ARM Limited

CS262-Design

Project Supervisor

Mr. Samyan Qayyum Wahla

Project Members(G-5) :

Mukarram Ali 2021-CS-58

Rayan Rasheed 2021-CS-59

Ammad Aslam 2021-CS-67

DEPARTMENT OF COMPUTER SCIENCE

UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Contents

1 Project Description

We are facing not just a challenging business environment but a rapidly changing one. There is always room for improvement. Running a Distribution System without a proper application can lead to many problems. In the warehouse, by not managing your stock count, cost price changes and not being able to view your stock position can have drastic consequences. The company has no way of knowing how much stock is left in your warehouses and how much you've sold out. If you don't know how much you have left, you can't restock your inventory which will cause delays and frustrate your customers. It is important to track your inventory so that you can plan ahead effectively. The company is unable to keep a record of its employees, day-to-day sales and profits, customer information and transportation cost. Searching a 10-year-old record for the company is extremely challenging. Speed is also critical when ensuring that you get the correct items picked, packed and distributed to the correct address and client. Understanding the right approach to solving these challenges is essential for distributors to adapt, transform and differentiate themselves in this new challenging business landscape.

We are all aware that we live in an increasing electronic world. Simply put if the Distribution Company is not using a proper, well-managed application then it is facing commercial suicide. So, our Project aims to develop such an application for the Company that can address all the above problems and help the company improve the distribution system and management of the company processes.

The administrator of the application is the General Manager of the company. He can hire the company employees (sales agent), riders (to deliver the packages) and an inventory Manager. He is in the charge of each module. Each employee would have their individual accounts. The inventory Manager is in charge of the warehouse, keeping the record of the product as being delivered by the Manufacturing Company. The sales agent will deal with the clients, their orders and assign the delivery task to the riders depending upon the location assigned to them. The rider would be a given a specific area in which he would have to deliver all the goods and complete all its orders in the given time. Further bonuses will be given according to the statistic chart at the end of the month.

The Distribution Company that we have plumped for is the Shoe Distribution Company. A well-recognized shoe Manufacturing Company ADIDAS is our manufacturer. The implementation of this application will work with a company called ARM. As a distribution company we deliver the products to numerous retailers, wholesalers, concept stores, buyers and agents all over the Lahore. It operates on its own as it is an independent company establishing a connection between the products and the client.

The Record and performance are also evaluated automatically in order to provide the appraisal. The company established the excellent track record for the best customer satisfaction. As a footwear sourcing company, we also provide sustainable material sourcing options to help our clients choose a greener path. An organization's main focus must be to satisfy its customers and in order to do so we are providing a refund policy for the client and email verification after the order had been sent to them. Sometimes, the rider doesn't deliver the parcel to the exact location. So, to overcome this problem we provide the rider with the exact location

of client on Google map. The client should place the minimum required order then the sales agent would request the inventory manager for the confirmation of the stock. The application also maintains the daily attendance of the employees, the scheduling of the riders and fuel consumption cost. The clients would also be informed of the special offers. Our system is cited as the most efficient tool that is at the company's disposal.
The application offers three modes.
• Manager authorized mode.
• Sales Agent authorized mode.
• Inventory Supervisor mode
• Rider authorized mode.

2 Project Features:

• User interface screen will be operated according to the role of the person who signs in.
• Client can pay in installments or pay in advance.
• Rider will be informed with the stock availability during placing orders. If the required order of the client is out of stock, order could not be placed and an email would be sent to the Inventory Supervisor to inform about stock unavailability.
• Sales Agent will assign the location of decided area to the Rider. The order will be delivered the very next day.
• Rider could only deliver limited number of orders in a single day.
• Attendance of all employees.

3 Technology Stack

Language	1. Python
IDEs	1.Visual Studio Community Visual Studio Code

4 Project actors

Actor Name	Manager
Actor Type	Primary
Description	Manager can hire and fire employees to the Company. He can view all the records or monthly reports and stock. He also has to check the performances at all times, check inventory. He gives monthly incentives to all the employees according to their performance.

