Electronic Products Management System

CS261 Mid Term Project Requirement Report Details



Project Supervisor

Mr. Samyan Qayyum Wahla

Project Members (Gid: 33)

|  |  |
| --- | --- |
| Member 1 Name: Asadullah Bhatti | Member 1 Registration No.  2020-CS-101 |
| Member 2 Name: Muhammad Umar | Member 2 Registration No.  2020-CS-116 |

Department of Computer Science

University of Engineering and Technology, Lahore

Pakistan

Table of Contents

[Table of Contents 1](#_Toc87026926)

[Motivation: 2](#_Toc87026927)

[Project Description: 2](#_Toc87026928)

[Project Features: 3](#_Toc87026929)

[Business Need: 4](#_Toc87026930)

[Audience: 4](#_Toc87026931)

[Scrapping Purpose: 5](#_Toc87026932)

[Algorithms 6](#_Toc87026933)

[UI Design: 21](#_Toc87026934)

[Technology Stack: 22](#_Toc87026935)

[Files 22](#_Toc87026936)

[Implementation: 22](#_Toc87026937)

[Use Cases: 24](#_Toc87026938)

[Use Case 1(Start Project): 24](#_Toc87026939)

[Use Case 2(Run Scraping): 25](#_Toc87026940)

[Use Case 3(Pause Scraping): 25](#_Toc87026941)

[Use Case 4(Continue Scraping): 26](#_Toc87026942)

[Use Case 5(Linear Sorting): 27](#_Toc87026943)

[Use Case 6(Multiple Sorting): 28](#_Toc87026944)

[Use Case 7(Linear Searching): 29](#_Toc87026945)

[Use Case 8(Binary Searching): 30](#_Toc87026946)

[User Interface Details 32](#_Toc87026947)

[Project Plan 32](#_Toc87026948)

# Motivation:

The motivation behind to choose and complete this project is that it's use in this time or current era. Today life has become so fast and modern. Now it's a digital era, so online shopping for different kind of things has become a trend and electronic products are also among those objects. There are many online websites that are used for online shopping e.g. Amazon, Daraz, Flipkart and Ali Baba etc. People have to buy electronic products but do not know from where they could get the better products. This is where our project comes and it provides an easy way to that person to see the whole data at a single page obtained by the scrapping of the websites that are used for electronic products further he/she can sort the products by any of the provided attributes and see the best product according to. In this way he will not be confused whether there was a much better or much cheaper product.

# Project Description:

*This project is about Electronic Products Management System that’s main purpose is to scrap data from different websites and perform some sorting algorithms on it. The websites that we used to scrap data about electronic products are Amazon, Daraz, Flipkart and AliBaba etc. In this era of online shopping, it is the need of the time to have apps and websites and projects that give them the information of the required products such as their names, their brands, their rating, number of people who ratted, number of questions answered by owners, price and discount etc. User can easily analyse the whole data very easily he/she does not need to visit different websites and to compare them manually. This project not only provide the data of electronic products at a single platform but also provide the features like sorting and searching algorithm. If you are confused that which attribute you should use to sort you can pick more than one attributes at a single time as it also provides the multiple sorting. It has advantages that you can sort data about electronic products in ascending and descending order. There are seven attributes we are going to scrap that are given below:*

* *Name*
* Properties
* Price before discount
* Price after discount
* Discount
* Number of people who rated
* Ratings

These are the attributes we are going to scrap from different websites and sort algorithms that are we are going to use are:

* Insertion Sort
* Selection Sort
* Bubble Sort
* Quick Sort
* Counting Sort
* Bucket Sort
* Hybrid Sort
* Linear Search
* Radix Sort

*This project has also three other sorts of deterministic algorithms and two more algorithms of randomized algorithms to sort data. The algorithms for searching are linear searching algorithm and binary searching algorithm used in project. It provides different filters for searching it is one of the best and useful projects for online shopping of electronic products. Sample of data scrapping from Daraz.*

# Project Features:

This project has following features.

* This project scraps data about electronic products having attributes:

1. *Name*
2. Properties
3. Price before discount
4. Price after discount
5. Discount
6. Number of people who rated
7. Ratings

* Used websites for scrap data are given below:

1. Flip Kart
2. Amazon
3. Daraz

* Sorting algorithms that are used are given below:

1. Insertion Sort
2. Selection Sort
3. Merge Sort
4. Quick Sort
5. Counting Sort
6. Bucket Sort
7. Hybrid Sort
8. Radix Sort

* Searching Algorithms:

1. Linear Sorting
2. Binary Search

# Business Need:

There are the people looking for the best and cheapest electronic product but for this purpose they have to spend a lot of time to check all the products and a lot of efforts to write down all the attributes so that can compare it with other products. To save his time and spend less effort and to get better comparison they need a project or website doing this job as our projects is capable of doing all this so they will use it.

From manufacturers here we mean the companies producing these products which always work on the rule of "Demand and Supply" and they could not wait for the products in the market to finish up then start further production to overcome this problem they have to see customers response which is saved in the ratings attributes. Secondly they have to compare the prices of products which they can do using price attribute. Our project will do all this stuff for them so they will use it making it useful from business point of view.

