  CS261-Data Structure and Algorithms

                                          Mid Project Proposal (Fall 2021)

**CURE VIBE**

**Project Supervisor**

Mr. Samyan Qayyum Wahla

**Project Members (G18)**

|  |  |
| --- | --- |
| Amna | 2020-CS-123 |
| Mahnoor | 2020-CS-146 |

Department of Computer Science

         University of Engineering and Technology, Lahore, Pakistan

TABLE OF CONTENTS

Contents

[1 Proposer Details: 7](#_Toc86984123)

[1.1 Group Number 7](#_Toc86984124)

[1.2 Registration number of Group Members 7](#_Toc86984125)

[2 Proposal Details: 7](#_Toc86984126)

[2.1 Proposed Project Title 7](#_Toc86984127)

[2.2 Executive Summary 7](#_Toc86984128)

[3 Business Case: 9](#_Toc86984129)

[3.1 Business need for Project 9](#_Toc86984130)

[3.2 End user of the Product 9](#_Toc86984131)

[3.3 Motivation for Project 9](#_Toc86984132)

[3.4 Level of Impact 9](#_Toc86984133)

[3.5 Implications of not Proceeding 10](#_Toc86984134)

[4 Technical Details: 11](#_Toc86984135)

[4.1 Name of Entity 11](#_Toc86984136)

[4.2 Attributes of Entity 11](#_Toc86984137)

[4.3 Sample of Scrapping Scope 13](#_Toc86984138)

[4.4 GitHub Repository Link 14](#_Toc86984139)

[4.5 Sorting Algorithms 14](#_Toc86984140)

[5 Algorithms 14](#_Toc86984141)

[5.1 Algorithm Name 14](#_Toc86984142)

[5.2 Description 14](#_Toc86984143)

[5.3 Searching Algorithms 16](#_Toc86984144)

[5.4 Searching Filters 16](#_Toc86984145)

[5.5 Multi-Level Sorting 17](#_Toc86984146)

[5.6 other features 17](#_Toc86984147)

[6 Interface 17](#_Toc86984148)

[6.1 UI Component Name 17](#_Toc86984149)

[6.2 Type of UI component 17](#_Toc86984150)

[6.3 Purpose of UI Component/Other details 17](#_Toc86984151)

[7 Algorithm Details 18](#_Toc86984152)

[7.1 Insertion Sort 19](#_Toc86984153)

[7.1.1 Description 19](#_Toc86984154)

[7.1.2 Pseudo Code 21](#_Toc86984155)

[7.1.3 Python Code 21](#_Toc86984156)

[7.1.4 Time Complexity Analysis 21](#_Toc86984157)

[7.1.5 Proof of Correctness 22](#_Toc86984158)

[7.1.6 Strengths 23](#_Toc86984159)

[7.1.7 Weaknesses 23](#_Toc86984160)

[7.1.8 Dry Run 23](#_Toc86984161)

[7.2 Merge Sort 25](#_Toc86984162)

[7.2.1 Description 25](#_Toc86984163)

[7.2.2 Pseudo Code 27](#_Toc86984164)

[7.2.3 Python Code 27](#_Toc86984165)

[7.2.4 Time Complexity Analysis 28](#_Toc86984166)

[7.2.5 Proof of Correctness 29](#_Toc86984167)

[7.2.6 Strengths 30](#_Toc86984168)

[7.2.7 Weaknesses 30](#_Toc86984169)

[7.2.8 Dry Run 30](#_Toc86984170)

[7.3 Bubble Sort 32](#_Toc86984171)

[7.3.1 Description 32](#_Toc86984172)

[7.3.2 Pseudo Code 34](#_Toc86984173)

[7.3.3 Python Code 34](#_Toc86984174)

[7.3.4 Time Complexity Analysis 34](#_Toc86984175)

[7.3.5 Proof of Correctness 35](#_Toc86984176)

[7.3.6 Strengths 36](#_Toc86984177)

[7.3.7 Weaknesses 36](#_Toc86984178)

[7.3.8 Dry Run 36](#_Toc86984179)

[7.4 Quick Sort 38](#_Toc86984180)

[7.4.1 Description 38](#_Toc86984181)

[7.4.2 Pseudo Code 40](#_Toc86984182)

[7.4.3 Python Code 40](#_Toc86984183)

[7.4.4 Time Complexity Analysis 41](#_Toc86984184)

[7.4.5 Proof of Correctness 42](#_Toc86984185)

[7.4.6 Strengths 43](#_Toc86984186)

[7.4.7 Weaknesses 43](#_Toc86984187)

[7.4.8 Dry Run 43](#_Toc86984188)

[7.5 Heap Sort 45](#_Toc86984189)

[7.5.1 Description 45](#_Toc86984190)

[7.5.2 Pseudo Code 47](#_Toc86984191)

[7.5.3 Python Code 47](#_Toc86984192)

[7.5.4 Time Complexity Analysis 48](#_Toc86984193)

[7.5.5 Proof of Correctness 48](#_Toc86984194)

[7.5.6 Strengths 49](#_Toc86984195)

[7.5.7 Weaknesses 49](#_Toc86984196)

[7.5.8 Dry Run 49](#_Toc86984197)

[7.6 Selection Sort 51](#_Toc86984198)

[7.6.1 Description 51](#_Toc86984199)

[7.6.2 Pseudo Code 53](#_Toc86984200)

[7.6.3 Python Code 53](#_Toc86984201)

[7.6.4 Time Complexity Analysis 54](#_Toc86984202)

[7.6.5 Proof of Correctness 55](#_Toc86984203)

[7.6.6 Strengths 56](#_Toc86984204)

[7.6.7 Weaknesses 56](#_Toc86984205)

[7.6.8 Dry Run 56](#_Toc86984206)

[7.7 Shell Sort 58](#_Toc86984207)

[7.7.1 Description 58](#_Toc86984208)

[7.7.2 Pseudo Code 60](#_Toc86984209)

[7.7.3 Python Code 60](#_Toc86984210)

[7.7.4 Time Complexity Analysis 60](#_Toc86984211)

[7.7.5 Proof of Correctness 61](#_Toc86984212)

[7.7.6 Strengths 62](#_Toc86984213)

[7.7.7 Weaknesses 62](#_Toc86984214)

[7.7.8 Dry Run 62](#_Toc86984215)

[7.8 Tree Sort 64](#_Toc86984216)

[7.8.1 Description 64](#_Toc86984217)

[7.8.2 Pseudo Code 66](#_Toc86984218)

[7.8.3 Python Code 66](#_Toc86984219)

[7.8.4 Time Complexity Analysis 66](#_Toc86984220)

[7.8.5 Proof of Correctness 67](#_Toc86984221)

[7.8.6 Strengths 68](#_Toc86984222)

[7.8.7 Weaknesses 68](#_Toc86984223)

[7.8.8 Dry Run 68](#_Toc86984224)

[7.9 Tim Sort 69](#_Toc86984225)

[7.9.1 Description 69](#_Toc86984226)

[7.9.2 Pseudo Code 71](#_Toc86984227)

[7.9.3 Python Code 71](#_Toc86984228)

[7.9.4 Time Complexity Analysis 73](#_Toc86984229)

[7.9.5 Proof of Correctness 74](#_Toc86984230)

[7.9.6 Strengths 75](#_Toc86984231)

[7.9.7 Weaknesses 75](#_Toc86984232)

[7.9.8 Dry Run 75](#_Toc86984233)

[7.10 Counting Sort 78](#_Toc86984234)

[7.10.1 Description 78](#_Toc86984235)

[7.10.2 Pseudo Code 80](#_Toc86984236)

[7.10.3 Python Code 80](#_Toc86984237)

[7.10.4 Time Complexity Analysis 81](#_Toc86984238)

[7.10.5 Proof of Correctness 82](#_Toc86984239)

[7.10.6 Strengths 83](#_Toc86984240)

[7.10.7 Weaknesses 83](#_Toc86984241)

[7.10.8 Dry Run 83](#_Toc86984242)

[8 INTEGRATION: 85](#_Toc86984243)

[8.1 Scrapping: 85](#_Toc86984244)

[8.2 Difficulties faced: 86](#_Toc86984245)

[8.3 Ideal Source: 87](#_Toc86984246)

[9 Project Requirements: 87](#_Toc86984247)

[10 UI implementation 88](#_Toc86984248)

[10.1 Ideas and Implementation 89](#_Toc86984249)

[10.2 Details of UI 90](#_Toc86984250)

[11 Collaboration 92](#_Toc86984251)

[12 Task Division 92](#_Toc86984252)

[13 Final Application 92](#_Toc86984253)

[13.1 Remaining 94](#_Toc86984254)

[14 Unfinished ideas 94](#_Toc86984255)

# Proposer Details:

|  |  |
| --- | --- |
| Group Number | G18 |
| Registration number of Group Members | 2020-CS-123  2020-CS-146 |