Actor Name	Inventory Supervisor
Actor Type	Primary
Description	This system will also have an inventory supervisor. He will manage inventory in the warehouse and will notify the general manager whenever a new order should be made. He can buy stock from the supplier after the approval of the General Manager. He will provide authorization to the rider after checking the stock from the warehouse.

Actor Name	Sales Agent
Actor Type	Primary
Description	Sales Agent will also have an account of its own. He will keep a track of all the orders of the riders and their information. He will assign a specific area to a specific rider. He can also update the location of riders. He will sendout email to the client after the delivery of the product.

Actor Name	Rider
Actor Type	Primary
Description	The Rider has most of the responsibilities in the system. Rider will take order from the shopkeepers and deliver the order afterwards. Rider is able to see the products when creating an order. He can view the history of his delivered orders and can also view pending orders (to be delivered). He will send out email to sale agent about the order placement. He can add and update the client data.

Actor Name	FBR
Actor Type	Off-Stage Actor
Description	FBR collect tax from companies from their earned profits under the tax ordinance law, 2001. It is pertinent for all registered companies to pay tax to work legally across country.

5 Use Cases

5.1 Reset Password:

Use Case ID	U01
Name	Reset Password
Actor	The Manager, Rider, Inventory Supervisor and Sales agent.
Description	If the user forgets his/her password, they can reset it.
Flow	Base Flow: 1.The user opens the application. 2.He enters the email and password. 3.He clicks on login. 4.A message box appears containing the message “Not matched in Data Base”. 5.He clicks on forget password. 6.After clicking, he receives an email containing the new password. Repeats step 1-3 7.Successfully login in the main page.

5.2 Add Employee

Use Case ID	U02
Name	Add Employee
Actor	The Manager
Description	The Manager can add a new employee to the company. It could either be the rider, sales agent, workers or the inventory supervisor. He would take the name, email, CNIC, address, phone number, bank account of the employee. After filling out the details, he will give them a password and status and a base salary depending upon the status.
Flow	Base Flow: 1.Customer arrives at company. 2.Fill out the form to give interview. 3.After passing interview, he will officially become company’s employee. To give him access to the application, the manager will register that employ by entering all his details which include his name, CNIC, E-mail address, status, bank account, telephone number. 4.After entering all his information, he has given his login details to login into application. Alternative Flow: 3a.The employee is a rider. 1.He also assigns a vehicle to the rider.

5.3 Deduction of Fuel Money:

Use Case ID	U03
Name	Deduction of Fuel Money
Actor	The Manager
Description	The rider will send a report to the Manager on a weekly or daily basis about the fuel consumption depending. Depending upon that report money will be deducted from the company account automatically by the confirmation of the Manager.
Flow	Base Flow: 1.The rider opens the fuel report. 2.Adds all the information about fuel consumption. 3.After clicking sent report, the Manager will receive the report. 4.The Manager will open finance module and will delete the total amount of money spend on fuel of each vehicle.

5.4 Give Salaries:

Use Case ID	U04
Name	Give Salaries
Actor	The Manager
Description	The Manager is in the charge of giving salaries to all the employees.
Flow	Base Flow: 1.The Manager logged into the system. 2.He clicks on the Finance button and from the dropdown menu, he selects "Salaries". 3.Now he can view the all the employees and the salary that is needed to be paid to them. 4.When the manager clicks pay button, money will be transferred to their account and deducted from the company account. 5.A message box will be shown of successfully transaction of money. 6.An email would be sent out to the employee being paid. Alternative flow: 1a.The Manager forgets his password. 1.He clicks on 'Forgot Password' to recover his account. 4a.Company account does not have enough money to pay the employees. 1.He will debit into the company account. 2.He requests the employee to get paid in installments.