# Audience:

There are two main users of this project:

1) Customers:

They are the many people that are looking for the best and cheapest electronic products to buy or sell but for this purpose they have to spend a lot of time to check all the products depending upon their interests and a lot of efforts to write down all the attributes so that can compare it with other products. To save his time and spend less effort and to get better comparison so they will definitely use this project.

2) Manufacturers:

From manufacturers here we mean the companies producing these products which always work on the rule of "Demand and Supply" and they could not wait for the products in the market to finish up then start further production to overcome this problem they have to see customers response which is saved in the ratings attributes. Secondly they have to compare the prices of products which they can do using price attribute. They will also definitely use it

# Scrapping Purpose:

Al

# Algorithms

|  |  |
| --- | --- |
| Name | Insertion Sort |
| Description | Insertion sort is a simple sorting algorithm. The basic principle of insertion sort is that given data is sorted by checking or consuming one input element each repetition, and produced a sorted output list. At each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there. It repeats until no input elements remain. This type of algorithm is used by most card players when sorting the cards in their hands. When holding a hand of cards, players will often check or scan their cards from left to right, looking for the first card that is out of place. When they see a card that is not in order among cards they sort the cards exactly the same way as insertion sort. The insertion sort achieves a sorted data set by identifying an element that out of order relative to the elements around it, removing it from the list, shifting elements up one place and then placing the removed element in its correct location. One more real-world example of insertion sort is how tailors arrange shirts in a cupboard, they always keep them in sorted order of size and thus insert new shirts at the right position very quickly by moving other shirts forward to keep the right place for a new shirt. Time complexity of best case of Insertion sort is O(n). It is much less efficient on large lists than more advanced algorithms such as quicksort, heapsort, or merge sort. However, insertion sort provides several advantages: Simple implementation. |
| Pseudo Code | INSERTION-SORT(A)  1 for j D 2 to A:length  2 key=A[j]  3 // Insert A[j] into the sorted sequence A[1 .. j-1]  4 i=j-1  5 while i>0 and A[j]> key  6 A[j+1]= A[j]  7 i=i-1  8 A[i+1]=key |
| Python Code | def sortArray(arr):  for i in range(1 , len(arr)):  key=arr[i]  j=i-1  while(j>=0 and arr[j]>key):  arr[j+1]=arr[j]  j=j-1  arr[j+1]=key    Input=[5, 7, -8, 9, 10, 4, -7, 0, -12, 1, 6, 2, 3, -4, -15, 12]  sortArray(Input)  print("The output is: "+str(Input)) |
| Time Complexity | Best Case: Ω (n)  Average Case: θ (n^2)  Worst Case: O(n^2) |
| Dry run on small inputs |  |
| Strengths | 1. Simple and really good foe small lists 2. It is stable sort 3. It sorts by rearrange arrays into sorted order in-place so no need of other spaces and space requirement is minimum. |
| Weaknesses | 1. It takes n-squared processing steps to sort 2. It takes large time so not good for large inputs 3. It checks the whole list each time |

|  |  |
| --- | --- |
| Name | Merge Sort |
| Description | Merge sort is an efficient, general-purpose, and comparison-based sorting algorithm. The principle of merge sort is divide the given list of data into sub-lists, each list containing elements that were present in given unsorted list. Now, take adjacent pairs of two sub-lists (which contain one element in list) and merge them to form a list of 2 elements. That will now convert the sub-lists into lists of size 2. Now we repeat the process until one sorted output list is obtained that will be our sorted list in this way, given data or array is sorted by using merge sort algorithm. Merge sort also known as Divide and Conquer algorithm. Time complexity of Merge Sort is O(n\*Log n) in best case. Time complexity is also same in worst case and average case i.e. O(n\*Log n). Considering average time complexity of insertion sort algorithm and Merge Sort algorithm, we can say that Merge Sort is efficient in terms of time and Insertion Sort is efficient in terms of space |
| Pseudo Code | MERGE(A, p, q, r)  1 n1 = q -p + 1  2 n2 = r - q  3 let L[1 .. n1 +1] and R[1 .. n2 +1] be new arrays  4 for i = 1 to n1  5 L[i]=A[p+i-1]  6 for j = 1 to n2  7 R[j]=A[q+j]  8 L=[n1 +1]=infinity  9 R=[n2 +1]=infinity  10 i = 1  11 j = 1  12 for k = p to r  13 if L[i]<=R[j]  14 A[k]=L[i]  15 i=i+1  16 else A[k]=R[j]  17 j = j + 1  **OR**  MergeSort(arr[], l, r)  If r > l  1. Find the middle point to divide the array into two halves:  middle m = l+ (r-l)/2  2. Call mergeSort for first half:  Call mergeSort(arr, l, m)  3. Call mergeSort for second half:  Call mergeSort(arr, m+1, r)  4. Merge the two halves sorted in step 2 and 3:  Call merge(arr, l, m, r) |
| Python Code | def merge\_sort(arr):  if(len(arr)>1):  left\_arr=arr[:len(arr)//2]  right\_arr=arr[len(arr)//2:]    # recursion  merge\_sort(left\_arr)  merge\_sort(right\_arr)  # merge  i=0  j=0  k=0    while(i<len(left\_arr) and j<len(right\_arr)):  if(left\_arr[i]<right\_arr[j]):  arr[k]=left\_arr[i]  i=i+1  else:  arr[k]=right\_arr[j]  j=j+1  k=k+1    while(i<len(left\_arr)):  arr[k]=left\_arr[i]  i=i+1  k=k+1    while(j<len(right\_arr)):  arr[k]=right\_arr[j]  j=j+1  k=k+1    arr=[4, 1, 3, 5, 9, 7, 8 ,10 , 12]  merge\_sort(arr)  print(arr) |
| Time Complexity | Best Case: Ω ( nlog(n))  Average Case: θ (nlog(n))  Worst Case: O( nlog(n)) |
| Dry run on small inputs |  |
| Strengths | 1. It is stable sort 2. It takes less time to complete the sorting 3. It is specially good for large input lists |
| Weaknesses | 1. Slower comparative to the other sort algorithms for smaller tasks 2. It is divide and conquer so it takes same time depending on n which is length of list and cannot take the advantage of already in order elements. 3. It uses extra spaces to store the sub elements and do not rearranges in actual list. |

