GameBase Web Scraping

CS261- Mid Term Project Report



Project Supervisor

Mr. Samyan Qayyum Wahla

Project Members(GID59)

| Mian Zain-ul-Tahir | 2020-CS-57 |
| --- | --- |
| Muhammad Aqib | 2020-CS-92 |

Department of Computer Science

University of Engineering and Technology

Lahore, Pakistan

**Table of Contents**

[**Project Description:**](#_gjdgxs) **4**

[Introduction](#_30j0zll) 4

[Project’s purpose](#_3znysh7) 4

[Scope](#_2et92p0) 4

[Motivation](#_tyjcwt) 4

[Expectations](#_3dy6vkm) 5

[**Algorithms:**](#_t7tzzwa0t3kg) **6**

[Insertion Sort:](#_tvpr8w5l27hj) 6

[Description](#_mjzpqvun1hn) 6

[Pseudo Code](#_ct0aldanaeta) 6

[Code in Python](#_pxr17xizyk6q) 6

[Strengths](#_q9wqfy9j6mqg) 7

[Weaknesses](#_vzxtgrcs7xtp) 7

[Quick Sort:](#_55xkhpg4kopg) 7

[Description](#_eh6pnlpii2s) 7

[Pseudocode](#_ipl0euis48di) 7

[Code in python](#_bprq0epmasqi) 8

[Strengths](#_e3oyzrcxaha5) 8

[Weaknesses](#_63y01ekrlva1) 8

[Time Complexity](#_l0lbrra7742u) 8

[Selection Sort:](#_vg6kqju5yals) 9

[Description](#_pbr69za6d0yp) 9

[Pseudocode](#_or209g2s8r4o) 9

[Code in python](#_4a2ggzxaxbds) 9

[Strengths](#_e6d1dzjuh2sh) 10

[Weaknesses](#_f6dzwwyv5yyk) 10

[Time Complexity](#_hpd7zpkfy78q) 10

[Merge Sort:](#_i5voica3sc1i) 10

[Description:](#_khi1z01yt9ni) 10

[Pseudocode](#_d0n4wrte8b62) 10

[Code in python](#_xkeyzx18mx2n) 11

[Strengths](#_rr7mph50ut3m) 11

[Weaknesses](#_bwehr0g35svj) 11

[Time Complexity](#_7sstz0l1iily) 11

[Bubble Sort:](#_uj2s0f2cxmk9) 12

[Description](#_2cmshumgsqvc) 12

[Pseudocode](#_c9hat2pzott2) 12

[Code in python](#_315e5l4tw4z0) 12

[Strengths](#_yt5zv6rbudh8) 12

[Weaknesses](#_oiwtnujhfzcr) 12

[Time Complexity](#_h096eu5j1z3v) 13

[Counting Sort:](#_agdleuy0t6rn) 13

[Description](#_ogay1erfs65d) 13

[Pseudocode](#_djvnyjxom1el) 13

[Code in python](#_thkuvsfgupac) 13

[Strengths](#_o5c8ayt3zumz) 14

[Weaknesses](#_lnyn2lbmuz5l) 14

[Time Complexity](#_uqhimf7gedwr) 14

[Radix Sort:](#_lb3wgw8h7k6i) 14

[Description](#_gmobm4l7zm7s) 14

[Pseudocode](#_mirexe9336an) 14

[Code in python](#_3otmnapez022) 14

[Strengths](#_h1nirsl7c4vm) 15

[Weaknesses](#_uantlh91mawf) 15

[Time Complexity](#_v26ilbhcm8fq) 15

[Bucket Sort:](#_q3c6ptiditen) 16

[Description](#_1ij225y5mvz8) 16

[Pseudocode](#_cpey7sdsodzq) 16

[Code in python](#_p7y9w2ty89az) 16

[Strengths](#_d1nqxv4cbhxq) 16

[Weaknesses](#_q56fa3onbs7z) 17

[Time Complexity](#_ohooe4rwaceb) 17

[**Interfaces for project:**](#_iutum47nuypm) **18**

[Main Page](#_t4m57zjzeq9w) 18

[Scrapping Page](#_px608fsvpay0) 19

[Sorting Page](#_z6ed7swe65tn) 20

# **Project Description:**

## **Introduction**

This project will allow the user to manage records of 1 million entities in such a way that users will be able to start, pause and stop the scraping of data from the internet. Moreover users will have full control of presentation of data in the list i.e. user can sort data according to his need. User can choose a specific algorithm from the algorithms available in the software, using which he wants to sort data as he/she wants. User will also be provided with a progress bar from which he can see the progress of scraping and get a better idea of scrapping progress so he can easily decide where to pause or stop.