# Proposal Details:

|  |  |
| --- | --- |
| Proposed Project Title | *CURE VIBE* |
| Executive Summary | This project is about pharmacy products and the name of the project is CURE VIBE. We will scrap data from 3-4 different sites and will add them. There will be scrapping of one million items with the 7 attributes. It will only include one product that is medicine. This extend will assist you how? One can purchase solutions through it online and the solutions are in category and each sort of medication is in it and one can indeed discover pharmaceutical through the sort of malady and it makes a difference patients as they can counsel specialists and can enter their medicine their Our main screen will be of login which will include username and password and along with remember me and forget password after entering login through correct credentials next screen will appear of data. It will include bar in which there will be option of pause, start, resume and stop the scrapping data. The user can enter any and the data will be stopped or paused or will resume accordingly. Then this screen will consist of three combo box. In first combo box there will be attributes in which the user will choose the attribute. The attributes are Product name, Product Id, Listed price, Description, Category, Common uses and ingredients. Then in next combo box there will be order whether user wants to run it in ascending order or in descending order. If user enters it in ascending order the data will be shown in ascending order and if user enters on descending order data will be shown in the descending order accordingly. The third combo box will consist of different algorithms and user will choose one of the given algorithm. 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. The third screen will be of search. In search user can search any item by product id and product name and the product with the attributes will be shown in the table. |

# Business Case:

|  |  |
| --- | --- |
| Business need for Project | Managing the customer record, price record of medicines and ingredients record without any human error. Data of almost million medicines can be accessed easily. Description of every medicine mentioned in our project will Facilities the users.  Following businesses can get benefit from our project:   * Medical stores business * Private clinics business * Pharmacies business |
| End user of the Product | Company providing medicines to cure vibe(pharmacy) is the end user in our project. |
| Motivation for Project | The motivation for choosing this specific project named CURE VIBE (Pharmacy) is that this is one of the most commonly used software /application/websites in real life domain. So, this can ease the sorting and searching of medicines for user within few steps. |
| Level of Impact | The level of impact expected in this project is that there will be web scrapping of products from different websites in the range of million which means that we will get the idea to retrieve large amount of data in few steps.  In this project, there is arrangement of data in either ascending or descending order according to the user demand.  Also, we will be able to implement different sorting algorithms like merge sort algorithm, quick sort algorithm, selection sort, insertion sort, hybrid sort, bucket sort etc. which will provide user different options to sort data according to his own choice.  Moreover, our project can synthesize the huge data which is being scrapped from different websites.  . |
| Implications of not Proceeding | The implications that are not being able to proceed include that it cannot be online. There can be no usage to sort data online, or to purchase or sell medicines in real life domain |

# Technical Details:

|  |  |
| --- | --- |
| Name of Entity | MEDICINES |
| Attributes of Entity | |  |  |  | | --- | --- | --- | | Name | Data Type | Description | | Product Name | String | The name of the medicine is in String. It will consist information about medicines name through which we can easily search medicines. | | Quantity | Int | The amount of medicine or tablets in one packing is in integer as it is in number. It will tell the amount of tablets in one medicine. | | Listed Price | Integer | The price of medicine of which it will be sold will be in integer. Through this customer can know the exact price of the medicine. | | Description | String | The description about medicine. The places where it can be used is in string. It will help the patient know about medicine. | | Category | String | The category in which medicine is like it can be in painkillers or for asthma patients is in string. It will help patients find the medicine by the category or they can easily search the medicine just by typing category | | Common Uses | String | The uses of medicine that how it will cure and for which purpose itcan be used is in string. It will help patient know the uses of medicine. | | Ingredients | String | The Things that are used in making of that medicine is in string. | |
| Sample of Scrapping Scope | LINK:  <https://www.1mg.com/>  The categories are given:    When we will open one by one:    One of the following above medicines:    From here we will extract our attributes which are following:   * + Product name   + Quantity   + Listed Price   + Description   + Category   + Common Uses   + Ingredients |
| GitHub Repository Link | https://github.com/Amnaaaaa/CS261F21PID18 |
| Sorting Algorithms | Sorting algorithms in our project are insertion sort, merge sort, bubble sort, selection sort, hybrid sort, counting sort, etc. The brief description of algorithms are given below. |

# Algorithms

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | Algorithm Name | Description | | INSERTION SORT | Insertion sort is a technique for sorting an array in ascending order by comparing each element one by one. | | MERGE SORT | Merge sort is a method of sorting an array that involves three steps:   * Divide * Conquer * Merge. | | BUBBLE SORT | Bubble sort is a sorting algorithm that is used for sorting elements and is one of the simplest sorting algorithms in which elements are sorted by repeatedly swapping elements that are in the wrong order. | | SELECTION SORT | Selection sort algorithm works by finding smallest element from the array and then move it to the first index and so on until the whole array is sorted. | | BUCKET SORT | Bucket sort algorithm sort the array by dividing the elements in form of buckets and then solve them either by applying different algorithms or applying recursively sort algorithm | | COUNTING SORT | Counting sort is an integer sorting algorithm that works by iterating the input until the array is sorted. | | QUICK SORT | Quick sort is also like merge sort which works on divide, conquer and combine. It selects an element as pivot point and then move the elements accordingly until the array is sorted. | | HEAP SORT | Heap sort is more like of a selection sort which divides the array in sorted and unsorted elements then solve the unsorted elements. | | TIMSORT | Tim sort is derived from insertion sort and merge sort. It divides the array into blocks to sort the array. | | TREE SORT | Tree sort algorithm sorts the array by building binary search tree from the elements of the  array which are to be sorted | | |
| Searching Algorithms | **Linear Search:**  It is a method used to find the element in the list and works until the element in the list is searched but if there is no element then the linear search doesn't work and it works according to the algorithm.  **Binary Search:**  It is a method of finding element from the sorted list that divides the list into elements from which element can be found.  **Fibonacci Search:**  Fibonacci search is a method that works by comparing. so, we can say it is a comparison-based method and it uses Fibonacci based elements to Search the element in the list. |
| Searching Filters | **Product Name:**  Name of all medicines will be available can be choose. The name will be in string.    **Quantity:**  The quantity of the medicine needed will be written here in integer.  **Listed Price:**  The price of the medicine at which it will be sold will be written here in integer.  **Description:**  The description of the specific medicine will be written here through which anyone can  know the use of the medicine searched.    **Category:**  The category will tell that medicine is for which purpose whether it's for immunity booster or is some kind of booster.  **Common Uses:**  The common uses if the medicine will be written here in detail  **Ingredients:**  The ingredients of the medicine will be provided in this.  (All above can be arranged in ascending and descending order and can be sorted with 10 algorithms) |
| Multi-Level Sorting | I will provide multilevel searching in between category and listed prices. suppose the products are sorted according to category then we can also subsort them according to prices. |
| other features | Not yet |

# Interface

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | UI Component Name | Type of UI component | Purpose of UI Component/Other details | |  | |  |  | | --- | --- | | Main Heading | Text Field | | User Name | Text Field | | Password | Text Field(Password Field) | | Remember Me | Check Box | | Login | Button | | The purpose of user name is to take input in string of the user.  In the password field, it will take the password from user.  Checkbox remember me will help the user to save his/her password on the application.  Forgot password link will help to reset the password again.  Login button will open the application for the user. | |  | |  |  | | --- | --- | | Attributes | Dropdown | | Order | Dropdown | | Algorithms | Dropdown | | Run | Button | | Exit | Button | | Attributes will give an drop down list of product name, product id, listed price, description etc.  Order will give an option to choose whether user wants to sort data in ascending or descending order.  Algorithms dropdown will show list of different algorithms from which user wants to sort data.  Run button will execute the options, which are being chosen above.  Exit button will exit the screen. | |  | |  |  | | --- | --- | | *Option* | *Dropdown* | | *Enter* | *Input Text Field* | | *Ok* | *Button* | | *Exit* | *Button* | | *Back* | *Button* | | Option drop down will show options to search with either product name or with product id.  Ok button will execute the option selected.  Exit button will exit the screen.  Back button will take the user to the previous screen. | |