|  |  |
| --- | --- |
| Name | Selection Sort |
| Description | Selection Sort is in place comparison sorting algorithm. The basic principle of selection sort is that, this sorting algorithm, iterates through the array and finds the smallest number in the array and swaps it with the first element if it is smaller than the first element. Next, it goes on to the second element and check that element. It repeats its process until the given data or list of data is sorted completely. In this way, array is sorted using selection sort. Consider the array [10,5,2,1], The first element is 10. The next part we must find the smallest number from the remaining array. The smallest number from 5 2 and 1 is 1. So, we replace 10 by 1. The new array is [1,5,2,10] Again, this process is repeated. Finally, we get the sorted array as [1,2,5,10]. Although selection sort is good to sort data but it is very slower than merge sort because the merge-sort algorithm copies elements back and forth to a temporary array during each merge. So, we expect a merge sort to be about 40 times faster than a selection sort. Time complexity of selection sort is O(n2) time complexity, which makes it inefficient on large lists, and generally performs worse than the similar insertion sort. |
| Pseudo Code | procedure selection sort  list : array of items  n : size of list  for i = 1 to n - 1  /\* set current element as minimum\*/  min = i  /\* check the element to be minimum \*/  for j = i+1 to n  if list[j] < list[min] then  min = j  end if  end for  /\* swap the minimum element with the current element\*/  if indexMin != i then  swap list[min] and list[i]  end if  end for  end procedure |
| Python Code | def sortArray(arr):  for i in range(len(arr)-1):  min=i  for j in range(i+1, len(arr)):  if(arr[min]>arr[j]):  min=j  temp=arr[i]  arr[i]=arr[min]  arr[min]=temp    Input=[5, 8, 9, 3, 5, 7, 1, 3, 4, 9, 3, 5, 1, 8, 11, 4,2]  sortArray(Input)  print("The output is: "+str(Input)) |
| Time Complexity | Best Case: Ω (n^2)  Average Case: θ (n^2)  Worst Case: O( n^2) |
| Dry run on small inputs |  |
| Strengths | 1. It sorts by rearrange arrays into sorted order in-place so no need of other spaces and space requirement is minimum. 2. It is very good and fast for small input lists 3. It can take full advantage of already sorted lists as in that case it will not swap elements. |
| Weaknesses | 1. It is not stable 2. It has a poor efficiency when dealing with large lists 3. It takes n-squared processing steps to sort |

|  |  |
| --- | --- |
| Name | Bubble Sort |
| Description | Bubble sort sometimes referred to as sinking sort. The basic principle of bubble sort is that look at the first number in the given list then compare the current number with the next number. If the next number is smaller than the current number, move to the next number along in the given list and make this the current number. Repeat the given process until the last number in the list has been reached. In this way the array is sorted using bubble sort algorithm. The bubble sort is better than merge sort in practice for small set of data, but as size of input data increases, the performance of bubble sort suddenly drop down and the exact opposite behavior I found with merge sort. Bubble Sort is an easy-to-implement, stable sorting algorithm with a time complexity of O(n²) in the average and worst cases – and O(n) in the best case. |
| Pseudo Code | procedure bubbleSort( list : array of items )  loop = list.count;  for i = 0 to loop-1 do:  swapped = false  for j = 0 to loop-1 do:  /\* compare the adjacent elements \*/  if list[j] > list[j+1] then  /\* swap them \*/  swap( list[j], list[j+1] )  swapped = true  end if  end for  /\*if no number was swapped that means array is sorted now, break the loop.\*/  if(not swapped) then  break  end if  end for  end procedure return list |
| Python Code | def sortArray(arr):  for i in range(len(arr)):  flag=0  for j in range(len(arr)-1-i):  if(arr[j]>arr[j+1]):  temp=arr[j]  arr[j]=arr[j+1]  arr[j+1]=temp  flag=1  if(flag==0):  break    Input=[5, 7, -8, 9, 10, 4, -7, 0, -12, 1, 6, 2, 3, -4, -15, 12,30]  sortArray(Input)  print("The output is: "+str(Input)) |
| Time Complexity | Best Case: Ω (n)  Average Case: θ (n^2)  Worst Case: O( n^2) |
| Dry run on small inputs |  |
| Strengths | 1. It is a stable sort 2. It sorts by rearrange arrays into sorted order in-place so no need of other spaces and space requirement is minimum. 3. It is very simple and easy to understand and write. It’s not logically complicated. |
| Weaknesses | 1. It takes n-squared processing steps to sort 2. It is not good for huge lists as it compares key value with the whole list elements 3. It is a slow process and take a lot of time to sort especially when the values coming after key value will be smaller. |