5.5 Give Bonuses

Use Case ID	U05
Name	Give Bonus
Actor	The Manager
Description	The Manager give bonuses to riders and sales agent depending upon their monthly performances based upon bar chart.
Flow	Base Flow: 1.The Manager logged into the system. 2.He clicks on the Finance button and from the dropdown menu, he selects "Salaries". 3.Two bar graphs will be shown to him. First will be of the rider performances based on their total orders and sales agent depending upon the working days. 4.He will select the employee and click on Bonus button. 5.A pop up will be shown in which he will input the bonus amount ranging from 10 thousand to 20 for rider and 5 to 10 thousand for the sales agent. 6.He clicks on pay and the money will be transferred to the employee. Alternative flow: 1a.The Manager forgets his password. 1.He clicks on 'Forgot Password' to recover his account.

5.6 Update Employee:

Use Case ID	U06
Name	Update Employee
Actor	The Manager
Description	The Manager is able to update employees by clicking the button that list the employees and then select the employee whose information needs to be update.
Flow	Base Flow: 1.The Manager logged into the system. 2.An employee comes to him and asks to change some information about him. 3.The Manager clicks on the button and gets the list of all the employees of the company. 4.He searches for that particular employee. 5.He clicks and updates the information that is required to be updated. Alternative Flow: 1a. The Manager forgets his password. 1. He clicks on 'Forgot Password' to recover his account 4a. The employee name does not found in the data base. 1. The Manager uses U02.

5.7 Add Vehicle:

Use Case ID	U07
Name	Add Vehicle
Actor	The Manager
Description	The Manager is able add buy a new vehicle for the riders to deliver the products.
Flow	Base Flow: 1.The Manager logged into the system. 2.Today is the day to buy a new vehicle. 3.He clicks on the button of add vehicle. 4.Enters the truck model number and fuel average of that truck. 5.Enters the price of that vehicle. 6.Clicks add. 7.Money gets deducted from the company account. Alternative Flow: 1a. The Manager forgets his password. 1. He clicks on 'Forgot Password' to recover his account.

5.8 Delete Employee:

Use Case ID	U08
Name	Delete Employee
Actor	The Manager
Description	The Manager gets to fire the employee by deleting his information from the Database or when any employee leaves the company.
Flow	Base Flow: 1.The Manager logged into the system. 2.An employee comes to him and asks to resign 3.The Manager clicks on delete employee option. 4.The Manager gets the list of all the employees of the company. 5.He searches for that particular employee. 6.He clicks and deletes that employee. Alternative Flow: 1a. The Manager forgets his password. 1. He clicks on 'Forgot Password' to recover his account 5a. The employee is a rider. 1. The vehicle associated with is now free and can be assign to any new rider.