# Algorithm Details

|  |  |
| --- | --- |
| Insertion Sort |  |
| Description | Insertion sort is used for sorting array in ascending order by comparing elements one by one with each other. Insertion sort is just like playing cards in which the cards are arranged in an irregular form. In insertion sorting the irregular data is sorted in a regular form.  The procedure is to iterate it to the specified number of numbers and then if the last value is small then the next value then it will print the specified number and vice versa.    **EXAMPLE:**         7,45,3,66,5,34,89    In the given example by using insertion sorting the numbers will be arranged in the ascending order. The example after insertion sort will be as follow: 3,5,7,34,45,66,89   The main issue with insertion sort is then when the numbers are arranged in reverse order then it consumes a lot of time to compile it in ascending order.    **AUXILIARY SPACE:**  Insertion sort can be written in **O(1)**  way.    **USES:**                 Insertion sort is only used when there is less numbers of data.                 It is also used when the array is approximately arranged and needs few numbers to be sorted.      We can use binary search to reduce the number of comparisons in normal insertion sort. Binary Insertion Sort uses binary search to find the proper location to insert the selected item at each iteration. In normal insertion, sorting takes O (i) (at ith iteration) in worst case. We can reduce it to O (logi) by using binary search.    **HOW TO IMPLEMENT INSERTION SORT?**   1. Create an unsorted array in an irregular form. 2. Use insertion sort to iterate the whole array. 3. The array will be arranged in sorted form after compilation.     **CONCLUSION:**          Insertion sort is used only when we have a small number of values as it takes a lot of time. so, it is fewerly used in algorithms. |

|  |  |
| --- | --- |
| Pseudo Code | Insertion Sort(x)    For j=2 to array.length  Key=array[j]  //insert array[j]into the sorted-sequence array[1..j-1]  I=j-1  While i>0 and array[i]>key  array[i+1] = array[i]  I=i-1  array[i+1] = key |
| Python Code | *def InsertionSort(x):*    x=[]  inputElements = int(input("Enter number of elements : "))  #applying for loop for taking inputs    for i in range(0, inputElements):      elements = int(input())        x.append(elements)  #insertion    for j in range(1,len(x)):          key=x[j]          i=j-1          while i>0 and x[i] > key:              x[i+1]=x[i]              i=i-1          x[i+1]=key        #for i in range(len(x)):  print(x) |
| Time Complexity Analysis | Time complexity of insertion sort is O(N^2) in average case. But if the array in which we are going to apply insertion sort performs no swapping than the time complexity is O(N).  **Average case :**  Time complexity is: O(N^2)    **Best Case:**  Time complexity is: O(N)    **Worst Case:**  Time complexity is: O(N^2) |

|  |  |
| --- | --- |
| Proof of Correctness | To prove Insertion sort is correct there are three steps:  **Initialization:**  The step starts with the first element of the array to be sorted. The first element of the array is considered toe placed at 1 instead of 0.  **Maintenance:**  The whole array is to be checked by comparing with one another elements and is compared until whole array is sorted. The loop checks elements in array one by one and move the sorted elements to the left side and it repeats this until whole array is sorted. In the end the array consist of all the sorted elements. This is how the loop works that is placed inside the code.    **Termination:**  After the completion of the sorting of the array the loop terminates by printing the values of sorted array in output. But this happens after the completion of the loop. |

|  |  |
| --- | --- |
| Strengths | * Insertion sort is stable algorithm. * Insertion sort is a simple algorithm without any complexity. * This algorithm can be used to sort small arrays or lists. |
| Weaknesses | * Insertion sort does not act efficiently for larger arrays or lists. * Insertion sort is not a stable algorithm. * It will consume more time if the array or list data is not efficiently arranged. |
| Dry Run | *def InsertionSort(x):*    x=[]  inputElements = int(input("Enter number of elements : "))  #applying for loop for taking inputs    for i in range(0, inputElements):      elements = int(input())        x.append(elements)  #insertion    for j in range(1,len(x)):          key=x[j]          i=j-1          while i>0 and x[i] > key:              x[i+1]=x[i]              i=i-1          x[i+1]=key        #for i in range(len(x)):  print(x)  C:\Users\MAHNOOR SHAD\Downloads\WhatsApp Image 2021-11-05 at 2.42.08 AM.jpeg |

|  |  |
| --- | --- |
| Merge Sort |  |
| Description | Merge sort is used for sorting array by following three steps which are divide, conquer and combine.  Merge sorting is useful when there is a large number of values in an unsorted way. The main principle behind the merge sort is “DIVIDE AND CONQUER”.  Merge sort divides the array into two parts and compute them separately. This is easy way to arrange array more precisely.  The merge (array, l, m, r) is a key process that assumes that array[l..m] and array[m+1..r] are sorted and merges the two sorted sub-arrays into one.    **EXAMPLE**:               3 565 34 43 74 23 4 42 48 838 78  In the above array by using merge sorting the array will be divided into two equal parts which will further be divided into more equal parts. In this way it complies the array in ascending sorted order. The array after compilation will be as follow:        3 4 23 34 42 43 48 74 78 565 838     The time complexity used in merge sorting is:                                   T(n) = 2T(n/2) + θ(n)    **PRINCIPLE:**                The principle behind merge sorting is “DIVIDE AND CONQUER”     The main issue with merge sorting is that it takes a lot of time for small arrays. It repeats the whole step if the array is approximately arranged. In merge sorting algorithm requires an additional memory space of 0(n) for the temporary array.    **USES:**              Merge sorting is useful when we have a lot of values in array which cannot be sorted using insertion sorting.    **AUXILIARY SPACE:**  Insertion sort can be written in **O(1) way**.    **APPLICATIONS**:   * In external sorting * In inversion count problems * For sorting linked lists O(nLogn) time         Merge sorting is best used when array contains huge number of values. It takes time but gives more precise value then insertion sorting. |

|  |  |
| --- | --- |
| Pseudo Code | *Merge Sort (Array):*    MergeSort(array[],left ,right)  If right>left  First of all find the middle point to divide and divide the array in two halves  Mid x=left+(right-left)/2  Apply mergeSort On both first and second half  mergeSort(array,left,x)  mergeSort(array,x+1,right)  And finally combine the two sorted arrays  Merge(array,right,x,left) |
| Python Code | def mergeSort(x):      if len(x) > 1:          middleTerms = len(x) // 2          leftSideTerms = x[:middleTerms]          rightSideTerms =x[middleTerms:]    #initializing variables          variable1 = 0          variable2= 0          variable3= 0                mergeSort(leftSideTerms)          mergeSort(rightSideTerms)            while variable1 < len(leftSideTerms) and variable2 < len(rightSideTerms):                x[variable3] = leftSideTerms[variable1]                  variable1 +=1              else:                  x[variable3] = rightSideTerms[variable2]                  variable2 += 1                    variable3 += 1          while variable1 < len(leftSideTerms):              x[variable3] = leftSideTerms[variable1]                variable1 += 1                variable3 += 1            while variable2 < len(rightSideTerms):              x[variable3]=rightSideTerms[variable2]              variable2 += 1              variable3 += 1    #taking input array  #DRIVER CODE  x=[]  inputElements = int(input("Enter number of elements : "))  #applying for loop for taking inputs    for i in range(0, inputElements):      elements = int(input())        x.append(elements)      mergeSort(x)    print(x) |
| Time Complexity Analysis | Time complexity of bubble sort is O(N^2) in the average case.  **Average case:**  Time complexity is: O(n\*Log n)  **Best Case:**  Time complexity is: O(n\*Log n)  **Worst Case:**  Time complexity is: O(n\*Log n) |

|  |  |
| --- | --- |
| Proof of Correctness | There are three steps to proving sort is correct:  **Initialization:**  The first element of the array to be sorted is used to begin the step. The array's first element is considered to be placed at 1 rather than 0.  **Maintenance:**  The entire array is to be checked by comparing one element to another until the entire array is sorted. The loop checks each element in the array one by one and moves the sorted elements to the left side, repeating this process until the entire array is sorted. Finally, the array contains all of the sorted elements. This is how the inside-the-code loop functions.  **Termination:**  After the array has been sorted, the loop is terminated by printing the sorted array's values in output However, this occurs after the loop has been completed. A |

|  |  |
| --- | --- |
| Strengths | * It is a stable sort. * It works efficiently for larger arrays or lists. * It combines two sorted arrays efficiently |
| Weaknesses | * It is not a stable sort. * It does not works efficiently for smaller arrays or lists. * It requires more memory space than other algorithms. |
| Dry Run | def mergeSort(x):      if len(x) > 1:          middleTerms = len(x) // 2          leftSideTerms = x[:middleTerms]          rightSideTerms =x[middleTerms:]    #initializing variables          variable1 = 0          variable2= 0          variable3= 0                mergeSort(leftSideTerms)          mergeSort(rightSideTerms)            while variable1 < len(leftSideTerms) and variable2 < len(rightSideTerms):                x[variable3] = leftSideTerms[variable1]                  variable1 +=1              else:                  x[variable3] = rightSideTerms[variable2]                  variable2 += 1                    variable3 += 1          while variable1 < len(leftSideTerms):              x[variable3] = leftSideTerms[variable1]                variable1 += 1                variable3 += 1            while variable2 < len(rightSideTerms):              x[variable3]=rightSideTerms[variable2]              variable2 += 1              variable3 += 1    #taking input array  #DRIVER CODE  x=[]  inputElements = int(input("Enter number of elements : "))  #applying for loop for taking inputs    for i in range(0, inputElements):      elements = int(input())        x.append(elements)      mergeSort(x)    print(x)    C:\Users\MAHNOOR SHAD\Downloads\WhatsApp Image 2021-11-05 at 2.42.11 AM.jpeg |