|  |  |
| --- | --- |
| Name | Hybrid Sort |
| Description | Hybrid means a thing made by combining two different elements. As name suggests hybrid sort algorithm is an algorithm that combines two or more other algorithms that solve the same problem, either choosing one (depending on the data) or switching between them throughout the algorithm. This is generally done by combining each of the desired features so that overall algorithm is the better than individual components. Hybrid algorithm" does not refer to simply combining multiple algorithms to solve a different problem – many algorithms can be considered as combinations of simpler pieces – but only to combining algorithms that solve the same problem, but differ in other characteristics, notably performance. When the total number of elements is below some threshold (perhaps ten elements), switch to a non-recursive sorting algorithm such as that performs fewer swaps, comparisons, or other operations on such small arrays. |
| Pseudo Code |  |
| Python Code |  |
| Time Complexity | Best Case: Ω ( )  Average Case: θ ( )  Worst Case: O( ) |
| Dry run on small inputs |  |
| Strengths |  |
| Weaknesses |  |

|  |  |
| --- | --- |
| Name | Counting Sort |
| Description | Counting sort is a sorting algorithm in which it sorts on the basis of keys in the specific range. The basic principle of counting sort is that it counts the number of objects having specific key values and then doing some algorithm operations to calculate position of every element in the given array. After that it placed every element according to the position that is calculated by arithmetic operations. After doing that array is generated and that array is sorted array of given unsorted array. In this way you can sort any array using counting array. Counting sort works has better performance because it sorts elements that are in a range of values. So, to apply counting sort, you need to ensure that the elements in the array are in a range k. The time complexity of counting sort algorithm is **O(n+k)** where n is the number of elements in the array and k is the range of the elements. |
| 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 | def countingSort(arr):  brr=[0 for i in range(len(arr))]  maximum=int(max(arr))  count\_size=maximum+1  count=[0 for i in range(count\_size)]    for i in range(len(arr)):  count[arr[i]]=count[arr[i]]+1    for i in range(1,len(count)):  count[i]=count[i]+count[i-1]    for i in range(len(arr)):  count[arr[i]]=count[arr[i]]-1  brr[count[arr[i]]]=arr[i]    return brr    arr = [-5, -10, 0, -3, 8, 5, -1, 10]  minimum=int(min(arr))  for i in range(len(arr)):  arr[i]=arr[i]-minimum  end = countingSort(arr)  for i in range(len(end)):  end[i]=end[i]+minimum    print("Sorted character array is " + str(end)) |
| Time Complexity | Best Case: Ω (n+k)  Average Case: θ (n+k )  Worst Case: O(n+k) |
| Dry run on small inputs |  |
| Strengths | 1. It is not a comparison based algorithm that why it has linear time complexity. 2. It is a stable sort |
| Weaknesses | 1. If the range of elements is too large then it takes a lot of space to counts the number of that element occurring. 2. It can only count the integers as the count array can only contain integers. 3. Even with small range there would be many empty places in count array |

|  |  |
| --- | --- |
| Name | Quick Sort |
| Description | Quicksort is an in-place sorting algorithm. The basic principle of quick sort is it is a technique that continuously divide the list into two parts and move the lower elements to one side and higher elements to other side. Then we pick a number as a pivot and arrange elements in such a way that elements lower than the pivot comes to left side of the pivot and elements higher than the pivot comes to right side of the pivot point. It then picks a pivot for the left side and moves those items to left and right of the pivot and continues the pivot picking and dividing until there is only one item left in the group. It then proceeds to the right side and performs the same operation again. It repeatedly performs the operations until the given array is sorted. In this way array is sorted using quick sort. Time complexity of quick sort of best case O(nlogn), worst cases O(n^2), average case O(nlogn). When implemented well, it can be somewhat faster than merge sort and about two or three times faster than heapsort |
| Pseudo Code | quickSort(arr[], low, high)  {  if (low < high)  {  /\* pi is partitioning index, arr[pi] is now at right place \*/  pi = partition(arr, low, high);  quickSort(arr, low, pi - 1); // Before pi  quickSort(arr, pi + 1, high); // After pi  }  }  partition (arr[], low, high)  {  // pivot (Element to be placed at right position)  pivot = arr[high];  i = (low - 1) // Index of smaller element and indicates the  // right position of pivot found so far  for (j = low; j <= high- 1; j++)  {  // If current element is smaller than the pivot  if (arr[j] < pivot)  {  i++; // increment index of smaller element  swap arr[i] and arr[j]  }  }  swap arr[i + 1] and arr[high])  return (i + 1)  } |
| Python Code | def partition(arr, low, high):  i = (low-1)  pivot = arr[high]    for j in range(low, high):  if arr[j] <= pivot:  i = i+1  a=arr[i]  arr[i]=arr[j]  arr[j]=a    b=arr[i + 1]  arr[i+1]=arr[high]  arr[high]=b  return (i+1)  def quickSort(arr, low, high):  if len(arr) == 1:  return arr  if low < high:  pi = partition(arr, low, high)  quickSort(arr, low, pi-1)  quickSort(arr, pi+1, high)  arr=[]  n=int(input("Enter number of elements of array: "))  for i in range(n):  x=int(input("Enter "+str(i+1)+" element : "))  arr.append(x)  n = len(arr)  quickSort(arr, 0, n-1)  print(arr) |
| Time Complexity | Best Case: Ω ( nlog(n))  Average Case: θ (nlog(n))  Worst Case: O( n^2) |
| Dry run on small inputs |  |
| Strengths | 1. It sorts by rearrange arrays into sorted order in-place so no need of other spaces and space requirement is minimum. 2. It requires only n (log n) time to sort n items. 3. It is also a divide and conquer algorithm. |
| Weaknesses | 1. Quick sort is not a stable sort 2. It is divide and conquer so cannot take advantage of already sorted elements. 3. It also take quadratic steps in worst case |

