Virtual World Projects

Report

Submitted By:
Hammad Ahmad
2020-CS-30
Muhammad Huzaifa Khan
2020-CS-02
Group: 50

Submitted To:
Mr. Samyan Qayyum Wahla
CS-261-DSA

Department of Computer Sciences
University of Engineering and Technology, Lahore

Table of Contents

1.Project Overview

Description

The project is a retrieval system where data related to projects available on freelancing websites will be extracted. The data will be useful for online jobs analysts where they would be able to see project's details in a specific manner. For example, they would be able to see projects with the highest payments, highest reviews, ratings, etc. 1 million data is to be fetched and different sorting and searching techniques will be applied converting the data into some type of information. The project's flow will be in a manner where 1 million data will be extracted as a result from different freelancing websites using their URLs. The data scraping will be controlled by the users where they would be able to start, pause, resume, and stop the scrapping. The data extracted then will undergo sorting techniques where the users would be able to sort the data according to the types provided to them. Searching on different attributes of the entity will be according to the data type of the respective attribute for example, for integer the searching will be according to the ranges of integers and in case of string it will be according to the letters in the word. A progress bar will be displayed on the UI showing the scrapping progress. The main purpose of the project will be to show the time taken for each sorting technique. We would be able to examine the time that takes for each sorting technique and tell which technique will be the best for a specific amount of data. Sorting on a particular column and among columns (multi-level sorting) will be a feature providing the user with different types of sorting.

Motivation

The project will be helping in visualizing how sorting is applied on a real-life problem. So far, we have been implementing different types of sorting algorithms on integers and now to extend them on real life examples will give a broader and more vivid understanding of these algorithms. Seeing the time complexity of algorithms on large data will help to explain their limitations and as a result, tell which algorithm will be the best when we have large data or small data.

Target Audience

Online Project's Analysts, People searching for projects, and Algorithm Experts analyzing time for each algorithm.

Business Need

Analysis of online projects available on freelancing websites requires them to be in an area where they can be sorted according to their ratings, prices, categories. Analysts can use this to determine the projects available in a certain category, the prices of each project, and the ratings of the project dealer telling if it is suitable and safe to take the project of the person. Moreover, the time analysis of sorting techniques can help Algorithm Analyzers to conclude which algorithm works best for a certain amount of information.

2.Scraped Data

Attributes

Name	Data Type	Description
Title	String	The title will tell what the project is about.
Category	String	The category in which the project falls for example, making logos, symbols, trademarks, etc. will fall in the category of logo design.
Name	String	The person who is advertising the project.
Cost ($)	Integer	The cost for completing the project.
Delivery (Days)	Integer	Number of days in which the project will be delivered.
Reviews	Integer	Number of reviews of the person's project.
Ratings	Float	Rating of the person providing the project.

Sample Scrapping

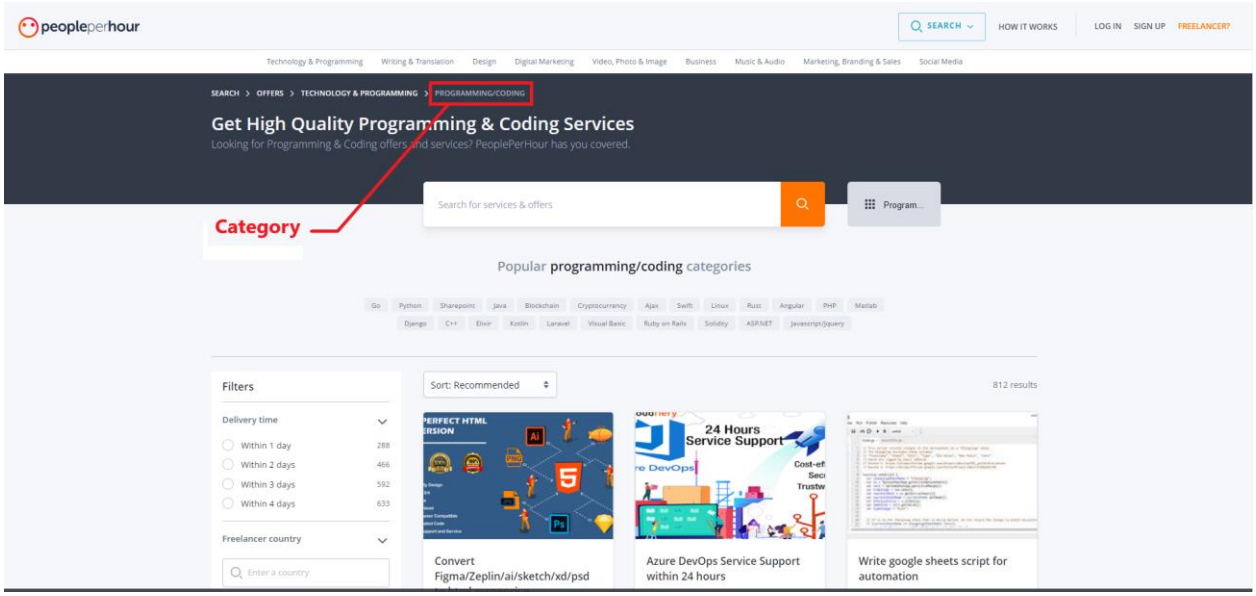

Figure 1: Interface of the webpage "People Per Hour" with highlighted attributes of the Project