|  |  |
| --- | --- |
| Bubble Sort |  |
| Description | Bubble sort is used for sorting algorithm and is one is the simplest sorting algorithm in which the elements are sorted by repeatedly swapping of elements which are in wrong order.  The simplest sorting used in algorithm is bubble sorting. In this type of sorting the numbers are swapped if they are in wrong order.    **WORST AND AVERAGE CASE TIME:**  The time complexity for bubble sorting is O(n\*n). Worst case occurs when array is reverse sorted as it will consume a lot of time to swapped when any wrong order occurs.  It is best used when array is nearly arranged. The time complexity for this type of bubble sorting is O(n).                  If the array is nearly sorted then bubble sorting is the best sorting which is to be used in algorithm as it will sort it in less time. It is used for the beginners as it is easy to use and can be understand easily.    **EXAMPLE:**               2,4,5,7,6,54,3,1,9       In the above example bubble sort will swapped the number after 7 and will keep swapping until array is completely arranged. It will take a lot of time but arrange the given array. The array after swapping will be in the given shape        1,2,3,4,5,6,9,54    **AUXILARY SPACE**:                   The auxiliary space used in bubble sorting is O(1).    **OTHER NAMES**:         Bubble sorting is also known as comparison sorting. It is useful only for small arrays and for those which are nearly in arranged form.        It is very time-consuming work as it swaps the numbers and in every try it checks whether the last value is small then the next one or not.    **USES:**  Following are the uses:   * Is used where complexity does not occur. * Is used when simple and easy code is preferred.   **STABILITY:**                Bubble sorting is stable sorting just like merge and insertion sorting. The algorithm of bubble sorting is simple and gives stable array.    **CONCLUSION:**             Bubble sort is best in use when array is nearly arranged and has less values. If array is large and has many unsorted values then it is not used. |

|  |  |
| --- | --- |
| Pseudo Code | BubbleSort(list)  Loop=list.count  Apply the for loop for iteration  //now comparing the elements of list and swapping them  If list[j]>list[j+1]  Swap(list[j],list[j+1]  If there occurs no swapping in comparison than,  If()  Break  Return list |
| Python Code | x=[]  inputElements = int(input("Enter number of elements : "))  #applying for loop for taking inputs    for i in range(0, inputElements):      elements = int(input())        x.append(elements)  index=0  #using swapping technique(bubble sort)  for i in range(0,len(x)):          if x[i]!=0:              temp=x[index]              x[index]=x[i]              x[i]=temp              index+=1     # for i in range(len(x)):    print(x) |
| Time Complexity Analysis | Time complexity of bubble sort is O(N^2) in the average case.  **Average case:**  Time complexity is: O(N^2)    **Best Case:**  Time complexity is: O(N)    **Worst Case:**  Time complexity is: O(N^2) |

|  |  |
| --- | --- |
| Proof of Correctness | There are three steps to proving  sort is correct:  **Initialization:**  The step begins with the first element of the array to be sorted. The array's first element is assumed to be a 1 rather than a 0.    **Maintenance:**   The entire array must be checked by comparing one element to the next until the array is sorted. The loop iteratively checks each element in the array and moves the sorted elements to the left side, repeating the process until the entire array is sorted. Finally, the array contains all of the elements that have been sorted. This is how the within-the-code loop works.  **Termination**:  After sorting the array, the loop is terminated by printing the sorted array's values in output. However, this occurs after the loop has been completed. |

|  |  |
| --- | --- |
| Strengths | * It is a stable algorithm. * It is a simple algorithm to implement as it just works by swapping. * No extra space is required as it performs swapping inside the array without using temporary space. |
| Weaknesses | * It is not a stable algorithm. * As bubble sort performs lot of swapping so it takes much time to give the output. * It does not works efficiently for large arrays or lists because it will make them to consume a lot of time. |
| Dry Run | x=[]  inputElements = int(input("Enter number of elements : "))  #applying for loop for taking inputs    for i in range(0, inputElements):      elements = int(input())        x.append(elements)  index=0  #using swapping technique(bubble sort)  for i in range(0,len(x)):          if x[i]!=0:              temp=x[index]              x[index]=x[i]              x[i]=temp              index+=1     # for i in range(len(x)):    print(x)    C:\Users\MAHNOOR SHAD\Desktop\WhatsApp Image 2021-11-05 at 2.42.08 AM (1).jpeg |

|  |  |
| --- | --- |
| Quick Sort |  |
| Description | Quick sort is also like merge sort which works on divide, conquer and combine. It selects an element as pivot point and then move the elements accordingly until the array is sorted  Quick sort is just like merge sort as it also works on divide and conquer rule. Quick select is an algorithm which works in a way that we can find the smallest element in an un ordered array or list.  It works in a similar way as quicksort ,divides the data in two according to the pivot defined in the code.     But it is slightly different from merge sorting as it uses a pivot point about which the values of array are organized. The following can be used as pivot:   1. The first value of array. 2. The last value of array. 3. The middle value of array. 4. Abrupt value from array.               The key process in Quick sorting is partition (). Target of partitions is, given an array and an element x of array as pivot. All this should be done in linear time.    **PROCEDURE:**                           First take the left most value of array as the pivot point and then apply the quick sorting. It will divide the array into portions and then compute them. In this way it is arranged.    **PRINCIPLE:**      The quick sort works on divide and conquer principle.         The worst case for quick sorting is when it takes any large or small value as a pivot point. Due to which it takes a lot of time and becomes a bit complicated. The formula is as follow:                T(n) = T(n-1) + (n)     The best case is when it takes the middle value as pivot point. It becomes easy to solve such algorithms. The formula for such case is given below:                      T(n) = 2T(n/2) + (n)    **STABILTY**:              Quick sorting is not a stable sorting. However, it can be made stable by comparing parameters.    **3-WAY QUICK SORT**:               In 3- way Quick sorting array is divided into 3 portions and each portion is given a pivot point. It removes all the ambiguity and solves algorithm easily.    **CONCLUSION:**                Quick sorting is useful if pivot point is middle value but it is not a stable sorting for algorithm. |

|  |  |
| --- | --- |
| Pseudo Code | function quickSelect(list, left, right, k)   if left = right  return list[left]  Select a pivotIndex between left and right pivotIndex := partition(list, left, right,pivotIndex)   if k = pivotIndex   return list[k]  else if k < pivotIndex  right := pivotIndex – 1  else  left := pivotIndex + 1 |
| Python Code | #quick sort algorithm    def quickSort(array,l,r):        if(l<r):            pi = partition(array, l, r)          quickSort(arayr, l, pi - 1)          quickSort(array, pi + 1, r)      def partition (array, l, r):        pivot = array[r]      i = (l - 1)      for j in range (l,r):          if(array[j]<pivot):              i+=1              array[i] , array[j] = array[j] , array[i]      array[i+1],array[r] = array[r] , array[i+1]      return (i+1)    #driver code  array=[]    inputElements = int(input("Enter number of elements : "))    #applying for loop for taking inputs  size = len(array)  for i in range(0, inputElements):        elements = int(input())          array.append(elements)    quickSort(array,0 , size-1)  print(array) |
| Time Complexity Analysis | Time complexity of Quick select is O(N^2) in average case.  **Average case:**  Time complexity is: O(N log(N))    **Best Case:**  Time complexity is: O(N log(N))    **Worst Case:**  Time complexity is: O(N^2) |

|  |  |
| --- | --- |
| Proof of Correctness | **Initialization:**   The procedure starts with the first element of the array to be sorted. The first element of the array is assumed to be a 1 rather than a 0.    **Maintenance:**   The entire array must be checked by comparing one element to the next until the array is sorted. The loop iteratively checks each element in the array and moves the sorted elements to the left side, repeating the process until the entire array is sorted. Finally, the array contains all of the elements that have been sorted. This is how the within-the-code loop works.    **Termination:**  After sorting the array, the loop is ended by printing the sorted array's values in output. This, however, occurs after the loop has been completed. |

|  |  |
| --- | --- |
| Strengths | * It performs sorting efficiently fast. * It does not require more space. * It does not consume more time to give the output. |
| Weaknesses | * Quick sort is not stable. * It can cause many errors if the code is not written efficiently. * It does not perform or gives the correct output when the array is in sorted order. |
| Dry Run | #quick sort algorithm    def quickSort(array,l,r):        if(l<r):            pi = partition(array, l, r)          quickSort(arayr, l, pi - 1)          quickSort(array, pi + 1, r)      def partition (array, l, r):        pivot = array[r]      i = (l - 1)      for j in range (l,r):          if(array[j]<pivot):              i+=1              array[i] , array[j] = array[j] , array[i]      array[i+1],array[r] = array[r] , array[i+1]      return (i+1)    #driver code  array=[]    inputElements = int(input("Enter number of elements : "))    #applying for loop for taking inputs  size = len(array)  for i in range(0, inputElements):        elements = int(input())          array.append(elements)    quickSort(array,0 , size-1)  print(array)  C:\Users\MAHNOOR SHAD\Desktop\WhatsApp Image 2021-11-05 at 2.42.08 AM (1).jpeg |