We will be providing fully advanced filters for sorting and searching for each column and our software will also be able to apply multi column sorting. As far as UI is concerned fully functional and advanced GUI will be provided to users therefore the software will be interactive, eye catching and easy to use. For Graphical User Interface we will use PyQt5 (a library of python).

For this project we will scrape data from different websites and we will be scraping about seven attributes for each entity. Attributes will be Name, Developer, Size, Category, Rating, Downloads, Price.

After scraping all these attributes from the web our program will store it in a csv file and also display on screen from where user can perform sorting and searching on data according to his/her need i.e. user will sort data using any algorithm he wants from the given algorithms in the program. Advanced filters for string columns will be implemented such as contains, end with, starts with etc. UI will have the option for searching based on each column. After sorting the column, time will be displayed in milliseconds. Users will have the option to apply composite filters.

## **Project’s purpose**

The main purpose of this project is to learn about the merits and demerits of various algorithms and to verify the outcome ourselves through implementing those same algorithms in python and running with massive amounts of data. The data we are using to test the algorithms comes from the web scraping. The data will then be subjected to algorithms that will process the data and show us the time taken, which we are using as the standard for program’s efficiency.

## **Scope**

A new and emerging game development company that wants to create a brandly new game will be required to have the data about all the games that have already been released and the users that want to experience new genres of games with specific requirements (i.e. size, category etc.) will find this program helpful.

## **Motivation**

As the mid term project we are required to implement all our learning and finding throughout the course of DSA. In this project we are free to choose any entity for scrapping. The reason we choose to work on games is because the google search showed that the play store contains more than 2 million data entities. We chose it so that we can do actual work on the latest knowledge we acquired in this semester through this data acquired. We sincerely hope that by pushing ourselves in this project we will learn a lot. Working hard to attain high marks and facing the pressure of deadlines, we hope to polish our skills and level up our knowledge.

## **Expectations**

Our end goal is to learn a lot about algorithm efficiency and to know about which algorithms are more efficient than others and vice versa. We sincerely hope that by doing this project we will be able to write efficient programs that will have less cost and better performance.

# **Algorithms:**

We used different algorithms for sorting and provided the control to the user that which algorithms user wants to use for sorting or searching purpose from following algorithms

## **Insertion Sort:**

### **Description**

In this insertion sort, we always assume that the first element of the array is sorted even if the whole array is unsorted. We compare each element of the array with the previous one to generate a sorted array.

For every single iteration insertion sort algorithm removes one element from array and swaps/inserts a sorted element in the appropriate index in array and iterations goes on until or unless the whole array is sorted.

### 

### **Pseudo Code**

| INSERTION-SORT(A)  for i = 1 to n  key ← A [i]  j ← i – 1  while j > = 0 and A[j] > key  A[j+1] ← A[j]  j ← j – 1  End while  A[j+1] ← key  End for |
| --- |

### 

### **Code in Python**

| def InsertionSort(*Arr*):  for i in range(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  return Arr |
| --- |

### **Strengths**

* Simplicity
* Space requirement is minimal
* Better than merge sort for smaller number of inputs

### **Weaknesses**

* Not good for large unsorted arrays
* Takes O(n^2) times to sort the array with n inputs
* Not reliable as other sorting algorithms

## **Quick Sort:**

### 

### **Description**

One of the most efficient sorting algorithms is the Quick sort algorithm. It sorts arrays much faster and requires no extra space for the sorting process that it follows unlike other algorithms.in this algorithm we use the approach known as divide and conquer.

In quick sort we select one random element as pivot and then we start sorting by comparing and moving elements smaller than pivot on the left of pivot and elements greater than pivot to the right of pivot by swapping elements. And this process is done on smaller subarrays rather than one big array entered by the user that's why it is faster.

However in this sorting approach relative order of similar items is not preserved which makes it an unstable sorting algorithm.