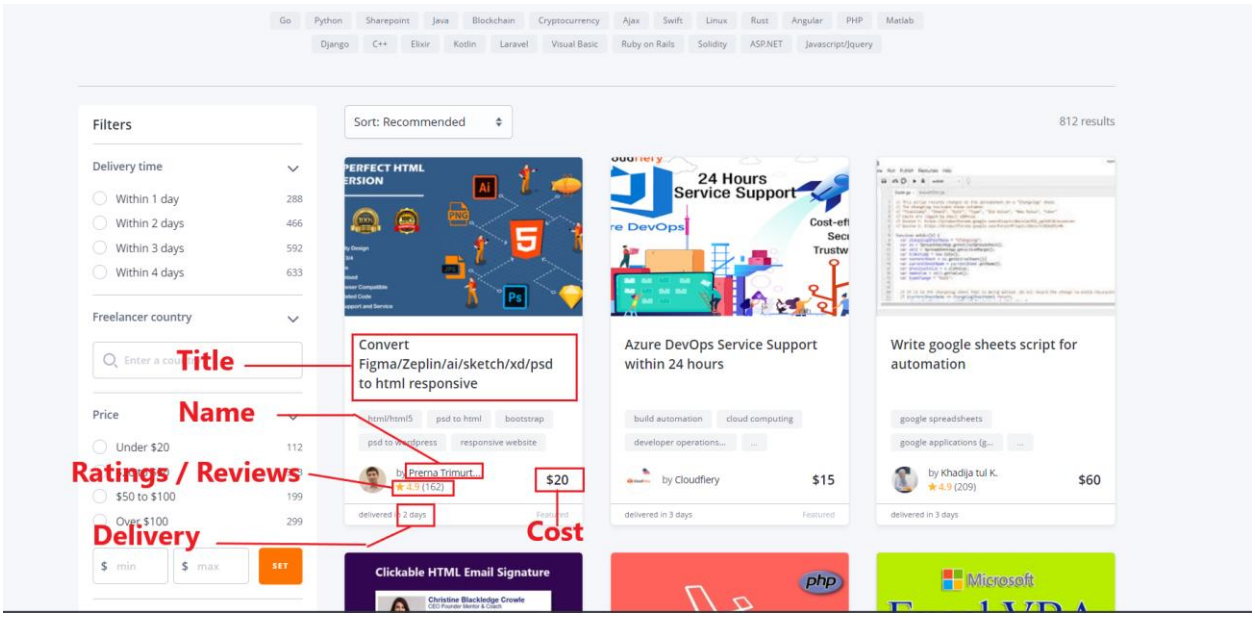

Figure 2: Scrolled interface of the webpage "Peoples Per Hour" with the highlighted attributes

3.Algorithms

Sorting Algorithms

Selection Sort:

Description

First the minimum element throughout the array is selected and inserted as a first element of the array. Then the second minimum element is selected, and this goes on until all elements don't get sorted.

Algorithm

```
def selectionSort(A):
    for i in range(len(A)):
        for n in range(i+1,len(A)):
            if A[n] < A[i]:
                new =  A[i]
                A[i] = A[n]
                A[n] = new
    return A
```

Insertion Sort:

Description

We have three parts of the array: the sorted part, current element, and the unsorted part. Current element is picked starting from index 1 of the array and is inserted in the sorted array at its correct position.

Algorithm

```
def insertionSort(A):
    for i in range(1, len(A)):
        currentNum = A[i]
        for j in range (i-1 , -1 , -1):
            if (currentNum < A[j]):
                A[j+1] = A[j]
                A[j] = currentNum
            else :
                A[j+1] = currentNum
                break
    return A
```

Merge Sort:

Description

A recursive algorithm which divides the array into two halves until single elements are left. Then each half is sorted and combined with its other half making the array sorted.

Algorithm

```
def mergeSort(A , a , b , indexArray):
    if a!=b :
        mid = (a + b) // 2
        copyIndex = indexArray.copy()
        mergeSort(A , a , mid , indexArray)
        mergeSort(A , mid + 1 , b ,indexArray)
        merge(A , a  , mid , b , indexArray)
    return indexArray
def merge(A , a , mid , b , indexArray):
    L = []
    R = []
    Li = []
    Ri = []
    copyIndex = indexArray.copy()
    for i in range(a , mid+1):
        L.append(A[i])
        Li.append(copyIndex[i])
    for i in range(mid+1 , b+1):
        R.append(A[i])
        Ri.append(copyIndex[i])
    i = a
    j = 0
    k = 0
    while (j < len(L) and k < len(R)):
        if (L[j] < R[k]):
            indexArray[i] = Li[j]
            A[i] = L[j]

            j+=1
            i+=1
        else:
            A[i] = R[k]
            indexArray[i] = Ri[k]
            k+=1
            i+=1
    while (j < len(L)):
        A[i] = L[j]
        indexArray[i] = Li[j]
        i+=1
        j+=1
    while (k < len(R)):
        indexArray[i] = Ri[k]
        A[i] = R[k]
        k+=1
        i+=1
```

Bubble Sort:

Description

Iterative algorithm which swaps adjacent elements w.r.t the condition applied e.g., if the left element is larger than the right adjacent element, then swap. This causes a sorted array to appear from the end of the array.