|  |  |
| --- | --- |
| Name | Radix Sort |
| Description | Radix sort is a non-comparative sorting algorithm. It avoids comparison by creating and distributing elements into buckets according to their radix. The basic principle of radix sort is that it sorts elements in an array one by one starting from least significant element to most significant element. Suppose if you want to sort 10 elements in ascending order using radix sort, first sort the digit of unit place. After that sort the tenth place digit after that hundred place digit and so on. In this way array or data is sorted using radix sort. The time complexity of radix sort is given by the formula,T(n) = O(d\*(n+b)), where d is the number of digits in the given list, n is the number of elements in the list, and b is the base or bucket size used, which is normally base 10 for decimal representation. |
| Pseudo Code | It is almost same as the counting and will work on the same principle the only difference is it checks and sorts according to the numbers occurring at different decimal places |
| Python Code | def countingSort(arr , pos):  minimum=int(min(arr))  for i in range(len(arr)):  arr[i]=arr[i]-minimum    n=len(arr)  brr=[0 for i in range(len(arr))]  count=[0 for i in range(10)]    for i in range(len(arr)):  index = arr[i] // pos  count[index % 10] =count[index % 10]+1    for i in range(1,len(count)):  count[i]=count[i]+count[i-1]    i = n - 1  while i >= 0:  index = arr[i] // pos  count[index % 10] -= 1  brr[count[index % 10]] = arr[i]  i -= 1    for i in range(0, len(arr)):  arr[i] = brr[i]    for i in range(len(arr)):  arr[i]=arr[i]+minimum    def radixSort(arr):  max1 = max(arr)  exp = 1  while max1 / exp > 0:  countingSort(arr, exp)  exp =exp\*10    arr = [110, 45, 65, 50, 90,602, 24, 2, 66]  radixSort(arr)    print(arr) |
| Time Complexity | Best Case: Ω ( nk)  Average Case: θ (nk )  Worst Case: O( nk) |
| Dry run on small inputs |  |
| Strengths | 1. It also solves the issue of counting sort creating a large count array for large range because it compares numbers decimal place wise so the length of count is fixed to 10. 2. It is a stable sort |
| Weaknesses | 1. It takes more space than quick sort 2. Its constant is larger than other sorts |