5.8.1 Votes

This tells us the Total votes by public

5.9 Sample Scrapping Source

We used IMBD as a scraping source

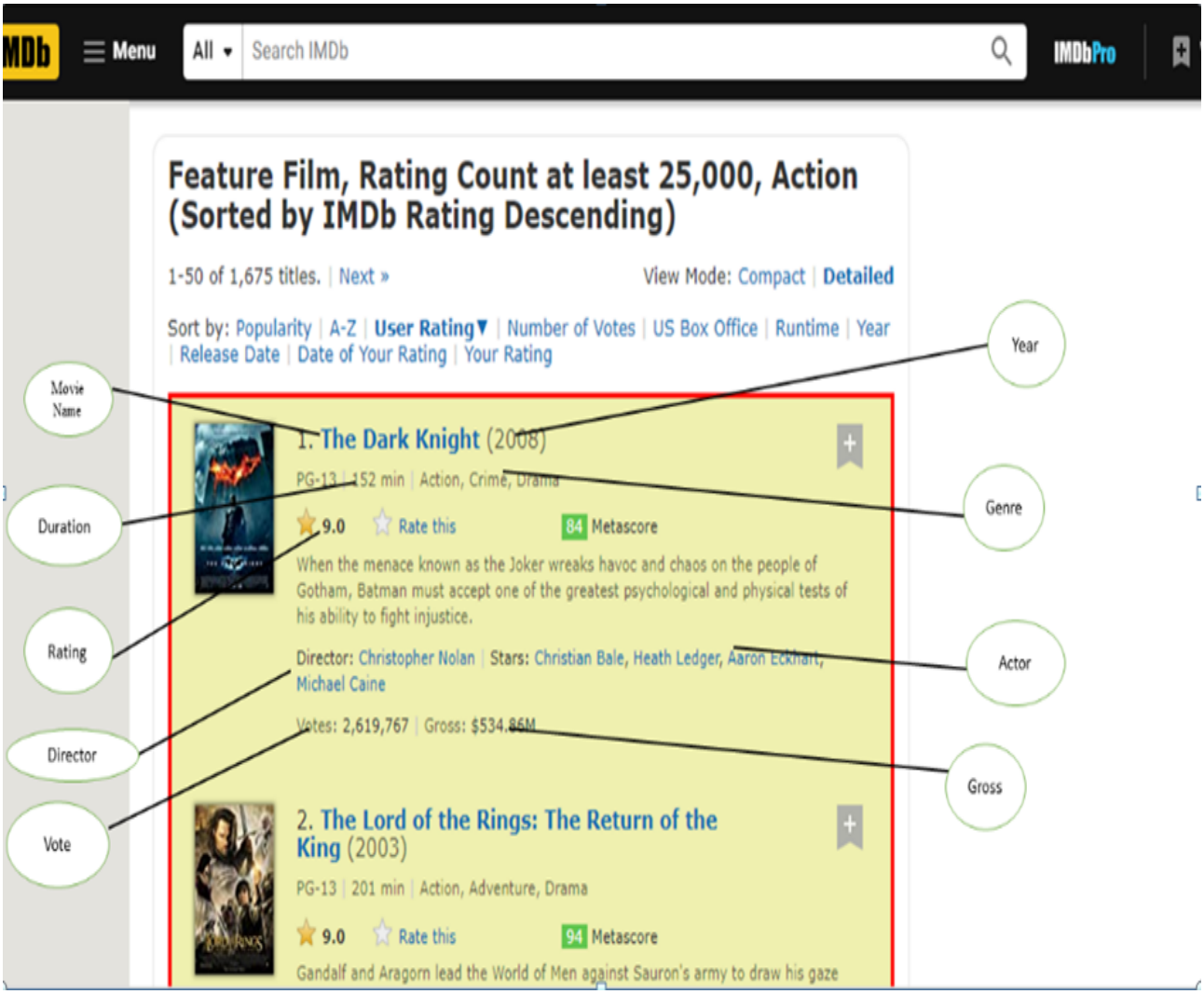

Figure 1: Sample Scraping Source. Screenshot taken from IMBD.com

6 Alorithms

6.1 Sorting Algorithms

[HTML]9B9B9B **Algorithm Name: Insertion Sort**
Description Sorting algorithm with $O(n^2)$ time complexity. The elements of the array is compared sequentially with other elements and then placed at their respective position in the left. It is stable sorting algorithm
Pseudo Code: InsertionSort(A) for i = 0 to A.Length pick = A[i] j = i - 1 while j >= 0 and A[j] >pick: A[j+1] = A[j] j = j - 1 A[j+1] = pick

Code in Python:
def Insertionsort(arr,attr,start,end):
for i in range(start, len(arr)):
pick = arr[i]
j = i - 1
while j >= 0 and arr[j] >pick:
arr[j + 1] = arr[j]
j -= 1
arr[j + 1] = pick
return arr