|  |  |
| --- | --- |
| Heap Sort |  |
| Description | Heap sort is more like of a selection sort which divides the array in sorted and unsorted elements then solve the unsorted elements. Heap sort is a comparison-based sorting technique based on binary heap data structure. It is similar to selection sort where we first find the minimum element and place the minimum element at the beginning. We repeat the same process for the remaining elements.   Binary heap is a tree in which the parent node is always greater than the values of both the children node. It forms a tree in which every node creates two other nodes    **PROCEDURE**:        Firstly, build a heap of arrays and store the largest value in the root of the heap. Then check the value of each heap. Continue this step unless the array is in sorted form.    **NOTES**:   * Heap sort is an in-place algorithm. * Its typical implementation is not stable, but can be made stable. * The time complexity for heap sorting is 0(logon).     It is easy to understandable and simple to write. It uses less space in memory. It is more efficient then other sorting. It computes the array more fastly and accurately.    **APPLICATIONS**:   1. Sort a nearly sorted (or K sorted) array. 2. k largest(or smallest) elements in an array.   **APPLICATIONS OF HEAPSORT**:        Heap Data Structure is generally taught with Heapsort. Heapsort algorithm has limited uses because Quicksort is better in practice. Nevertheless, the Heap data structure itself is enormously used. Following are some uses other than Heap sort.    ***Priority Queues****:*  Priority queues can be efficiently implemented using Binary Heap because it supports insert(), delete() and extractmax(), decrease Key() operations in O (logn) time.  Binomoial Heap and Fibonacci Heap are variations of Binary Heap.    **CONCLUSION**:            Heapsorting is time consuming sorting in algorithm but is an accurate method which uses tree to check the array. Heap sorting arranges data into ascending order and creates a tree . |

|  |  |
| --- | --- |
| Pseudo Code | heapsort(array,size)  maximum=max(array)  //if root is not maximum than perform the swapping operation  Array[I],array[i+1]=array[i+1],array[i]  For I in range  heapArray(array,size,maximum)  arr[i],arr[0]=arr[0],arr[i]  heapsort(array,size) |
| Python Code | #heap sort      def heapify(array,size,x):      maximum=i      left=2\*i+1      right=2\*i+2        if maximum!=i:      arr[i], arr[maximum] = array[maximum], array[i]            heapify(array, size, maximum)  heapSort(array):  #In this step we will perform swapping       for i in range():                array[i], arayr[i-1] = array[i-1], array[i]    x=[]  inputElements = int(input("Enter number of elements : "))  #applying for loop for taking inputs    for i in range(0, inputElements):      elements = int(input())        x.append(elements)      heapSort(array)      for i in range(len(array))          print(array) |
| Time Complexity Analysis | Time complexity of heap sort is O(nlogn).  **Average case:**  Time complexity is: O(nlogn)    **Best Case:**  Time complexity is: O(n)    **Worst Case:**  Time complexity is: O(nlogn) |

|  |  |
| --- | --- |
| Proof of Correctness | To prove  sort is correct there are three steps:  **Initialization:**  The step starts with the first element of the array to be sorted. The first element of the array is considered toe placed at 1 instead of 0.  **Maintenance:**  The whole array is to be checked by comparing with one another elements and is compared until whole array is sorted. The loop checks elements in array one by one and move the sorted elements to the left side and it repeats this until whole array is sorted. In the end the array consist of all the sorted elements. This is how the loop works that is placed inside the code.    **Termination:**  After the completion of the sorting of the array the loop terminates by printing the values of sorted array in output. But this happens after the completion of the loop |

|  |  |  |  |
| --- | --- | --- | --- |
| Strengths | | * It is an efficient and simple sorting algorithm. * It does not use more memory. * It does not consume more time to performing sorting | |
| Weaknesses | | * It is not stable sorting algorithm. * It sometimes produces complexity in the code. * Order of elements having equal values are changed which can cause errors. | |
| Dry Run | | #heap sort      def heapify(array,size,x):      maximum=i      left=2\*i+1      right=2\*i+2        if maximum!=i:      arr[i], arr[maximum] = array[maximum], array[i]            heapify(array, size, maximum)  heapSort(array):  #In this step we will perform swapping       for i in range():                array[i], arayr[i-1] = array[i-1], array[i]    x=[]  inputElements = int(input("Enter number of elements : "))  #applying for loop for taking inputs    for i in range(0, inputElements):      elements = int(input())        x.append(elements)      heapSort(array)      for i in range(len(array))          print(array)  C:\Users\MAHNOOR SHAD\Downloads\WhatsApp Image 2021-11-05 at 2.42.09 AM.jpeg | |
| Selection Sort |  | |
| Description | Selection sort algorithm works by finding smallest element from the array and then move it to the first index and so on until the whole array is sorted. Selection sort sorts an array in a way that it finds the smallest value of the array and sets it at the beginning of the array. In this way the array is sorted in ascending order. This algorithm has two subarrays in a single array which can be as follow:   1. The subarray which is already sorted. 2. Remaining subarray which is unsorted.     The time complexity of selection sorting is O(n^2). As there are two nested loops in selection sorting.    **AUXILIARY SPACE:**         The auxiliary space for selection sorting is O(1).         The good point about selection sorting is that is does not creates more loop then the total number of values.    **SPACE REQUIRED**:          There is no need of space for selection sorting as it gets the first number and uses it as reference.    **EXAMPLE:**         34,5,12,56,87,11    In the above example selection sort will select the smallest number of array and sets it at extreme left and compares others value with the reference point. The required answer after using selection sorting is as follow:          5,11,12,34,56,87    **USING OF LOOPS:**               In selection sort loop is the principle as it moves in the form of a loop to arrange the array in ascending order from left to right.    **PROCEDURE:**     Set min value as 0 . Then makes a loop of selection sorting. Then repeat the step until the array is completely sorted.    **WORST CASE COMPLEXITY:**          Selection sorting is not good for large number of values as it will take a lot of time just to arrange it in sortable form as the time complexity is O(n^2).    **CONCLUSION:**        Selection sorting uses loops which is only useful when we have less data as it takes a lot of time to assemble it. Therefore, for large numbers selection is not used as algorithm. | |

|  |  |
| --- | --- |
| Pseudo Code | SelectionSort(list,size)  For I in range (1 to n-1)  //set the first element in the list minimum  Minimum=i  For j=i+1 to n  If list[j]<list[minimum] then  Minimum=j  //if the minimum index Is not equal to I than perform the swapping  Swap list[minimum] and list[i]  Return list |
| Python Code | def selectionSort(x):      for i in range(len(x)):          #assuming the first element to be the smallest element           smallestElement = i        for j in range(i+1, len(x)):              if x[smallestElement] > x[j]:                  smallestElement = j     # x[i], x[smallestElement] = x[smallestElement], x[i]      temp=x[i]      x[i]=x[smallestElement]      x[smallestElement]=temp        #DRIVER CODE  x=[]  inputElements = int(input("Enter number of elements : "))  #applying for loop for taking inputs    for i in range(0, inputElements):      elements = int(input())        x.append(elements)      selectionSort(x)      for i in range(len(x)):          print(x) |
| Time Complexity Analysis | Time complexity of selection sort is O(N^2).  **Average case:**  Time complexity is: O(N^2)    **Best Case:**  Time complexity is: O(N^2)    **Worst Case:**  Time complexity is: O(N^2) |

|  |  |
| --- | --- |
| Proof of Correctness | There are three steps to proving sort is correct:    **Initialization:**    The first element of the array to be sorted is used to begin the step. The array's first element is considered to be placed at 1 rather than 0.    **Maintenance:**    The entire array is to be checked by comparing one element to another until the entire array is sorted. The loop checks each element in the array one by one and moves the sorted elements to the left side, repeating this process until the entire array is sorted. Finally, the array contains all of the sorted elements. This is how the inside-the-code loop functions.      **Termination:**    After the array has been sorted, the loop is terminated by printing the sorted array's values in output However, this occurs after the loop has been completed |

|  |  |
| --- | --- |
| Strengths | * It works efficiently if the array or list is sorted. * It gives best output on the smaller arrays * It does not require a lot of space . |
| Weaknesses | * It is not a stable algorithm. * It does not performs well on larger arrays and lists. * It takes more time to perform sorting than other algorithms. |
| Dry Run | def selectionSort(x):      for i in range(len(x)):          #assuming the first element to be the smallest element           smallestElement = i        for j in range(i+1, len(x)):              if x[smallestElement] > x[j]:                  smallestElement = j     # x[i], x[smallestElement] = x[smallestElement], x[i]      temp=x[i]      x[i]=x[smallestElement]      x[smallestElement]=temp        #DRIVER CODE  x=[]  inputElements = int(input("Enter number of elements : "))  #applying for loop for taking inputs    for i in range(0, inputElements):      elements = int(input())        x.append(elements)      selectionSort(x)      for i in range(len(x)):          print(x)    C:\Users\MAHNOOR SHAD\Downloads\WhatsApp Image 2021-11-05 at 2.42.09 AM (1).jpeg |