Algorithm

```
def bubbleSort(A , indexArray):
    a = 0
    j = len(A)-2
    while a < j:
        for i in range(j+1):
            if(A[i] > A[i+1] ):
                indexArray[i],indexArray[i+1] = indexArray[i+1],indexArray[i]
                A[i] , A[i+1] = A[i+1] , A[i]
        j-=1
    return indexArray
```

Bucket Sort:

Description

Make Buckets equal to the largest element of the array. Insert each element in its bucket and sort each bucket individually. At the end, combine all the buckets in a final array.

Algorithm

```
def calculateDivNumber(n):   # calculates the number with which each number will be divided in bucket sort
    tens = "1"
    for i in range(n):
        tens += "0"
    return int(tens)
def bucketSort(A , indexArray):
    B=[[],[],[],[],[],[],[],[],[],[]]
    idx=[[],[],[],[],[],[],[],[],[],[]]
    typechr = checkType(str(A[0]))
    if type(typechr) == str:
        return 0
    maxi = max(A)
    divNum = calculateDivNumber(len(str(maxi)))  # Number with which each number is going to be divided
    for i in range(len(A)):
        floatNum  = A[i]/divNum
        B[math.floor(floatNum*10)].append(A[i]) # Appending numbers in their corresponding buckets
        idx[math.floor(floatNum*10)].append(indexArray[i])
    return allignArray(insertionBucketSort(B , idx))
def allignArray(A):                    # Allign and store the buckets back to original array
```

```
    output = []
    for r in range(len(A)):
        for c in range(len(A[r])):
            output.append(A[r][c])
    return output
def insertionBucketSort(A , indexArray):
    for i in range(0, len(A)):
        row = A[i]
        irow = indexArray[i]
        insertionSort(row , irow)
    return indexArray
```

Counting Sort:

Description

Make an array containing 0's of length equal to the largest number contained in the input array. Count each number in the input array and increment at the index equal to the element in the new array. Create a final Element and insert the elements according to a rule.

Algorithm

```
def key(A, i):
    ascii = []
    for character in A:
        if type(character) == str:
            ascii.append(ord(character[0]))
        else:
            ascii.append(character)

    return (ascii[i]+(-1*min(ascii)))

def countingSort(A , indexArray):
    A = list(map(str, A))
    copyIdx = indexArray.copy()
    ascii = []
    for character in A:
        typechr = checkType(character)
        if typechr == character :
            ascii.append(ord(character[0]))
        elif type(typechr) == float:
            return 0
        else:
            ascii.append(typechr)
    size = (max(ascii) - min(ascii)) + 1
    count = []
    output = [0]*(len(A))
    #  Making an array of size with 0 stored at each place
    for i in range(size+1):
```

```
    count.append(0)
  #  counting the occurence of each element
  for i in range(len(A)):
    count[key(ascii,i)]+=1
  for i in range(1,len(count)):
    count[i] += count[i-1]
  for i in range(len(A)-1 , -1 , -1):
    count[key(ascii,i)]-=1
    output[count[key(ascii,i)]]= A[i]
    indexArray[count[key(ascii,i)]]= copyIdx[i]

  return indexArray
```

Radix Sort:

Description

Sorts the numbers from least significant integers to most significant. Numbers are inserted into buckets from 0 to 9 for each integer and at the end the first element inserted is removed first from the bucket.

Algorithm

```
def CountingSortforRadix(ascii , index , A , indexArray):
  currentIDXARRAY = []
  sortedarr = [0]*len(ascii)
  copyIndex = indexArray.copy()

  for i in range(len(ascii)):
    num = len(str(ascii[i]))
    if(num < A):
      newNum = addZeros(ascii[i] , A)
      currentIDXARRAY.append(int(newNum[index]))
    else:
      newNum = str(ascii[i])
      currentIDXARRAY.append(int(newNum[index]))
  size = (max(currentIDXARRAY) - min(currentIDXARRAY)) + 1
  count = []
  output = [0]*(len(ascii))
  #  Making an array of size with 0 stored at each place
  for i in range(size+1):
    count.append(0)
  #  counting the occurence of each element
  for i in range(len(currentIDXARRAY)):
    count[key(currentIDXARRAY,i)]+=1

  for i in range(1,len(count)):
    count[i] += count[i-1]
  for i in range(len(currentIDXARRAY)-1 , -1 , -1):
```

```
        count[key(currentIDXARRAY,i)] -= 1
        output[i] = count[key(currentIDXARRAY,i)]
        # indexArray[key(currentIDXARRAY,i)], indexArray[i] =
indexArray[i],indexArray[key(currentIDXARRAY,i)]

    for i in range(len(output)):
        sortedarr[output[i]] = ascii[i]
        copyIndex[output[i]] = indexArray[i]

    ascii = sortedarr
    indexArray = copyIndex

    return ascii ,indexArray
def key(A , i):
    return(A[i]+(-1*min(A)))
def addZeros(A , digits):
    length = len(str(A))
    diff = digits - length
    for i in range(diff):
        A = "0"+str(A)
    return A
def radixSort(A , indexArray):
    typechr = checkType(A[0])
    ascii = []
    if (type(typechr) == str):
        for i in range(len(A)):
            ascii.append(ord(A[i][0]))
    elif type(typechr) == float:
        return 0
    else:
        ascii = A
    A = len(str(max(ascii)))
    for i in range (A-1,-1,-1):
        ascii ,indexArray = CountingSortforRadix(ascii , i , A, indexArray)
    return(indexArray)
```

Quick Sort:

Description

We select an element from the array and arrange the elements smaller than it on the left and greater to the right. Then the Quick Sort function is called again and sorts the left and right parts around the pivot.

