and right array separately and will sort the array partitionwise. The resultant sorted array will look like:   |  |  |  |  | | --- | --- | --- | --- | | 1 | 2 | 7 | 8 | |
|  |  |
| Algorithm Name | Counting sort |
| Description | It is an efficient sorting technique for the key in a specific range. It works as the counting of the distinct elements in the list. It is carried out by making a separate list and then checking for the distinct elements in the list. And then calculating the cumulative frequency of the list. And then using this cumulative frequency of the list we are going to put the values in the list. |
| Pseudo Code | *Fig: Pseudo Code for Counting sort*  CountingSort(input) :  k = range of elements  count = [k + 1]\*0  output = [length (input)]  **for** i = 0 **to** length(input) – 1:  j = key(input[i])  count[j] += 1  **for** i = 1 **to** k :  count[i] += count[i - 1]  **for** i = length(input) - 1 **down to** 0 **:**  j = key(input[i])  count[j] -= 1  output[count[j]] = input[i]  **return** output |
| Code | The simplest code for this function is  def CountingSort(arr1):  minimum = min(arr1)  k = (max(arr1) - minimum)+1  count = [0]\*(k)  output = [0]\*len(arr1)  for i in range(len(arr1)):  j =arr1[i]  count[j-minimum] += 1  for i in range(1,k):  count[i] += count[i-1]  for i in range(len(arr1)-1,-1,-1):  j = arr1[i]  count[j-minimum] -=1  output[count[j-minimum]] = arr1[i]  return output  *Fig: Python Code for Counting Sort* |
| Analysis | Best Case: Ω (n+k)  Intermediate Case: Θ(n+k)  Worst Case: O(n+k) |
| Strengths | * Counting sort has lesser time complexity when range is comparable to number of inputs. * The major advantage of this algorithm is its complexity * It is not a comparison based sorting technique |
| Weaknesses | * Counting sort can only be used with integer values only because otherwise the frequency indices cannot be made. * If non primitive array is sorted then an additional array is required for storing and sorting data. * Counting sort has large space complexity |
| Dry run | |  |  |  |  |  | | --- | --- | --- | --- | --- | | 17 | 44 | 86 | 54 | 11 |   First we take the first value as the key value and then we look for the list for a value smaller than the key value. Once we encounter the value that is 11 we shift the min index to 10 and then we look for more. Once the loop ends since the min index is not of value I that is 17 so we replace the ith value with jth value.   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 11 | 44 | 86 | 54 | 17 |     Then we take key value of i+1 index that is 54 and then again we move forward and look for the value smaller than it. Since 44 is smaller than 33 then again it will set value to min. But 17 is also smaller than 27 so again min index will be 17 value index. After the termination of the loop 33 is replaced with 17.   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 10 | 17 | 44 | 54 | 86 |   Since for the further values min index is the current index so it is already the sorted array. |
|  |  |
| Algorithm Name | Radix Sort |
| Description | The main idea of this sorting algorithm is to sort data digit by digit starting from the least significant one to most significant one. It also needs counting sort for sorting the list on each index. What it is going to do is to make a subarray on each index depending on the digit by digit evaluation and then we are going to apply counting sort on this index. The whole process is performed to the maximum digit count. |
| Pseudo code | *Fig: Pseudo code for Radix Sort*  RadixSort(Array,digits):  I=0  While i<digits:  Make 10 buckets for data    For j = 1 to A.size:  Bucket.append(value)  Apply sorting on each index  I+=1 |
| Code | The simplest code is as follows:  def findElement(value,digit):  digit = digit+1  output = 0  for i in range(digit):  output = value%10  value = value//10  return output  def RadixSort(array,digits):  i=0  while i<digits:  buckets = []  for \_ in range(10):  buckets.append([])  for j in range(0,len(array)):  value = findElement(array[j],i)  buckets[value].append(array[j])  index = 0  for ind in range(0,len(buckets)):  arr = buckets[ind]  for inde in range(0,len(arr)):  array[index] = buckets[ind][inde]  index += 1  i+=1 |
| Analysis | Best Case: Ω (nk)  Intermediate Case: Θ(nk)  Worst Case: O(nk) |
| Strengths | * It is fast when the range of array elemets is less * Radix sort is stable as the relative order of element is always maintained. * It is a non-comparison based sorting algorithm |
| Weaknesses | * It is much less flexible than other sorting algorithms because it depends on digits * The constant for radix is greater as compared to others. * It required more space or space complexity is greater |