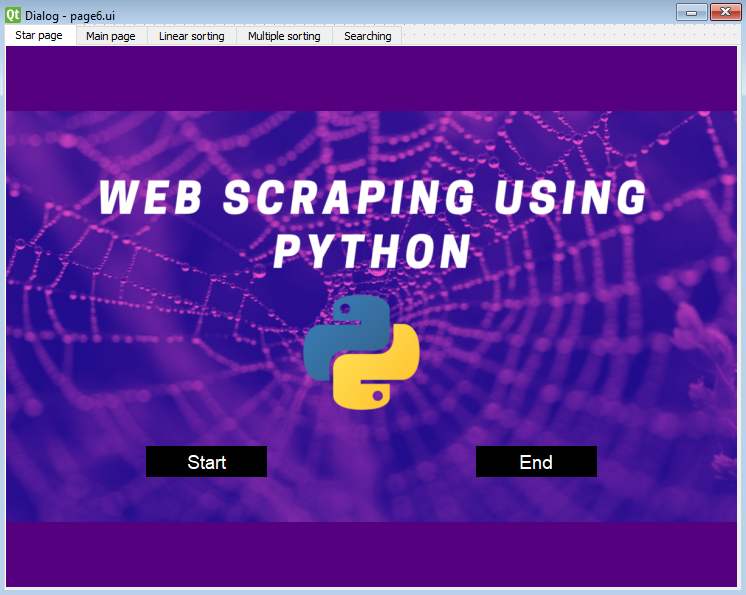
|  |  |
| --- | --- |
| Name | Bucket Sort |
| Description | Bucket Sort is a comparison-type algorithm. It is also known as bin sort. The basic principle of bucket sort is that first of all we make a list of empty buckets and initialized for each element in the array. Then iterate through the bucket list and insert elements from the array that is given. Where each element that inserts depends upon the list and largest element in it. After that sort each non-empty bucket with insertion sort or some other sorting algorithm according to require data. After that visits the bucket in order, once all buckets are filled in sorted manner all the data or array is also sorted. In this way array is sorted using bucket sort. Time Complexity of the bucket sort or bin sort is O(n + k) for best case and average case and O(n^2) for the worst case. |
| Pseudo Code | bucketSort(arr[], n)  1) Create n empty buckets (Or lists).  2) Do following for every array element arr[i].  .......a) Insert arr[i] into bucket[n\*array[i]]  3) Sort individual buckets using insertion sort.  4) Concatenate all sorted buckets. |
| Python Code | def insertionSort(brr):  for i in range(1 , len(brr)):  key=brr[i]  j=i-1  while(j>=0 and brr[j]>key):  brr[j+1]=brr[j]  j=j-1  brr[j+1]=key  return brr  def bucketSort(output):  arr = []  count = 10  for i in range(count):  arr.append([])    for j in x:  index\_b = int(count \* j)  arr[index\_b].append(j)    for i in range(count):  arr[i] = insertionSort(arr[i])    k = 0  for i in range(count):  for j in range(len(arr[i])):  output[k] = arr[i][j]  k =k+1  return output  x = [0.897, 0.565, 0.656,0.1234, 0.665, 0.3434]  print("Sorted Array is")  print(bucketSort(x)) |
| Time Complexity | Best Case: Ω ( n+k)  Average Case: θ (n+k )  Worst Case: O( n^2) |
| Dry run on small inputs |  |
| Strengths | 1. It is a stable sort 2. It can be used as external sorting algorithm 3. It makes new buckets for each so processed independently |
| Weaknesses | 1. It is not applicable to all data as the bucket scheme should be good. 2. It could get disturbed with a light change in input credentials. |

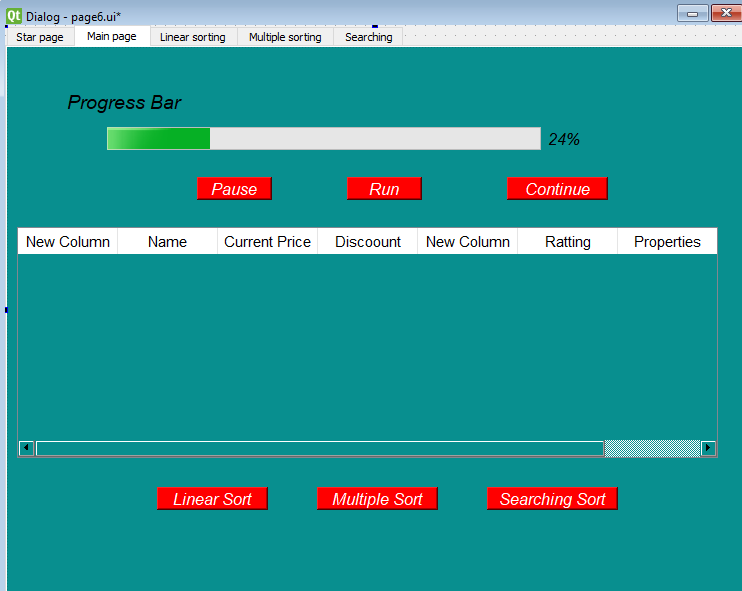
**Searching Algorithms:**

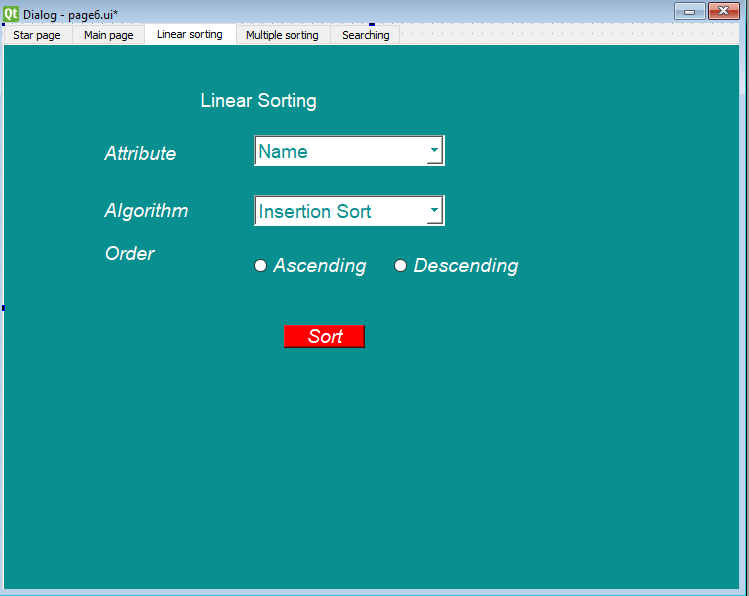
|  |  |
| --- | --- |
| Name | Linear Search |
| Description | Linear search is a very simple search algorithm. In this type of search, a sequential search is made over all items one by one. Every item is checked and if a match is found then that particular item is returned, otherwise the search continues till the end of the data collection. As a real world example, pickup the nearest phonebook and open it to the first page of names. We're looking to find the first "Smith". Look at the first name. Is it "Smith"? Probably not (it's probably a name that begins with 'A'). Now look at the next name. Is it "Smith"? Probably not. Keep looking at the next name until you find "Smith". |
| Pseudo Code |  |
| Python Code |  |
| Time Complexity | O(n) |
| Dry run on small inputs |  |
| Strengths | 1. It finds without sorting |
| Weaknesses | 1. It is slower than binary 2. It is sequential so check each and every element |