Algorithm

```
def quickSort(A , low , high , indexArray):
    # sys.setrecursionlimit(10**9)
    if low < high :
```

```
        pi = partition(A , low , high , indexArray)
        quickSort(A , low , pi-1 , indexArray)
        quickSort(A , pi+1 , high , indexArray)
    return indexArray
def partition(A , low , high , indexArray):
    pivot = A[high]
    i = low
    for j in range (low , high):
        if (A[j]<pivot):
            A[i] , A[j] = A[j],A[i]
            indexArray[i] , indexArray[j] = indexArray[j],indexArray[i]
            i+=1
    A[high] = A[i]
    indexArray[high] , indexArray[i] = indexArray[i] , indexArray[high]
    A[i] = pivot
    return i
```

Heap Sort:

Description:

We create heaps of the input elements and then make a max. heap by making the root nodes value maximum. After that we swap the root node with the smallest element at the end of the heap and remove the last largest element.

Algorithm

```
def maxHeapify(A, n, i , indexArray):
    largest = i
    left = 2 * i + 1     # left child
    right = 2 * i + 2    # right child
    if left < n and A[largest] < A[left]:
        largest = left
    if right < n and A[largest] < A[right]:
        largest = right
    if largest != i:
        A[i], A[largest] = A[largest], A[i]
        indexArray[i], indexArray[largest] = indexArray[largest], indexArray[i]
        maxHeapify(A, n, largest , indexArray)
def heapSort(A , indexArray):
    n = len(A)
    for i in range(n//2 - 1, -1, -1):
        maxHeapify(A, n, i , indexArray)
    for i in range(n-1, 0, -1):
        A[i], A[0] = A[0], A[i]
        indexArray[i], indexArray[0] = indexArray[0], indexArray[i]
        maxHeapify(A, i, 0 , indexArray)
    return indexArray
```

Pigeonhole Sort:

Description

Make an array of size equal to the range. Iterate through the array and subtract the number with the minimum number which will give the index of the new array where the current element is to be placed. At the end, place the elements of the new array in the original array giving a sorted array.

Algorithm

```
def pigeonselectionSort(A , ridx):
    for i in range(len(A)):
        for n in range(i+1,len(A)):
            if A[n] < A[i]:
                new =  A[i]
                A[i] = A[n]
                A[n] = new
                ridx[i] , ridx[n] = ridx[n] , ridx[i]
    return A , ridx
def pigeonholeSort(A , indexArray):
    ascii = []
    floatType = False

    for character in A:
        typechr = checkType(character)
        if type(typechr) == str :
            ascii.append(ord(character[0]))
        elif type(typechr) == float:
            # return 0
            ascii.append(typechr)
            floatType = True
        else:
            ascii.append(typechr)
    maxi = math.floor(max(ascii))
    mini = math.floor(min(ascii))
    rangeSize = maxi - mini + 1
    pHoles = [[] for j in range(rangeSize)]
    pHolesidx = [[] for j in range(rangeSize)]

    for i in range (len(A)):
        idx = math.floor(ascii[i] - mini)
        pHoles[idx].append(A[i])
        pHolesidx[idx].append(indexArray[i])

    i = 0
    for r in range(len(pHoles)):
        row = pHoles[r]
        rowidx = pHolesidx[r]
```

```
        if floatType:
            row , rowidx= pigeonselectionSort(row , rowidx)
        for c in range(len(row)):
            A[i] = row[c]
            indexArray[i] = rowidx[c]
            i+=1
    return indexArray
A = [0.4,0.234,5.2,7.3,1.5,0.1]
indexArray = [0,1,2,3,4,5]
print(pigeonholeSort(A , indexArray))
```

Shell Sort:

Description

Calculate the gap starting from n/2 and compare elements starting from index 0 with elements at index (n/2+current Index). Swap the elements if the left element is larger. Decrease the gap again by n/2 and again check the elements until a sorted array appears.

Algorithm

```
def shellSort(A , indexArray):
    n = len(A)
    gap = n//2

    while gap > 0:
        i = 0
        k = gap

        while k < n:
            if A[i] > A[k]:
                A[i],A[k] = A[k],A[i]
                indexArray[i],indexArray[k] = indexArray[k],indexArray[i]
            i += 1
            k += 1
            j = i
            while j-gap > -1:
                if A[j-gap] > A[j]:
                    A[j-gap],A[j] = A[j],A[j-gap]
                    indexArray[j-gap],indexArray[j] = indexArray[j],indexArray[j-gap]
                j -= 1
        gap = gap // 2
    return indexArray
```

Comb Sort:

Description

Calculate a gap dividing the total number of elements with 1.3 and compare elements starting from index 0 with index = [gap+index] and swap if the left element is greater. In the next pass, divide the gap by 1.3 again and keep doing the swapping until a sorted array appears.

Algorithm

```
def combSort (A , indexArray) :
    gap = len(A)
    while gap != 1 :
        gap = gap/1.3
        gap = int(gap)
        for i in range ( 0, len(A)-gap ) :
            if A[i] > A[i+gap] :
                A[i] , A[i+gap] = A[i+gap] , A[i]
                indexArray[i] , indexArray[i+gap] = indexArray[i+gap] , indexArray[i]

    return indexArray
```

Gnome Sort:

Description

Type of bubble sort in which iterate through the array and check if elements at current index and previous index are at the correct position or not. If not then swap and decrement the current index, otherwise, keep on iterating.