| Dry Run | The whole procedure is based on the counting of the digits. Once we get the maximum digits of a number we can easily evaluate the data sorted list like the inputted list A={110,85,21,6}   |  |  |  |  | | --- | --- | --- | --- | | 110 | 85 | 21 | 6 |   The loop will take mod with respect to the 10 to power of key digits number. And then we put the value in the bucket in the first run we will get values like:   |  |  |  |  | | --- | --- | --- | --- | | 110 | 21 | 85 | 6 |   Then the second digit will be counted and again the data will be accumulated in the list applying the sorts there we get:   |  |  |  |  | | --- | --- | --- | --- | | 6 | 110 | 21 | 85 |   Now considering the third and most significant bit of the data we can get the final result of the sorted array that is:   |  |  |  |  | | --- | --- | --- | --- | | 6 | 21 | 85 | 110 | |
|  |  |
| Algorithm Name | Bucket sort |
| Description | Bucket sort is mainly useful when the input is mainly distributed over a range. The range decided for its application is [0,1). In this process we separate the values as an absolute by taking the floor of the multiplication of the array element with the size of the array. And then adding it to the best possible index obtained through the floor method. After that we apply the insertion sort on each index list to obtain the sorted indexed list. After that based on sorting we will simply remove the value and place it to its original position. |
| Pseudo Code | *Fig: Pseudo code for Bucket Sort*  bucketSort(arr[], n)  Create n empty buckets  Do following for every array element arr[i].  Insert arr[i] into bucket[n\*array[i]]  Sort individual buckets using insertion sort.  Concatenate all sorted buckets. |
| Code | The simplest code for this process is as follows:  def BucketSort(arr,size):  bucket = []  for \_ in range(size):  bucket.append([])  for i in range(0,size):  n = math.floor(arr[i]\*size)  bucket[n].append(arr[i])  for i in range(0,len(bucket)):  # array = bucket[i]  InsertionSort(bucket[i])  index = 0  for i in range(0,len(bucket)):  array = bucket[i]  for j in range(0,len(array)):  arr[index] = bucket[i][j]  index += 1  *Fig: Python code for Bucket Sort* |
| Analysis | Best Case: Ω (n+k)  Intermediate Case: Θ(n+k)  Worst Case: O(n^2) |
| Strengths | * It allows each bucket to proceed independently so there is very less need to sort al the buckets * It can be used as an external sorting algorithm |
| Weaknesses | * It is only applicable to within range value but not to all data types. * If input distribution is uneven then this algorithm slows down * The worst case time complexity is n^2 which is not considered good as compared to other stable algorithms. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Dry run | In this we have to multiply the element with the size ro get a digit to be put in the basket. If we have an array like A={0.897,0.656,0.565,0.1234}  Then after the application of the whole process we get a sorted array like:   |  |  |  |  | | --- | --- | --- | --- | | 0.1234 | 0.565 | 0.656 | 0.897 | |

**Resource Management:**

|  |  |
| --- | --- |
| **Planning** |  |
| Task Division | As per planned schedule, Following the stages we had divided the tasks equally among us. Such as in the first stage we had meeting regarding the proposal and we both searched for multiple website and read articles regarding the best and suitable website for scrapping. In the second Task we had divided the algorithms between us and also we were looking up for the mistakes in each other’s work so that we could provide with the best possible report to the professor regarding the algorithms. Again next while making the user interface we both were consulting each other and decided at each step that what should be the fields or button that should be provided to the user so as to provide the user with best possible and easeful GUI. |
| Collaboration | We both members held several meetings at multiple places and spent hours together working for this project not only as person but also on **Teams** as well which was our online Tool for communication |
| **Integration** |  |
| Problems | We faced multiple problems while integrating multiple components of the program and making them work as a single unit however we were able to do so at the end. One of the common problem that we faced was to use the threading in order to be able to stop the program with the user demand as well as other operation should be performed at the same time like the amount data being scrapped need to be print in the list, Also when the pause button is pressed it means that we not only have to stop the scrapping process but also the data needs to be displayed that has been scrapped and also the percentage of the data scrapped. Not only this we were also dealing with the progress bar so depending upon the data obtained we had to change the value of the progress bar min and max value. |