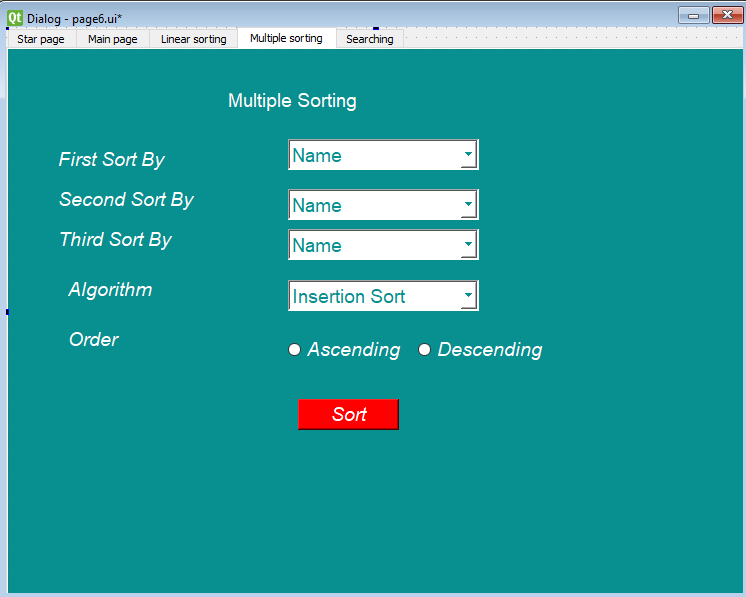
|  |  |
| --- | --- |
| Name | Binary Search |
| Description |  |
| Pseudo Code |  |
| Python Code |  |
| Time Complexity | O(nlog(n)) |
| Dry run on small inputs |  |
| Strengths | 1. It is faster than linear 2. It is an optimal searching algorithm 3. It is not sequential so can find number quickly |
| Weaknesses | 1. It requires already sorted array to find the number |

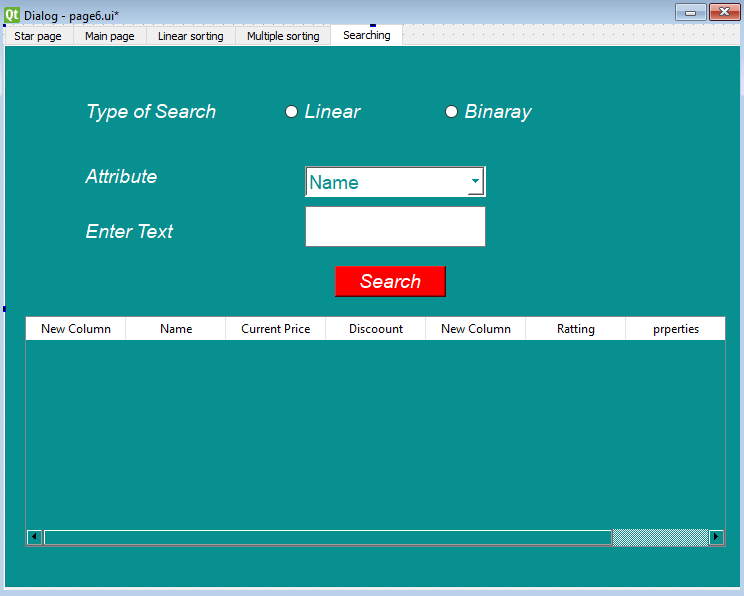
# UI Design:











# Technology Stack:

|  |  |
| --- | --- |
| Language | Python |
| GUI Designer | Qt designer |
| Libraries | Libraries that are include :   * Selenium * Request * Beautiful Soup * Pandas * Csv |
| IDEs | IDEs we used is:   * JupyterNotebook * Visual Studio * Pycharm |

# Files

|  |  |
| --- | --- |
| **File Name** | Asad.py |
| **Type** | Python File |
| **Functionality** | It is the main file in which all the code is written and is running |

|  |  |
| --- | --- |
| **File Name** | Test.csv |
| **Type** | csv File |
| **Functionality** | It contains the data which is scrapped and again the program read data from this file |

# Implementation:

# Use Cases:

There are seven cases that are being used in this project that are:

* Start Project
* Run Scraping
* Pause Scraping
* Continue Scraping
* Linear Sorting
* Multiple Sorting
* Linear Searching
* Binary Searching