Algorithm

```
def gnomeSort (A , indexArray ) :
    i = 0
    while i < len(A) :
        if i == 0 :
            i += 1
        if A[i] >= A[i-1] :
            i+= 1
        else :

            A[i] , A[i-1] = A[i-1],A[i]
            indexArray[i] , indexArray[i-1] = indexArray[i-1],indexArray[i]
            i -= 1
    return indexArray
```

Searching Algorithms

Linear Search:

Description

Here we will iterate through the complete array one by one and find the required information.

Algorithm

```
def linearSearch ( self, columnIDX, filterIDX, searchData ) :
```

```
        row = self.tableWidget.rowCount()
        details = sorting.getColumnFromTable(self)
        print(row)
        print(columnIDX)
        print(filterIDX)
        print(searchData)

        for i in range ( 0, row ) :

            if ( columnIDX == 1 or columnIDX == 2 or columnIDX == 3 or columnIDX == 4 or
columnIDX == 5 or columnIDX == 6 or columnIDX == 7 ) :
                if filterIDX == 1 :
                    if details[columnIDX][i].find(searchData)!=-1 :
                        self.tableWidget.item(i, columnIDX-1).setBackground(QtGui.QColor(0,255,0))
                elif filterIDX == 2 :
                    if details[columnIDX][i].startswith(searchData) :
                        self.tableWidget.item(i, columnIDX-1).setBackground(QtGui.QColor(0,255,0))
                elif filterIDX == 3 :
                    if details[columnIDX][i].endswith(searchData) :
                        self.tableWidget.item(i, columnIDX-1).setBackground(QtGui.QColor(0,255,0))

            if ( columnIDX == 4 or columnIDX == 5 or columnIDX == 6 ) :
                if filterIDX == 11 :
                    print(int(details[columnIDX][i]))
                    if ( int(details[columnIDX][i]) >= 0 and int(details[columnIDX][i]) <= 20 ) :
                        print(int(details[columnIDX][i]))
                        self.tableWidget.item(i, columnIDX-1).setBackground(QtGui.QColor(0,255,0))
                elif filterIDX == 12 :
                    if ( int(details[columnIDX][i]) > 20 and int(details[columnIDX][i]) <= 50 ) :
                        self.tableWidget.item(i, columnIDX-1).setBackground(QtGui.QColor(0,255,0))
                elif filterIDX == 13 :
                    if ( int(details[columnIDX][i]) > 50 and int(details[columnIDX][i]) <= 100 ) :
                        self.tableWidget.item(i, columnIDX-1).setBackground(QtGui.QColor(0,255,0))
                elif filterIDX == 14 :
                    if ( int(details[columnIDX][i]) > 100 and int(details[columnIDX][i]) <= 150 ) :
                        self.tableWidget.item(i, columnIDX-1).setBackground(QtGui.QColor(0,255,0))
                elif filterIDX == 15 :
                    if ( int(details[columnIDX][i]) > 150 and int(details[columnIDX][i]) <= 200 ) :
                        self.tableWidget.item(i, columnIDX-1).setBackground(QtGui.QColor(0,255,0))
                elif filterIDX == 16 :
                    if ( int(details[columnIDX][i]) > 200 ) :
                        self.tableWidget.item(i, columnIDX-1).setBackground(QtGui.QColor(0,255,0))
            if ( columnIDX == 7 ) :
                if filterIDX == 17 :
                    if ( float(details[columnIDX][i]) >= 0 and float(details[columnIDX][i]) <= 1 ) :
                        self.tableWidget.item(i, columnIDX-1).setBackground(QtGui.QColor(0,255,0))
```

```
            elif filterIDX == 18 :
                if ( float(details[columnIDX][i]) > 1 and float(details[columnIDX][i]) <= 2 ) :
                    self.tableWidget.item(i, columnIDX-1).setBackground(QtGui.QColor(0,255,0))
            elif filterIDX == 19 :
                if ( float(details[columnIDX][i]) > 2 and float(details[columnIDX][i]) <= 3 ) :
                    self.tableWidget.item(i, columnIDX-1).setBackground(QtGui.QColor(0,255,0))
            elif filterIDX == 20 :
                if ( float(details[columnIDX][i]) > 3 and float(details[columnIDX][i]) <= 3.5 ) :
                    self.tableWidget.item(i, columnIDX-1).setBackground(QtGui.QColor(0,255,0))
            elif filterIDX == 21 :
                if ( float(details[columnIDX][i]) > 3.5 and float(details[columnIDX][i]) <= 4 ) :
                    self.tableWidget.item(i, columnIDX-1).setBackground(QtGui.QColor(0,255,0))
            elif filterIDX == 22 :
                if ( float(details[columnIDX][i]) > 4 and float(details[columnIDX][i]) <= 4.5 ) :
                    self.tableWidget.item(i, columnIDX-1).setBackground(QtGui.QColor(0,255,0))
            elif filterIDX == 23 :
                if ( float(details[columnIDX][i]) > 4.5 and float(details[columnIDX][i]) <= 5 ) :
                    self.tableWidget.item(i, columnIDX-1).setBackground(QtGui.QColor(0,255,0))

    def findColumnidx(self) :
        index = self.selectSearchColumn.currentIndex()
        return index

    def findFilteridx(self) :
        self.given = self.getInputText()
        self.givenList = self.given.split()
        if self.givenList[0] == "contains" :
            return 1
        elif self.givenList[0] == "starts" and self.givenList[1] == "with" :
            return 2
        elif self.givenList[0] == "ends" and self.givenList[1] == "with" :
            return 2
        elif self.givenList[0] == "0-20" :
            return 11
        elif self.givenList[0] == "20-50" :
            return 12
        elif self.givenList[0] == "50-100" :
            return 13
        elif self.givenList[0] == "100-150" :
            return 14
        elif self.givenList[0] == "150-200" :
            return 15
        elif self.givenList[0] == ">200" :
            return 16
        elif self.givenList[0] == "0-1" :
            return 17
```

```
    elif self.givenList[0] == "1-2" :
        return 18
    elif self.givenList[0] == "2-3" :
        return 19
    elif self.givenList[0] == "3-3.5" :
        return 20
    elif self.givenList[0] == "3.5-4" :
        return 21
    elif self.givenList[0] == "4-4.5" :
        return 22
    elif self.givenList[0] == "4.5-5" :
        return 23

def getInputText(self):
    given = self.searchArea1.text()
    return given

def findSearchTypeidx(self) :
    Type = self.searchType.currentIndex()
    return Type

def connect(self) :
    value1 = self.findColumnidx()
    value2 = self.findFilteridx()
    value3 = self.findSearchTypeidx()
    given = self.getInputText()
    return (value1, value2, value3, given)

def get(self) :
    self.value1, self.value2, self.value3, self.given = self.connect()
    print (self.value1)
    print (self.value2)
    print (self.value3)
    print (self.given)
    self.givenList = self.given.split()
    self.searchText = ""
    IDX = self.findFilteridx()

    if IDX == 1 :
        self.searchText = self.givenList[1]
    elif IDX == 2 :
        self.searchText = self.givenList[2]
    elif IDX == 3 :
        self.searchText = self.givenList[2]
    print (self.searchText)
    if self.value3 == 1 :
```

```
    self.linearSearch ( self.value1, self.value2, self.searchText )
elif self.value3 == 2 :
    self.binarySearch ()
```