|  |  |
| --- | --- |
| Shell Sort |  |
| Description | Shell sort is an algorithm that is basically an insertion sort algorithm ,but in insertion sort the elements are compared to next indexes and in shell sort we reduce the gap between the elements gradually            Shell sorting is advanced type of insertion sorting . It is more efficient method. It avoids large number of span and if the number is small then it moveNs to extreme left or right end. This algorithm is used to sort them in proper way and then sort them for large spaced values. The spacing used in it is known se interval. This spacing is determined by knuth’s formula which is as follow:                           h = h \* 3 + 1  where −     h is interval with initial value 1      **BEST CASE COMPLEXITY:**            Shell sorting is best of medium sized arrays having type O(n).As the gap is less so it can easily compile the array in ascending order.    **WORST CASE COMPLEXITY:**        Shell sorting is not valid for large number as it uses intervals due to which a lot of time is used to compile it having type O(n).    **TYPE COMPLEXITY**:          The type complexity for shell sorting is O(n).    **PROCEDURE:**      Firstly take the value of h.Then divide it into different intervals. Sort each interval using insertion sort. Then in this way the required array will be sorted in ascending order.    **EXAMPLE:**             34,23,7,123,10,37,87     In the given example shell sorting will divide it into subshells and then sort each part by using insertion sorting. When all subshells are sorted then sort them with respect to each other. The above example after sorting is as follow            7,10,23,34,37,87,123    **CLASS**:        The class used for shell sorting is sorting algorithm.    **STABILITY:**         Shell sorting is not stable sorting used for algorithm but it can be made stable by making small intervals.    **APPLICATIONS**:  Shell sort 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. Shellsort is, for example, used in the  library.    **CONCLUSION**:       Shell sorting is useful for small intervals |

|  |  |
| --- | --- |
| Pseudo Code | shellSort(array,x)  divArray=x//2  //than apply while loop  Temporary=array[i]  J=1  Array[j]=temporary  //this while loop will execute until it reaches 0  Print(array) |
| Python Code | def shellSort(array,size):      arrayDivision=size/2      while arrayDivision>0:          for i in range(arrayDivision):              temp=array[i]               array[j] = array[j - arrayDivision]                  j -= arrayDivision                array[j] = temp          x=[]  inputElements = int(input("Enter number of elements : "))  #applying for loop for taking inputs    for i in range(0, inputElements):      elements = int(input())        x.append(elements)      shellSort(array,size)      for i in range(len(array)):          print(array) |
| Time Complexity Analysis | Time complexity of shell sort is O(n^2).  **Average case:**  Time complexity is: O(N^2)    **Best Case:**  Time complexity is: O(N)    **Worst Case:**  Time complexity is: O(N^2) |

|  |  |
| --- | --- |
| Proof of Correctness | There are three steps to proving sort is correct:    **Initialization:**    The first element of the array to be sorted is used to begin the step. The array's first element is considered to be placed at 1 rather than 0.    **Maintenance**:    The entire array is to be checked by comparing one element to another until the entire array is sorted. The loop checks each element in the array one by one and moves the sorted elements to the left side, repeating this process until the entire array is sorted. Finally, the array contains all of the sorted elements. This is how the inside-the-code loop functions.      **Termination**:    After the array has been sorted, the loop is terminated by printing the sorted array's values in output However, this occurs after the loop has been completed |

|  |  |
| --- | --- |
| Strengths | * Shell sort does not need more space or memory. * It is efficient for medium sized arrays. * It is similar to insertion sort but it works better than insertion sort. |
| Weaknesses | * It is not a stable algorithm. * It is not efficient for large arrays and lists. * It is a complex algorithm |
| Dry Run | def shellSort(array,size):      arrayDivision=size/2      while arrayDivision>0:          for i in range(arrayDivision):              temp=array[i]               array[j] = array[j - arrayDivision]                  j -= arrayDivision                array[j] = temp          x=[]  inputElements = int(input("Enter number of elements : "))  #applying for loop for taking inputs    for i in range(0, inputElements):      elements = int(input())        x.append(elements)      shellSort(array,size)      for i in range(len(array)):          print(array)    C:\Users\MAHNOOR SHAD\Downloads\WhatsApp Image 2021-11-05 at 2.42.10 AM.jpeg |

|  |  |
| --- | --- |
| Tree Sort |  |
| Description | Tree sort algorithm sorts the array by building binary search tree from the elements of the  array which are to be sorted Tree sorting is an algorithm which is based on binary search tree. It first creates a search tree and then sort it in ascending order.    **PROCEDURE:**        First take input of arrays in an element form and then creates a binary tree and gives the sorted form of data.    **AVERAGE CASE TIME COMPLEXITY**:              For tree sorting the average case time complexity is O (n log n)    **WORST CASE TIME COMPLEXITY:**           For tree  sorting the worst time complexity is O(n^2).This can be improved by using self balancing binary sheet.    **AUXILARY SPACE:**   The auxiliary space used for tree sorting is O(n)    **STABILITY:**      Tree sorting is a stable sorting used in algorithm. It is stable like insertion and merge sorting.    **ADVANTAGES**:        Tree sorting is an efficient in sorting in which time is saved in order to sort an array . It build a tree in which the parent is divided into two parts which is further divided into more section    **EXAMPLE:**      43,5,2,89,454,34,3,44       In the given example using tree sorting the biggest number is taken as main reference and then it is divided into two parts in which data is sorted in ascending order. The data after sorting will be in the given form            2,3,5,34,43,44,89,454    **COMPARISION WITH QUICK SORT:**                Tree sorting has few advantages over quicksort. It has better worst case complexity when a self-balancing tree is used, but even more overhead.    **IMPLEMENTATION**:            Tree sorting is used in C++, JAVA PYTHON and  in other important languages.    **CONCLUSION:**                   In short tree sorting is more useful then the quick sorting as it makes a binary tree system in order to find the specified sorted order of the given array. |

|  |  |
| --- | --- |
| Pseudo Code | Iterate(X)  If X ==NULL  Break  Else   Iterate(X.left)  Print x.data  Iterate(X.right) |
| Python Code | *PHYTHON CODE OF TREE SORT HAS NOT WRITTEN YET.* |
| Time Complexity Analysis | Time complexity of tree sort is O(n(logn)) in the average case.  **Average case:**  Time complexity is: O(nlogn)    **Best Case:**  Time complexity is: O(nlogn)    **Worst Case:**  Time complexity is: O(n^2) and O(nlogn) |

|  |  |
| --- | --- |
| Proof of Correctness | There are three steps to proving sort is correct:    **Initialization:**    The first element of the array to be sorted is used to begin the step. The array's first element is considered to be placed at 1 rather than 0.    **Maintenance:**    The entire array is to be checked by comparing one element to another until the entire array is sorted. The loop checks each element in the array one by one and moves the sorted elements to the left side, repeating this process until the entire array is sorted. Finally, the array contains all of the sorted elements. This is how the inside-the-code loop functions.      **Termination:**    After the array has been sorted, the loop is terminated by printing the sorted array's values in output However, this occurs after the loop has been completed |

|  |  |
| --- | --- |
| Strengths | * It is a stable algorithm. * It is a fast and efficient sorting algorithm. * In tree sort ,user can make changes without much complexity. |
| Weaknesses | * It is not a stable algorithm. * It is a complex sorting algorithm. * It consumes more memory than few other sorting algorithms |
| Dry Run | No Code |

|  |  |
| --- | --- |
| Tim Sort |  |
| Description | Tim sort is derived from insertion sort and merge sort. It divides the array into blocks to sort the array. Tim sorting is the type of sorting which is combination of both insertion and merge sort. A stable sorting algorithm works in O(n Log n) time  Used in Java’s Arrays.sort() as well as Python’s sorted() and sort().  First sort small pieces using Insertion Sort, then merges the pieces using merge of merge sort. In this we divide the blocks which is known as run. We consider the size of each run as 32. After this we sort each run one by one in order to compute the whole data.    **WORST CASE COMPLEXITY:**             The worst case complexity for tim sorting is O(nlogn).This is the main diffulty while dealing with the tim sorting.    **BEST CASE COMPLEXITY:**           The best case complexity for tim sorting is O(n).    **WORKING:**               Tim sort is a data sorting algorithm. It implements the idea that the real-world data sets almost always contain already ordered subsequences, so the sorting strategy is to identify them and sort them further using both merge and insert methods.    **STABILITY**:            The tim sorting is a stable sorting as it an compute data as it is combination of merge and insertion sorting. It can be used when data is in large amount.    **IMPLEMENTATION:**            Tim sorting is implemented in languages like C++, C ,, JAVA ,PYTHON,etc.    **ADVANTAGE**:        Tim sorting is useful when we have to compute large number of values as it is combination of insertion and merge sorting.    **EXAMPLE:**         3,4,132,654,1,2,45,6 In the given example by using tim sorting the given array is computed using both merge and insertion sorting so after compution the data will be as follow           1,2,3,4,6,45,132,654    **CLASS:**     The class of tim sorting is also sorting algorithm just like insertion and merge sorting.    **CONCLUSION:**          If any data is not sorted using merge or insertion sorting  then tim sorting is used. |