| Lead | In both of the members the leading member was **Qazi Maaz Sajjad (2020-CS-63)** who was a holding force in this work and was leading the project not only in the consultancy part but in coding as well. |

**Graphical User Interface:**

|  |  |
| --- | --- |
| Pencil Tool | **F*ig-1:*** *The Complete UI Prototype of the project*  ***Fig-2****: Sorting for each column* |
| Qt-Design | *Fig: Python Qt-Design of Project* |
| **Component** | Components of UI are as follows |
| Algorithms | It is going to be drop down combo box that allows the user to get the desired algorithm to carry out the sorting. If the user does no choose any of the algorithm then it simply chooses **Merge Sort** as a by default sorting algorithm. |
| Sort By | This is a also a drop down menu using the combo box for user to choose the parameter on the basis of which the user want us to sort the data. |
| Start | It is button for starting the process of scrapping. Once it is pressed the scrapping process at the backend starts and it will start displaying data on to the table |
| Pause Scrapping | It is going to be a check box. If the user wants to stop the process of scrapping then it will simply click on this check box and the process will be stopped at the back end. |
| Stop | It is also a push button for stopping the process completely and then closing the system completely |
| Multilevel Sorting | It is a group box which contains multiple check boxes which are going to be used when the user wants to apply multi-level sorting. |
| Search By | It is again a group box and it includes two radio buttons and two text boxes and they are going to be used in the searching process. |
| Table | It is table widget and is going to help us to display the data that has been scrapped yet. |
| Progress Bar | It is a progress bar which will display the rate of which the data has been scrapped. |
| Theme |  |
| Light Theme | *Fig: Light Theme of the Project GUI* |

**User Manual:**

|  |  |
| --- | --- |
| **Element Name** | Algorithm |
| Detail | This is a drop down menu for you. The purpose of this menu is to allow you to choose that which algorithm you want for sorting the data. By choosing it the default algorithm will be removed and program will use the algorithm that the user has mentioned by clicking on one of the options provided by the menu. |
| Snapshot | *Fig: Algorithm Options* |
| On Click Operation | Once the user clicks on one of the options then the value is set as a standard for sorting a huge amount of data that has been obtained after scrapping. Once the button is clicked then the sorting is applied on to the data taking name as a by default parameter and the data is changed in the table in the sorted order. |
| Before Sorting | *Fig: Before sorting view of data in table* |
| After Sorting | *Fig: After Sorting view of the data In table* |
| **Element Name** | Sort By |
| Details | This is a first level sorting drop down parameter menu. By clicking on this the user will be provided with a list of attributes using which we can apply sorting on the whole bunch of data. It’s also a combo box based menu. Here the most important thing to remember is that it will not allow multilevel sorting here. Only the single level sorting will be applied here. |
| Snapshot | *Fig: Sorting Attributes Drop Down Menu* |
| On-click Operation | When the user click on the drop down menu, a list of options will appear on the screen. By hovering over the menus the user can check that which attribute user wants for making a standard while performing sorting at single level. When it clicks on one of the options the value is set on the drop down menu and also at the backend where it will be considered the standard while sorting.    *Fig: Value Selected for sorting attribute* |
| Element Name | Start Button |
| Description | The start button is one of the most important element in the complete project. One of the most important operation is going to be performed using this button. This button will start the scrapping process and once it is pressed the data will start to accumulate in the table. The button is a green colored button located in the left frame of the project panel. |
| Snapshot | *Fig: Start Button* |