Binary Search:

Description

A divide and conquer technique where we sort the array and divide it into two parts and check in which part our answer would be and discard the other half. We continue to do this until we find our required number.

4.UI

Initial GUI Design

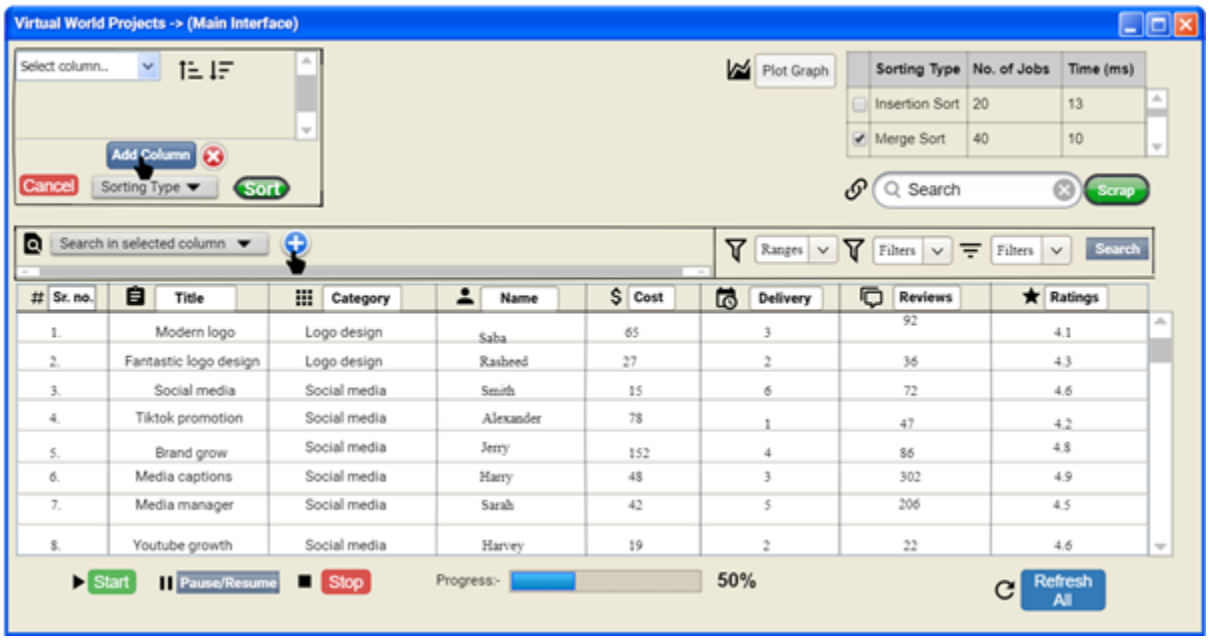

Figure 3: contains a table widget in which scrapped data get loaded through csv file.It contains two frames at the top right and top left corner of the window in which the techniques of time complexity analysis and the sorting processes will be implemented.It has two merged frames for searching of data throughout the table.