|  |  |
| --- | --- |
| Pseudo Code | timSort(array)  //divide the array in two halves  dividedArray=np.array(array,2)  //sort the two halves using insertion sort  InsertionSort(array)  For j=2 to array.length  Key=array[j]  //insert array[j]into the sorted-sequence array[1..j-1]  I=j-1  While i>0 and array[i]>key  array[i+1]=array[i]  I=i-1  array[i+1]=key  //Than use merge sort to merge the halves |
| Python Code | def mergeSort(array,size):      if len(x) > 1:          middleTerms = len(x) // 2          leftSideTerms = x[:middleTerms]          rightSideTerms =x[middleTerms:]    #initializing variables          variable1 = 0          variable2= 0          variable3= 0                mergeSort(leftSideTerms)          mergeSort(rightSideTerms)            while variable1 < len(leftSideTerms) and variable2 < len(rightSideTerms):                x[variable3] = leftSideTerms[variable1]                  variable1 +=1              else:                  x[variable3] = rightSideTerms[variable2]                  variable2 += 1                    variable3 += 1          while variable1 < len(leftSideTerms):              x[variable3] = leftSideTerms[variable1]                variable1 += 1                variable3 += 1            while variable2 < len(rightSideTerms):              x[variable3]=rightSideTerms[variable2]              variable2 += 1              variable3 += 1 |
| Time Complexity Analysis | Time complexity of tim sort is O(n(logn)) in the average case.  **Average case:**  Time complexity is: O(n(logn))    **Best Case:**  Time complexity is: O(n)    **Worst Case:**  Time complexity is: O(n(logn)) |

|  |  |
| --- | --- |
| Proof of Correctness | There are three steps to proving sort is correct:    **Initialization:**    The first element of the array to be sorted is used to begin the step. The array's first element is considered to be placed at 1 rather than 0.    **Maintenance:**    The entire array is to be checked by comparing one element to another until the entire array is sorted. The loop checks each element in the array one by one and moves the sorted elements to the left side, repeating this process until the entire array is sorted. Finally, the array contains all of the sorted elements. This is how the inside-the-code loop functions.      **Termination:**    After the array has been sorted, the loop is terminated by printing the sorted array's values in output However, this occurs after the loop has been completed |

|  |  |
| --- | --- |
| Strengths | * It is a stable sorting algorithm. * It reduces the work of merge and insertion sort. * It performs more efficiently than insertion and merge sort. |
| Weaknesses | * It is a complex sorting algorithm. * It can cause errors if the code is not written correctly and efficiently. * Calculation of time complexity is also difficult in this sorting algorithm. |
| Dry Run | def mergeSort(array,size):      if len(x) > 1:          middleTerms = len(x) // 2          leftSideTerms = x[:middleTerms]          rightSideTerms =x[middleTerms:]    #initializing variables          variable1 = 0          variable2= 0          variable3= 0                mergeSort(leftSideTerms)          mergeSort(rightSideTerms)            while variable1 < len(leftSideTerms) and variable2 < len(rightSideTerms):                x[variable3] = leftSideTerms[variable1]                  variable1 +=1              else:                  x[variable3] = rightSideTerms[variable2]                  variable2 += 1                    variable3 += 1          while variable1 < len(leftSideTerms):              x[variable3] = leftSideTerms[variable1]                variable1 += 1                variable3 += 1            while variable2 < len(rightSideTerms):              x[variable3]=rightSideTerms[variable2]              variable2 += 1              variable3 += 1  C:\Users\MAHNOOR SHAD\Downloads\WhatsApp Image 2021-11-05 at 2.43.41 AM.jpeg |

|  |  |
| --- | --- |
| Counting Sort |  |
| Description | Counting sort is an integer sorting algorithm that work by iterating the input until the array is sorted. Counting sort is a sorting technique based on keys between a specific range. It works by counting the number of objects having distinct key values (kind of hashing). Then doing some arithmetic to calculate the position of each object in the output sequence. Counting sorting is  is a type of sorting in which arithmetic functions are performed to find the value of the specified number in a array.    **TIME COMPLEXITY:**             The time complexity of counting sort is O(n+k). In it n is the number of elements while k is the range of the elements.    **AUXILARY SPACE:**             The auxiliary space for counting sort is O(n+k).    **DISADVANTAGE:**      The main issue with counting sort is that it cannot find any negative value if there is any negative value in it.    **IMPORTANT POINTS**:  Counting sort is efficient if the range of input data is not significantly greater than the number of objects to be sorted. Consider the situation where the input sequence is between range 1 to 10K and the data is 10, 5, 10K, 5K.   **2.** It is not a comparison based sorting. It running time complexity is O(n) with space proportional to the range of data. .  **3.** It is often used as a sub-routine to another sorting algorithm like radix sort.   **4.** Counting sort uses a partial hashing to count the occurrence of the data object in O(1).   **5.** Counting sort can be extended to work for negative inputs also.    **EXAMPLE:**        -3,-45,-34,4,234,56,23     In the given example by using counting sorting arithmetic functions are performed  to find the numbers in ascending order. In this way array is sorted . The example after performing counting sort is as follow .                -45,-34,-3,4,23,56,234    **CONCLUSION:**                   Counting sort is used when there are both positive or negative numbers so to sort them in  ascending order it is used. |

|  |  |
| --- | --- |
| Pseudo Code | function CountingSort(input)   k = range of elements of array   count ← array of k + 1 zeros   output ← array of same length as input   for i = 0 to length(input) - 1 do   j = key(input[i])   count[j] += 1  for i = 1 to k do   count[i] += count[i - 1]  for i = length(input) - 1 down to 0 do   j = key(input[i])   count[j] -= 1   output[count[j]] = input[i]  return output |
| Python Code | array=[]    inputElements = int(input("Enter number of elements : "))    #applying for loop for taking inputs    size = len(array)+1    for i in range(0, inputElements):        elements = int(input())          array.append(elements)    countSetter=[0]\*size  for i in range(len(array)):      countSetter[array[i]] +=1    for j in range(1,len(countSetter)):      countSetter[j]=countSetter[j]+countSetter[j-1]  array2=[0]\*len(array)    for k in range (array-1,-1,-1):      array2[countSetter[array[k]]-1]=array[k]      countSetter[array[k]]=countSetter[array[k]]-1    print(array2) |
| Time Complexity Analysis | Time complexity of counting sort is O(n+k).  **Average case:**  Time complexity is: O(n+k)    **Best Case:**  Time complexity is: O(n+k)    **Worst Case:**  Time complexity is: O(n+k). |

|  |  |
| --- | --- |
| Proof of Correctness | To prove Insertion sort is correct there are three steps:  **Initialization:**  The step starts with the first element of the array to be sorted. The first element of the array is considered toe placed at 1 instead of 0.  **Maintenance:**  The whole array is to be checked by comparing with one another elements and is compared until whole array is sorted. The loop checks elements in array one by one and move the sorted elements to the left side and it repeats this until whole array is sorted. In the end the array consist of all the sorted elements. This is how the loop works that is placed inside the code.  **Termination:**  After the completion of the sorting of the array the loop terminates by printing the values of sorted array in output. But this happens after the completion of the loop |

|  |  |
| --- | --- |
| Strengths | * It is a stable algorithm. * It is an efficient sorting algorithm as it first calculates the frequency of each element and than performs sorting. * It is fast sorting algorithm. |
| Weaknesses | * It is a stable algorithm. * It is an efficient sorting algorithm as it first calculates the frequency of each element and than performs sorting. * It is fast sorting algorithm. |
| Dry Run | array=[]    inputElements = int(input("Enter number of elements : "))    #applying for loop for taking inputs    size = len(array)+1    for i in range(0, inputElements):        elements = int(input())          array.append(elements)    countSetter=[0]\*size  for i in range(len(array)):      countSetter[array[i]] +=1    for j in range(1,len(countSetter)):      countSetter[j]=countSetter[j]+countSetter[j-1]  array2=[0]\*len(array)    for k in range (array-1,-1,-1):      array2[countSetter[array[k]]-1]=array[k]      countSetter[array[k]]=countSetter[array[k]]-1    print(array2)    C:\Users\MAHNOOR SHAD\Downloads\WhatsApp Image 2021-11-05 at 2.42.10 AM (1).jpeg |

# INTEGRATION:

## Scrapping:

|  |  |
| --- | --- |
| We scrapped our data of medicine from the following website:  <https://www.1mg.com/>  This is the biggest website of online medicine and there are medicines in the form of categories available on this site. Following are the attributes which we scrapped from this website:   * Product Name * Quantity * Price * Description * Category * Discount * Ingredients | |
| C:\Users\MAHNOOR SHAD\Pictures\N8 Screenshots\Screenshot (212).png | Following are the categories from where we scrapped data with number of pages scrapped:   * + Vitamins and Supplements   + Sexual Wellness   + Tata 1mg health products   + Explore something new   + Deals of the day   + Minimum 33% off   + Buy more, save more   + Health devices   + Oral care |
| C:\Users\MAHNOOR SHAD\Pictures\N8 Screenshots\Screenshot (213).png | Following are the categories from where we scrapped data with number of pages scrapped:   * Featured * Protein supplements * Covid test and prevention * Nutritional drinks * Diabetes care * Mobility equipments * Ayurveda * Winter care * Baby care * Women care |
| C:\Users\MAHNOOR SHAD\Pictures\N8 Screenshots\Screenshot (214).png | Following are the categories from where we scrapped data with number of pages scrapped:   * Weight management * Sanitizers and hand wash * Nebulizers and vaporizers * Doctors corner * Hair care * Popular combo deals * Healing aids * Stomach care * Elderly care * Health food and drink |
| C:\Users\MAHNOOR SHAD\Pictures\N8 Screenshots\Screenshot (215).png | Following are the categories from where we scrapped data with number of pages scrapped:   * Skin care * Vital signs and wearable * Immunity boosters * Bone, joint and muscle care * Men care |