| On-Click Operation | Once the button is pressed the program will start its working of scrapping data off the website and accumulating it in its bin for its personal use. As the data keeps on adding in the list the table keeps on updating the value of the rows and the data is displayed like:      *Fig: Data Scrapped Displayed in table*  Not only this the program will also keep note of the data remaining and will also notify the user about the details with the help of a progress bar that will constantly be running throughout the procedure and will update its value on each value being added to the list.  *Fig: Progress bar displaying the progress of scrapped data* |
| Element Name | Pause Scrapping |
| Description | It’s a check box which is located at the left panel of the project’s frame and is a orange colored text. The purpose of this checkbox is if it is checked then it will pause the process and will display the detail of the data obtained from the scrapping. |
| Snapshot | Fig: Pause Check-Box |
| On-click Operation | Once this system is enabled which means when the user click on the check box then it will be checked and the code behind it will be enabled. The scrapping will be stopped and the index of the last stored data will be saved so that when we want it will resume. So when it is enabled the data is stopped and the sorting can be performed if the user wants. During the time it is enabled the progress bar is stopped and the reading are printed in the text fields.    *Fig: Pausing Phase of scrapping*  Once the check box is disabled then again the system will erase the values and will resume the data scrapping from the point where it was paused.      *Fig: Resuming Phase of scrapping* |
| **Element Name** | Stop Button |
| Description | It is also a push button which located in the center of the left panel of the project’s window. It a red colored button indicating that it will end the process or everything that is happening currently |
| Snapshot | *Fig: Stop Button* |
| On-Click Operation | Once the button is pressed by the user the program will automatically stop everything and will close the system and exit to the window. |
| **Element Name** | Multi-level Sorting |
| Description | This option is also available in the left panel of the project frame and is located at the bottom of the left panel under a group box with the heading Multi-level sorting. It contains check boxes which holds a particular parameter name. Clicking on the name will mean that the user wants it to be considered the standard while sorting. |
| Snapshot | *Fig: Multi-Level sorting Menu* |
| On-Click Operation | When the user clicks on one of the attribute say “Name” then the name is considered a standard and using it the sorting operation is performed on the values. And the whole list is updated.  *Fig: Name as a Standard*  Now if the user wants that the name has sorted the data now I want the products to be sorted in such a way that the data remains sorted with name and then the product with low prices with name starting with character ‘A’ are sorted and then with character ‘B’ and so on. Then the user will check the price attribute as well and resultantly the data will be displayed in the table in such a way that the product starting with character ‘A’ and lowest price comes first and then with the character ‘A’ and price greater than the previous one comes and so on. Similarly for other characters as well.  *Fig: Multiple attributes selected for sorting* |
| **Element Name** | Searching |
| Description | If the user wants to search something then this system provides the user with the two options. It’s either the name or the price. Other than this this paradigm is located at the top of the right panel of the window. It contains two radio buttons and two text boxes and in which the input is taken for searching text. |
| Snapshot | *Fig: Searching Box* |
| On-Click Operation | When the user want to search by the name of the product then he simply clicks on the Name radio Button and then the text box in front of the radio button will be enabled and the other one will be disabled so that user won’t be able to enter the data in the below one and then user will get to enter the product name. The system will display al list of all the data containing the word inputted by the user in the table.    *Fig: Searching by Name*  Similarly if the user wants to search the product through a range of price then the system will display the list of all the products that are less than or equal to the value entered inside the text field and the data is displayed in the table.    *Fig: Searching by Price* |