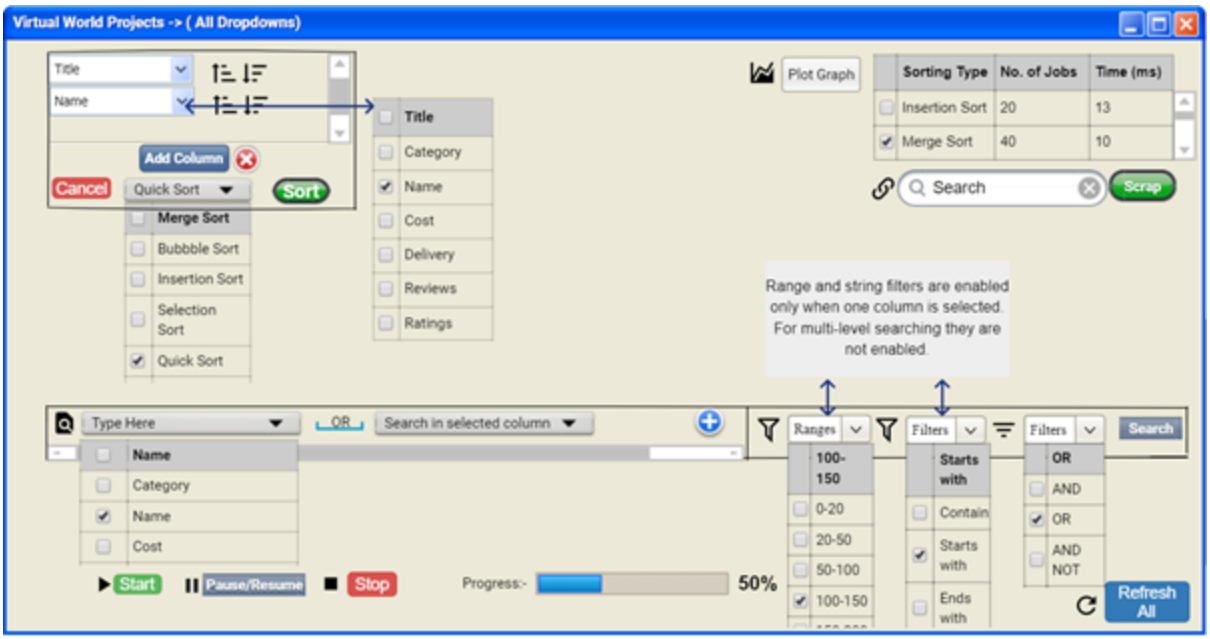

Figure 4 contains all the combo boxes with their dropdowns and consists of all the buttons to perform specific actions on scrapped data.

GUI Components

UI Component Name	**Type of UI component**	**Purpose of UI Component/Other details**
Select column	Dropdown	It will be used to avail the option of selecting column or multiple columns.
Sort ascending	Icon	By clicking this icon, we will be able to sort the selected column in an ascending manner.
Sort descending	Icon	By clicking this icon, we will be able to sort the selected column in a descending order.
Select sorting type	Dropdown	This dropdown will provide an option to select different types of sorting algorithms in order to sort a specific column.

Sort	Button	This button will enable the process of sorting.
Cancel	Button	By clicking this button, the user can unselect the column or algorithm type.
Plot graph	Button	This button will allow us to plot a graph between two types of sorting
Search	Search bar	This bar will take the URL as input.
Scrap	Button	The button will scrap the material according to the given URL.
Search in selected column	Dropdown	It will allow us to select a column & in result, we can search in that column using filters.
Add	Button	This will allow you to search through multiple columns using filters.
Ranges	Dropdown	In order to select the range for columns in terms of integers.
Filters	Dropdown	In order to select the filters for columns in terms of strings like contains, starts with & ends with, etc.
Filters	Dropdown	These composite filters will serve for the process of multi-column searching using AND, OR & NOT.
Start	Button	To start scraping.
Pause/Resume	Button	To pause and resume the scraping process.

Stop	Button	To stop scraping.
Progress	Bar	To show the progress of scraping.
Refresh All	Button	To reload the whole page and to discard all the changes.