## Difficulties faced:

|  |  |
| --- | --- |
| There were many difficulties faced during the project which are following: | The solution of the difficulties are following: |
| * During scrapping the data was repeating due to loop because of which each product was repeating thousand of times again and again and it was repeating like one page was scrapped and with next page the first page was scrapped again and with the third page the first two pages was scrapped and so on. * When I was converting my .ui file in .py file in cmd.exe the file was not converting because of some error and it was again and again showing that the file is not recognized. I checked the path variable and checked the path but all was in vain. I tried to convert the file using pyuic5 –x filename.ui –o filename.py. * I worked on jupyter so whenever I make changes in my code as mu ui was having 2 screens and they both were linked so whenever I change the code I had to again link the file so that it can recognize the changes otherwise it runs without adding new changes made | * To overcome this problem we made our code generic because of which the same page was not repeating again and again. * To overcome this I had to delete the anaconda and reinstall that again. * To overcome this I linked my file again after making new changes. |

## Ideal Source:

|  |  |
| --- | --- |
| Following are the ideal sources of scrapping data: | |
| * The data should be scrapped from the site which is open for scrapping data because some sites don’t allow to scrap data | * Before scrapping check the attributes which are required are there or not. |
| * The best scrapping is done from single page scrapping because it takes less time when you scrap data from two pages it takes double time as it has to scrap data from next page also | * The data should be scraped from some site which contains large amount of data |

# Project Requirements:

|  |  |
| --- | --- |
| **Count of entities** | According to the requirement of the project the data should be at least 1 million scrapped. |
| **Project requirements** | Following were the requirements of the project:   * It should have option of pause, start, resume and stop. * Progress bar showing number of entities scrapped * Attributes should be at least 7 * UI can have the option of taking URL as input and output(bonus) |
| **UI requirements** | Following were the requirements of UI:   * It should be designed in PyQT * Should display list of chosen entities(in table) |
| **Sorting** | Following are the requirements of sorting:   * There should be Option of sorting in UI * There should be at least 13-14 algorithms * Display time of sorting in mm |
| **Searching** | * User should have the option to choose algorithm through which the data should be sorted. * There should be advanced filters in UI(Pause, start, resume and stop) |
| **Multi-level sorting** | User should have the option to sort data in columns using different sorting. |
| **Multi column searching** | There should be an option to search filters. |
| **CSV file** | To insert the scrap data safely we made a csv file which contains all of our scrapped data in the form of columns |

# UI implementation

|  |  |
| --- | --- |
| We made our UI in Qt designer. I t was the requirement of the project. | Here is the image of Qt designer:  C:\Users\MAHNOOR SHAD\Pictures\N8 Screenshots\Screenshot (216).png |

## Ideas and Implementation

|  |  |
| --- | --- |
| Idea | Implementation |
| C:\Users\MAHNOOR SHAD\Pictures\N8 Screenshots\Screenshot (195).png  This was the idea of login screen as it was on desktop. | C:\Users\MAHNOOR SHAD\Pictures\N8 Screenshots\Screenshot (218).png  This is the implementation of login screen which contains login options which are:   * Username * Password   There is also the option of remember me and forget password and when you submit this new screen will open which is main screen. |
| C:\Users\MAHNOOR SHAD\Pictures\N8 Screenshots\Screenshot (196).png  This is the main page idea which contains algorithms and attribues along with table | C:\Users\MAHNOOR SHAD\Pictures\N8 Screenshots\Screenshot (222).png  This is the Implementation of main screen which contains search option thorugh which user can search any medicine. It also contains algorithms through which data will be sorted along with order. The table shows the data which is scrapped. |

## Details of UI

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| C:\Users\MAHNOOR SHAD\Pictures\N8 Screenshots\Screenshot (223).png | The main screen of project contains an option of search in which one can search any medicine by just entering name of medicine and after entering name click on search button and if the medicine is there it will be searched.   |  |  | | --- | --- | | UI Component Name | Type of UI component | | Search | label | | Product name | Text edit | | search | button | |
| C:\Users\MAHNOOR SHAD\Pictures\N8 Screenshots\Screenshot (224).png | This is also there on main screen which contains three combo box which are following:   * Algorithms * Order * Attributes   In the combo box of algorithms there are 13 algorithms through which data can be sorted. The 13 algorithms are:   * Insertion sort * Merge sort * Bubble sort * Quick sort * K-select * Bucket sort * Radix sort * Counting sort * Selection sort * Heap sort * Tree sort * Tim sort * Shell sort   The next combo box is of order by which the data can be arranged which is of two order:   * Ascending order * Descending order   The next combo box is of attributes and user will select which attribute they want to sort. Following are the names of attributes:   * Product name * Quantity * Price * Description * Category * Discount * Ingredients   After selecting the options from the above user will press the run button and the array will be sorted accordingly.   |  |  | | --- | --- | | UI Component Name | Type of UI component | | Algorithms | Combo box | | Order | Combo box | | Attributes | Combo box | | Run | Button | |
| C:\Users\MAHNOOR SHAD\Pictures\N8 Screenshots\Screenshot (225).png | This is also on main screen. This is the table on which scrapped data will be shown. There is a scroll bar and the following are the seven attributes:   * Product name * Quantity * Price * Description * Category * Discount * Ingredients |
| C:\Users\MAHNOOR SHAD\Pictures\N8 Screenshots\Screenshot (226).png | This is on the top of the main screen which contains following four options:   * Start * Pause * Resume * Stop   These will be used during scrapping data, the process in which data will be displayed on the table. User can use one of the following four options in main screen. |

# Collaboration

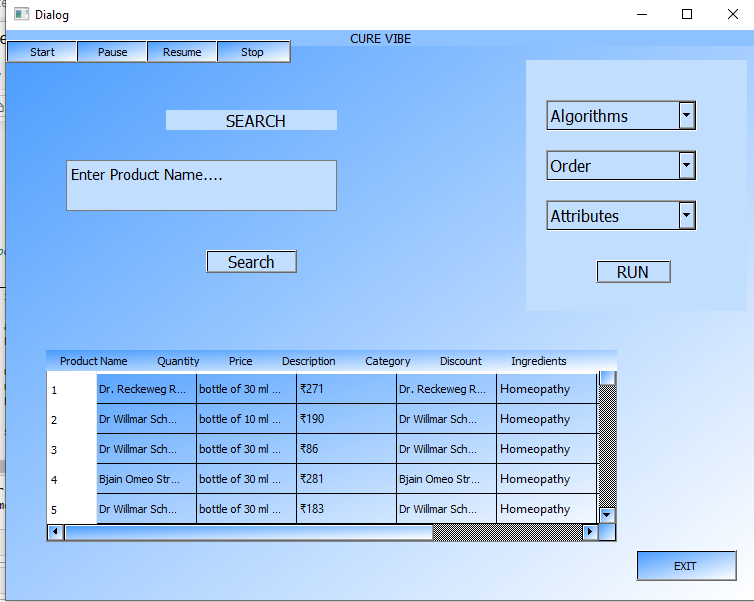
2020-CS-146 and 2020-CS-123 both collaborated in a quite well way. We both firstly shared our ideas and then implemented them on piece of paper. Later we implemented them with improvement on pencil tool and finally we implemented them with the best possible results in the form of UI. We implemented that in the form of code and resolved our errors with the help of each other. This project is the combine effort of both of us and the best result we can give.

# Task Division

We both divided the task keeping in view the strong points of each other and uploaded on GitHub. We both gave our best and we faced a lot of errors during the completion of our project but we resolved that with the help of each other.

# Final Application

The final result which came out of our project is:



We completed almost all the requirements of the project which were given to us which are:

**Count of entities:** According to the requirement of the project the data should be at least 1 million scrapped

**Project requirements:** Following were the requirements of the project:

* It should have option of pause, start, resume and stop.
* Progress bar showing number of entities scrapped
* Attributes should be at least 7

**UI requirements:** Following were the requirements of UI:

* It should be designed in PyQT
* Should display list of chosen entities(in table)

**Sorting:** Following are the requirements of sorting:

* There should be Option of sorting in UI
* There should be at least 13-14 algorithms
* Display time of sorting in mm

**Searching:**

* User should have the option to choose algorithm through which the data should be sorted.
* There should be advanced filters in UI(Pause, start, resume and stop)

**Multi-level sorting:** User should have the option to sort data in columns using different sorting.

**Multi column searching:** There should be an option to search filters.

**CSV file:** To insert the scrap data safely we made a csv file which contains all of our scrapped data in the form of columns.

All above requirements are completed in the project.

## Remaining

The remaining things which are not completed are following:

The order is not completed in ascending order and descending order also not all sorts are inserted in the code.

# Unfinished ideas

The ideas which we thought of implemented but we cannot implement because of less time and were also difficult are following:

* We cannot add prescription option which the patients can enter and the following medicine in the prescription will be added in add to cart
* We cannot enter the option of add to cart where medicines will be added and there will be option of place order.