## Use Case 1(Start Project):

|  |  |
| --- | --- |
| Use Case ID | U01 |
| Name | Start Project |
| Actor | User |
| Button name in GUI | Start |
| Description | It will move to main page if Start button is clicked |
| Button name in GUI | End |
| Description | It will end the program if End button is clicked |
| GUI |  |

## Use Case 2(Run Scraping):

|  |  |
| --- | --- |
| Use Case ID | U02 |
| Name | Run Scrapping |
| Actor | User |
| Button name in GUI | Run |
| Description | In this case scrapping will start from websites whose Url we have given in project and update table accordingly and it will also save data about data scrapping in n csv file having name ‘Test.csv’ |
| GUI |  |

## Use Case 3(Pause Scraping):

|  |  |
| --- | --- |
| Use Case ID | U03 |
| Name | Pause Scrapping |
| Actor | User |
| Button name in GUI | Pause |
| Description | In this case scraping stopped and table will show how much data is scrapped till now. |
| GUI |  |

## Use Case 4(Continue Scraping):

|  |  |
| --- | --- |
| Use Case ID | U04 |
| Name | Continue Scrapping |
| Actor | User |
| Button name in GUI | Continue |
| Description | In this case scraping continued where it is stopped and update the table accordingly and continue to stop till we don’t pause the scrapping and it will also update file where we are saving data about electronic products. |
| GUI |  |

## Use Case 5(Linear Sorting):

|  |  |
| --- | --- |
| Use Case ID | U05 |
| Name | Linear sorting |
| Actor | User |
| Button name in GUI | Sort |
| Description | In this case we sort data by any attributes i.e name, price before discount, price after discount and other attributes that are given in scrapping. We can also perform different sorting algorithms in ascending and descending i.e. Insertion Sort, Selection Sort, Merge Sort and other sorting algorithms. |
| GUI |  |

## Use Case 6(Multiple Sorting):

|  |  |
| --- | --- |
| Use Case ID | U06 |
| Name | Multiple sorting |
| Actor | User |
| Button name in GUI | Sort |
| Description | In this case we sort data by using different attributes at a same time i.e name, price before discount, price after discount and other attributes that are given in scrapping. We can also perform different sorting algorithms in ascending and descending i.e. Insertion Sort, Selection Sort, Merge Sort and other sorting algorithms. |
| GUI |  |

## Use Case 7(Linear Searching):

|  |  |
| --- | --- |
| Use Case ID | U07 |
| Name | Linear Searching |
| Actor | User |
| Description | In this case we search data linearly by using any attributes at a same time i.e name, price before discount, price after discount and other attributes that are given in scrapping. After searching it will show all possible results in given table. |
| GUI |  |

## Use Case 8(Binary Searching):

|  |  |
| --- | --- |
| Use Case ID | U08 |
| Name | Binary Searching |
| Actor | User |
| Description | In this case we search data by binary search using any attributes at a same time i.e name, price before discount, price after discount and other attributes that are given in scrapping. After searching it will show all possible results in given table. |
| GUI |  |

# User Interface Details

In this section, all the components that we used in the whole project is given below:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Use Case Id | Label | Drop Down | Table | Buttons/  Radio | Text Box | ProgressBar |
| U01 | 1 | 0 | 0 | 2 | 0 | 0 |
| U02 | 1 | 0 | 1 | 6 | 0 | 1 |
| U03 | 1 | 0 | 1 | 6 | 0 | 1 |
| U04 | 1 | 0 | 1 | 6 | 0 | 1 |
| U05 | 6 | 2 | 0 | 3 | 0 | 0 |
| U06 | 8 | 4 | 0 | 3 | 0 | 0 |
| U07 | 5 | 1 | 1 | 3 | 1 | 0 |
| U08 | 5 | 1 | 1 | 3 | 1 | 0 |

# Project Plan

|  |  |
| --- | --- |
| Work assigned | Member Registration number |
| First project proposal(which was rejected) | 2020-CS-116 |
| Project proposal | 2020-CS-101 also used data from first file |
| Algorithm report descriptions and dry runs | 2020-CS-116 |
| All other things of Algorithms   1. Strengths 2. Weaknesses 3. Pseudo codes 4. python codes 5. time complexity | 2020-CS-101 |
| Scraping from Daraz | 2020-CS-116 |
| Scraping from flipkart | 2020-CS-101 |
| GUI  page1.ui, page2.ui, page3.ui , page4.ui and page5.ui | 2020-CS-101 and 2020-CS-116 both made these files in hostel |
| Again GUI | 2020-CS-101  As jupyter was unable to run many files so made a page6.ui using tab widget and copy pasted upper files |
| Integration | Both members |
| Final reports | Both members |

Problems and solutions:

|  |  |
| --- | --- |
| Problems | Their Solutions |
| First of all the interface of QT designer was very difficult to use. | We watched a lot of tutorials and worked really hard to understand and work on it. |
| By running the code of conversion of .ui file to python file usually kernel dead | Converted .ui file to .py file by using cmd |
| Calling the scraping for different links | Made a function for it |
| While returning the list from a function “call before assignment” issue came | Made the list global |
| Sorting on strings | Still working on it |