Time complexity analysis for i = 0 to A.Length ——————>n-times pick = A[i] ——————>n-1 times j = i - 1 ——————>n-1 times while j >= 0 and A[j] >pick: ——————>summation from 0 to n(j) A[j+1] = A[j] ——————>summation from 0 to n(j-1)=n(n+1)/2 j = j - 1 ——————->summation from 0 to n(j-1)=n(n-1)/2 A[j+1] = pick ——————>n-1 times
Calculation: $n+n-1+n-1+n(n+1)/2+n(n-1)/2+n-1=O(n^2)$
Proof of Correctness: **Loop Invariant:** At the start of each iteration of the outer loop the elements of the sub-array A[i..j] are sorted. **Initialization:** Before the start of the loop there is only one element in the sub-array A[i..j] which is trivially sorted

Maintenance:
During the iteration of the loop the element A[j] is placed at its correct position by comparison with the previously sorted element. since the array is previously sorted, we can stop
reach an element lower than A[i] at this point A[i] is placed next to A[j].

Termination:
At the end of the loop the iteration is complete from i to n, thus the array is completely sorted

Strengths:
Only requires constant amount of additional memory
efficient for small data set
simple algorithm
In pace sorting algorithm

Weaknesses:
Does Not perform well on large data set
takes n^2 time complexity

Dry Run on small input:
Original Array:

5	11	2	0	8

Dry run: Step#, state of the array

1:	5	11	2	0	8
2:	2	5	11	0	8
3:	0	2	5	11	8
4:	0	2	5	8	11

Sorted Array

0	2	5	8	11

[HTML]9B9B9B **Algorithm Name: Selection Sort**

Description:
Sorting algorithm, with $O(n^2)$ time complexity. This algorithm iteratively selects the highest element from the array and puts it at its correct position. It is unstable sorting algorithm.

Pseudo code:
SelectionSort(arr):
for i = 1 to arr.Length:
min = i
for j = i to arr.Length:
if arr[j] <arr[min]
min = j
swap arr[min] and arr[i]

Code in Python
```
def SelectionSort(array,attr,start,end):
for i in range(start,end+1):
min=i
for j in range(i,end+1):
if((getattr(array[j],attr))<(getattr(array[min],attr))):
min=j
swap=array[min]
array[min]=array[i]
array[i]=swap
return array
```

Time Complexity Analysis
SelectionSort(arr):
for i = 1 to arr.Length: ——————————>n times
min = i ——————————>n-1 times
for j = i to arr.Length: ——————————>n(n+1)/2
if arr[j] <arr[min] ——————————>n(n+1)/2
min = j
swap arr[min] and arr[i]——————————->n-1 times

Calculation: $(n-1)+n(n+1)/2+n-1=O(n^2)$

Proof of Correctness:

Loop Invariant:
A[min] is the lower than all elements of the sub-array A[i..j]

Initialization:
Before the start of the inner loop the length of sun-array A[i..j] is 0. So A[min] is

[HTML]FFFFFF lower than,all of its elements (trivially).

Maintenance:
During the execution of the loop there are two cases. either A[min] =<A[j] nothing happensin this case, the second case is if A[min] >A[j], in this case the elements are swapped so the loop invariant

Termination:
After the termination of the loop A[min] is the smallest element in the sub-array A[i..n] since j <= n. We place this element at i. In this way when the outer loop goes from 0 to n, the array gets completely sorted

Strengths:
Is simple to use
Is easy to understand
Is in place
No additional temporary storage required

Weaknesses:
Not useful for larger number of items in array
Takes O(n^2) time complexity
Is unstable

Dry Run on small input:
Original array:

9	5	0	7	15

Dry run: Step#, State of array

1:	0	9	5	7	15
2:	0	5	9	7	15
3:	0	5	7	9	15

Sorted Array:

0	5	7	9	15

[HTML]9B9B9B **Algorithm Name: Merge Sort**

Description:
Sorting algorithm with O (n*lg n) time Complexity. It is done on the principal of Divide and Conquer. In this technique the original swapped so the loop invariant