Implemented GUI

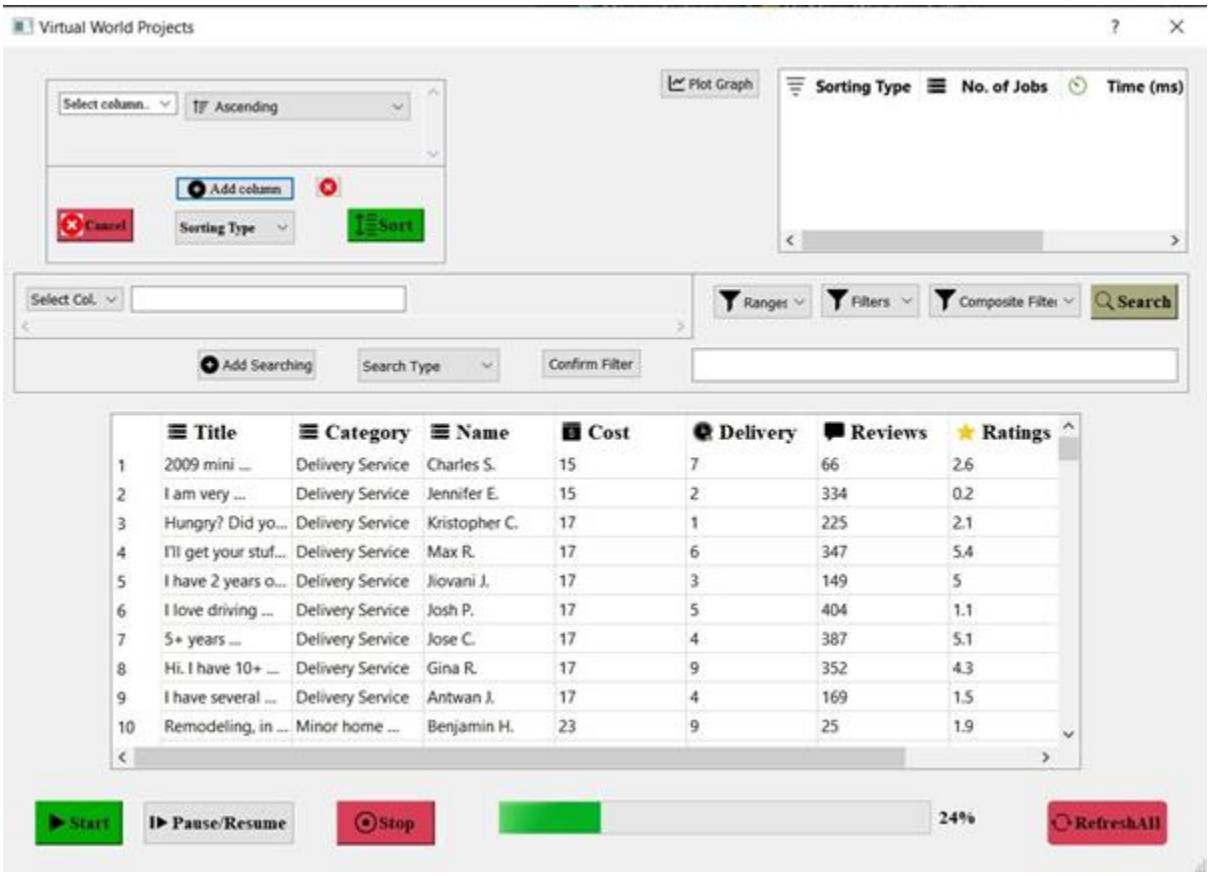

Figure 5 This figure contains the updated GUI that is working according to the mentioned requirements of the projects. The frame at the top left corner is for the implementation of sorting algorithms on the data. Frame at the top right corner will show the sorting processes performed and show the time complexity analysis for each algorithm. Moreover, there are two frames for searching of data merged into each other and have combo boxes and filters according to the requirements.

5.Integration

Problems and their solutions

1. The Wireframe that we made were not perfect so that all the functions can be applied perfectly. For example, when we were going to implement the searching part, over there the boxes that we showed in the wireframes earlier were not accurate that they could implement searching.
So, we had to alter our GUI a little bit so that things could start making sense. First, we added a confirm filter option to select the filter that we selected from the combo box, then we needed to insert both composite filters along with simple filters so on confirming the filters it checked for both the filters and wrote them in the search box in the format (CompositeFilter SimpleFilter SomethingToBeSearched).
2. The sorting and searching codes that we wrote previously were simply for integer type data. So, all of the codes needed alterations in them so that they can work simultaneously for integers, floats and strings. Although, the logic was same but coming up with the alterations was a time taking task.
3. Integration was time taking task as the syntax for getting elements from the PyQt Designer through code and implementing functions on them was not something that was documented somewhere on the internet. We needed to search into videos to find out our desired syntax which took a lot of time. But this problem was helpful in learning a lot of new things.
4. Splitting the filters and data in the searching part took time as the split function was not giving an array rather it was giving values in string. But then searching the proper syntax from the internet helped to solve that problem.
5. During the integration of algorithms with the GUI, the most common problem was that changing a little bit of the GUI and converting it to python file caused the previous useful code of the python file to be deleted too. We just then used to create a backup file and then copy pasted the previous code that we wanted from the back up file in the new python file.
6. Working on GitHub on the same project with the GUI part was again a hard task as we needed to be careful so that while fetching the data, the code that we wrote in the files does not get overwritten resulting in loss of data. So again, we created backup files to deal with this issue and copy pasted the previously written code to the new fetched files.

Overview of the whole working

First time working on a group project was a fun part as we learnt a lot of new things not only related to programming tasks but also with managing the work and distributing it among members which gave an idea that how projects are completed in the practical field coming ahead. Moreover, we learnt new softwares like PyQt to design our GUI and Pencil Tool for wireframes designing. As the project was specifically related to apply sorting and searching techniques on a large data, so it was a practical way to view the time complexity of the various algorithms that

we studied in our theory. Although the project could not be implemented completely but it cannot be said that learning was not there. Seeing the time taken by each algorithm for sorting on the same data helped to understand why it is important to write an optimized code.

6.Collaboration

The team members collaborated through in person meetings in the University Library discussing the problems both were facing and coming up with their solutions. Prospects regarding the coming tasks and distributing the work among them was discussed either in the class or through online meetings on Teams or WhatsApp so that work can be finished before hand. Furthermore, if work on a particular part was to be discussed and completed mutually then either they met at their homes or in the University Library to have one on one meetings. GitHub was used to work on the project so that each other's work status can be viewed.