Pseudo code:

```
MergeSort(A, start, end)
if start >= end
return
else
mid = (end - start)/2
MergeSort(A, start, mid)
MergeSort(A, mid+1, start)
Merge(A, start, mid, end)
Merge(A, start, mid, end)
let L = A[start .. mid] and R = A[mid+1..end]
i = j = k = 0
while i <L.Length and j <R.Length
if L[i] <R[j]
A[k] = L[i]
i = i + 1
else
A[k] = R[j]
j = j + 1
k = k + 1
while i <L.Length
A[k] = L[i]
i = i + 1
k = k + 1
while J <R.Length
A[k] = L[j]
j = j + 1
k = k + 1
```

Code in Python:

```
def MergeSort(Arr,attr,p,r):
if(p<r):
q=m.floor((p+r)/2)
MergeSort(Arr,attr,p, q)
MergeSort(Arr,attr,q+1,r)
Merge(Arr,attr,p,q,r)
return Arr
else:
return Arr
```

```
def Merge(Arr,attr,p,q,r):
n1=q-p+1
n2=r-q
L=[]
R=[]
for i in range(n1):
L.append(Arr[p+i])
for j in range(n2):
R.append(Arr[q+j+1])
attribute=getattr(Arr[0], attr)
if(type(attribute)!=str):
a=copy.deepcopy(Arr[4])
setattr(a, attr, sys.maxsize)
L.append(a)
R.append(a)
else:
a=copy.deepcopy(Arr[4])
setattr(a, attr, "zzzzzzzzzz")
L.append(a)
R.append(a)
i=0
j=0
for k in range(p,r+1):
if(getattr(L[i],attr)<getattr(R[j],attr)):
Arr[k]=L[i]
i +=1
else:
Arr[k]=R[j]
j +=1
```

Calculation:

The recurrence equation comes out to be: T(n) = 2T(n/2) + O(n) Solving this we get O(n*lgn)

Time Complexity Analysis:

Merge(A, start, mid, end) ——————>1 Time
let L = A[start .. mid] and R = A[mid+1..end] ——————>1 time
i = j = k = 0 ——————>1 times
while i <L.Length and j <R.Length ——————>n times
if L[i] <R[j] ——————->n-1 times
A[k] = L[i] ——————->1 times
i = i + 1 ——————->1 times
else ——————->n-1 times
A[k] = R[j]
j = j + 1

Proof of Correctness:
We assume that the merge function always works correctly.
We can use mathematical induction to prove the correctness of merge sort.

Base Step:
The array of 1 element is sorted. This is correct (trivially)

Induction Step:
For the induction step we suppose that merge sort correctly sorts an array of n elements, it will divise the array into two sub arrays until it reaches the base case.then it iwill, merge the sorted arrays using merge function which we assumed that it works perfectly therefore, after the recursion is completely the array is sorted and merge sort works correctly.

Strengths:
quicker for larger arrays
has consistent running time

Weaknesses:
Takes extra space to sub-arrays
Relatively slower for small data set
Still runs even if it is sorted

Dry Run on small input:
Original Array:

5	8	3	0	9

Dry run:

1.	5	8	3	0	9
2.	3	0	5	8	9
3.	3	5	0	8	9
4.	0	3	5	8	9

Sorted Array

0	3	5	8	9

Algorithm Name: Quick Sort

Description:
Quicksort partitions an array and then calls itself recursively twice to sort the two is res resulting subarrays. This algorithm is quite efficient for large-sized data sets as its average and worst-case complexity are O(n2). It is unstable sorting algorithm

Pseudo code:
QuickSort(A, l, r)
pivot = Partition(A, l, r)
QuickSort(A, l, pivot - 1)
QuickSort(A, pivot + 1, r)
Partition(A, l, r)
pivot = A[r]
i = l
for j = l to r
if A[j] <pivot
swap A[i] with A[j]
swap A[i] with A[r]
return i

Code in Python:

```
def parititonArray(A,attr,p,r):
x=getattr(A[r],attr)
i=p-1
for j in range(p,r):
if(getattr(A[j],attr)<x):
i +=1
(A[i],A[j])=(A[j],A[i])
(A[i+1],A[r])=(A[r],A[i+1])
return i+1
def QuickSort(A,attr,p,r):
if(p<r):
q=parititonArray(A,attr,p,r)
#print(A)
QuickSort(A,attr,p,q-1)
QuickSort(A,attr,q+1,r)
else:
return
```

Time Complexity Analysis:

QuickSort(A, l, r)
pivot = Partition(A, l, r)
QuickSort(A, l, pivot - 1)
QuickSort(A, pivot + 1, r)
Partition(A, l, r)
pivot = A[r]
i = l
for j = l to r ——————————->n times
if A[j] <pivot ——————————>n-1 times
swap A[i] with A[j] ——————————>n-1 times
swap A[i] with A[r]
return i
for j = l to r
if A[j] <pivot
swap A[i] with A[j]
swap A[i] with A[r]
return i

Calculation:

Time complexity will be O(n2)

Proof of Correctness:

Proof of partition function:

Loop Invariant: All the elements less than pivot are in the sub-array A[l..i-1] and those greater than the pivot are in the sub-array A[i+1..r]

Initialization:

In the beginning the sub-array is empty so all of its elements are less the pivot

Maintenance:

during the execution of the loop if an element less than pivot is found, it is put before i

Termination:

From above we can see that after the termination of the loop, all the elements less than pivot are sent to the sub-array A[l..i-1] which means those greater than the pivot are in the subarray A[i+1..r]

Strengths:

In sorts in place
No additional storage is required

Weaknesses:

The worse case is very similar to average performances of the bubble
Takes a lot of time to sort

Algorithm Name: Counting Sort

Description:
Sorting algorithm with linear time complexity O(n) sorts the elements of an array by counting the number of occurrences of each unique element in the array. The count is stored in an auxiliary array and the sorting is done by mapping the count as an index of the auxiliary array.

Pseudo code:

```
Counting-Sort(A,B,k)
let C[0..k] be a new array
for i=0 to k
C[i]=0
for j=1 to A.length
C[A[j]]=C[A[j]]+1
for i=1 to k
C[i]=C[i]+C[i-1]
for j=A.length downto 1
B[C[A[J]]]=A[j]
C[A[j]]=C[A[j]]-1
```

Code in Python:

```
def CountingSort(A,attr,start,end): #donot accept floating points
A1=[]
for i in range(len(A)):
A1.append(getattr(A[i],attr))
attribute=getattr(A[0], attr)
if(type(attribute)!=str):
maxA = int(max(A1))
minA = int(min(A1))
k = maxA - minA + 1
else:
maxA=max(A1)
minA=min(A1)
k=maxA-minA
B=[0]*len(A)
C=[0]*k
for i in range(len(A1)):
C[getattr(A[i],attr)-minA] +=1
for j in range(1,k):
C[j]=C[j]+C[j-1]
for j in range(len(A)-1,-1,-1):
C[getattr(A[j],attr)-minA] -=1
for i in range(len(A)):
A[i]=B[i]
```

Time Complexity Analysis:

Counting-Sort(A,B,k)

1 let C[0..k] be a new array
2 for i=0 to k ————————->n times
3 C[i]=0 ————————->n-1 times
4 for j=1 to A.length ————————->n times
5 C[A[j]]=C[A[j]]+1 ————————->n-1 times
6 for i=1 to k ————————->n times
7 C[i]=C[i]+C[i-1] ————————->n-1 times
8 for j=A.length downto 1 ————————->n times
9 B[C[A[J]]]=A[j] ————————->n-1 times

Calculation:1+ n-1+n+n-1+n+n+n-1+n+n-1=O(n)

Proof of Correctness:

Strengths:

Has lesser time complexity when number of elements become equal to range _____

Weaknesses:

Has large time complexity
Has large space complexity
Is not a comparison sort

Dry Run on small input:

Original Array:

7 3 5 2

Sorted Array:

2 3 5 7

Algorithm Name: Radix Sort

Description:
Sorting algorithm with linear time complexity O(n) that is used for integers. In Radix sort, there is digit by digit sorting is performed that is started from the least significant digit to the most significant digit.

Code in Python:

```
def RadixSort(A,attr,start,end):
A1=[]
for i in range(len(A)):
A1.append(getattr(A[i],attr))
maxA=max(A1)-min(A1)
d=len(str(maxA))
minA=int(min(A1))
for traverse in range(d):
B=[]
for x in range(10):
B.append([])
bucket=0
for idx in range(len(A)):
div=A1[idx]-minA
for n in range(traverse+1):
bucket=div%10
div=div//10
B[bucket].append(A1[idx])
A2=[]
for i in B:
for j in range(len(i)):
A2.append(i[j])
for i in range(len(A2)):
A[i]=A2[i]
return A
```

Time Complexity Analysis:
Time Complexity Analysis is same as counting sort but it is multiplied with the total characters of greater string

Strengths:
Fast when elements are less in number
Stable sort

Weaknesses:
Not much flexible in comparison to other sorts
Takes more space than quick sort
Not an inplace sorting

Dry Run on small input:_____

Original Array:
170 45 75 90 802

Dry run:

One's Place:
170 90 802 45 75

Ten's Place:
802 45 55 170 90

Hundreds Place:
45 55 90 170 802

Sorted Array:
45 55 90 170 802

Algorithm Name: Bucket Sort

Description:
Sorting algorithm with linear time complexity O(n) that separates the elements into multi groups said to be buckets. Elements in bucket sort are first uniformly divided into groups called buckets, and then they are sorted by any other sorting algorithm.

Pseudo code:

```
Bucket_Sort(A)
B[0,n-1]
n= A.length
for i =0 to n-1
makeB[i] an empty list
for i=1 to n
insert A[i] into B[[nA[i]]]
for i=0 to n-1
sort list B[i] with insertion sort
concatenate the lists B[0],B[1].....B[n-1]
```

Code in Python:

```
def bucketSort(A,start,end):
minA=min(A)
temp=[]
buckets=10
for i in range(buckets):
temp.append([])
realMax=max(A)
RealtiveMaxNum=max(A)-min(A)
for num in A:
if(num!=realMax):
pointing=(num-minA)/RealtiveMaxNum
idx=math.floor(pointing*buckets)
temp[idx].append(num)
for InArr in temp:
InsertionSort(InArr,0,len(InArr)-1)
A=[]
for i in temp:
for j in i:
A.append(j)
A.append(realMax)
return A
```

Time Complexity Analysis:_____

Bucket_Sort(A)
B[0,n-1]
n= A.length
for i =0 to n-1 ——————————->n times
makeB[i] an empty list ——————————->n-1 times
for i=1 to n ——————————->n times
insert A[i] into B[[nA[i]]] ——————————>n-1 times
for i=0 to n-1 ——————————->n times
sort list B[i] with insertion sort ——————————>n-1 times
concatenate the lists B[0],B[1].....B[n-1]

Calculation: n+n-1+n+n-1 = O(n)

Strengths:
Reduces number of comparisons hence faster than a bubble sort

Weaknesses:
Does Not perform in place sorting

Dry Run on small input:

Original Array:
5 1 4 2 8

Dry run:
1. 5 1 4 2 8
2. 1 5 4 2 8
3. 1 4 5 2 8
4. 1 4 2 5 8

Sorted Array:
1 4 2 5 8

6.2 Searching Algorithms

6.2.1 KMP Search

7 User Interface

7.1 Initial GUI Design

7.2 GUI Components

7.2.1 Main Screen

7.2.2 Scrapping Screen

7.3 Implemented GUI

7.3.1 Main Window Page

7.3.2 Scrapping Page

7.3.3 Advanced Search Page

8 Integration

8.1 Problem and their solution

8.2 Overview of Whole Working

8.3 Collaboration

8.4 Bonus Tasks