### 

### **Pseudocode**

| 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)  } |
| --- |

### **Code in python**

| def CountingSort(*Arr*):  k = int(max(Arr)) - int(min(Arr)) + 1  count = [0 for i in range(k)]  output = [0 for i in range(len(Arr))]  for i in range(0, len(Arr)):  count[Arr[i]-int(min(Arr))] += 1  for i in range(1, len(count)):  count[i] += count[i-1]  for i in range(len(Arr)-1,-1,-1):  output[count[Arr[i]-int(min(Arr))]-1] = Arr[i]  count[Arr[i]-int(min(Arr))] -= 1  return output |
| --- |

### **Strengths**

* Time complexity is O(n+k)
* Can be used as a reliable algorithm for mathematical problems.
* Stable algorithm

### **Weaknesses**

* It is recursive
* Worst case requires quadratic time
* Hard to implement requires complex logics.

### 

### **Time Complexity**

O(n+k)

## **Selection Sort:**

### 

### **Description**

Selection sort is somewhat like selecting the smallest element from the remaining elements of the array and swapping it with the element at its correct index. It is good for sorting small amounts of data like insertion sort.

This sorting algorithm is simpler than insertion sort as we simply find minimum and then move it to its correct location/index position but in some cases it is not as good as insertion sort. However extra spaces are not needed but only to store temporary variables.

### 

### **Pseudocode**

| 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 |
| --- |

### 

### **Code in python**

| def SelectionSort(*Arr*):  for i in range(len(Arr)-1):  minimum = i  for j in range(i+1, len(Arr)):  if Arr[j] < Arr[minimum]:  minimum = j  if minimum != i:  Arr[minimum], Arr[i] = Arr[i], Arr[minimum]  return Arr |
| --- |

### Strengths

* Performs well for small number of inputs
* No temporary space required during execution
* Initial ordering of item effect on its performance

### Weaknesses

* Performance is not good for large number of input
* Takes O(n2) time for n elements
* Other algorithms are more efficient like quick sort.

### Time Complexity

O(n2)

## **Merge Sort:**

### 

### **Description:**

Merge sort is one of the most commonly used algorithms in the field of computer science. It uses a divide and conquer approach to sort the given amount of inputs. In this sorting algorithm we divide our main array to sub arrays by calling functions recursively until base case approaches.

By dividing arrays to sub arrays the sorting task becomes less complex and less difficult.and after sorting each subarray the algorithm combines each subarray in sorted manner until a sorted array is given as output with actual entries of main array data. Merge sort is a stable sort i.e. the in order position of same elements isn't disturbed. It is also efficient in worst case scenario as its time complexity for worst case is O(n log n)

### 

### **Pseudocode**

| 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) |
| --- |

### **Code in python**

| def Merge(*Arr*, *l*, *r*, *m*):  n1 = l  n2 = m+1  A = []    while n1 <=m and n2 <= r:  if Arr[n1] < Arr[n2]:  A.append(Arr[n1])  n1 += 1  else:  A.append(Arr[n2])  n2 += 1  while n1 <= m:  A.append(Arr[n1])  n1 += 1  while n2 <= r:  A.append(Arr[n2])  n2 += 1  for i in range(l, r+1):  Arr[i] = A[i-l]    def MergeSort(*Arr*, *l*, *r*):  if r > l:  m = (l+r)//2  MergeSort(Arr, l, m)  MergeSort(Arr, m+1, r)  Merge(Arr, l, r, m) |
| --- |

### **Strengths**

* Performance does not depend on size of input.
* Worst case time is O(n log n)
* Stable sort

### 

### **Weaknesses**

* Less efficient
* Requires temporary space for execution
* More space than other algorithms

### 

### **Time Complexity**

O(n log n)

## **Bubble Sort:**

### **Description**

It swaps the adjacent pair of items if required throughout the whole list until sorting is done.

### **Pseudocode**

| 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 |
| --- |

### **Code in python**

| def BubbleSort(*Arr*):  for i in range(len(Arr)-1):  for j in range(len(Arr)-1):  if Arr[j] > Arr[j+1]:  Arr[j],Arr[j+1] = Arr[j+1], Arr[j]  return Arr |
| --- |

### **Strengths**

* Easy to implement
* Elements swapped i.e. don’t use extra space
* Space requirements is minimum

### 

### **Weaknesses**

* Doesn’t deal with lists with huge number of elements
* Requires O(n^2) time
* It is not for real life use i.e. only use for teaching purposes (Now-Days)

### 

### **Time Complexity**

O(n log n)

## **Counting Sort:**

### **Description**

It works by counting the number of objects having unique key values and then perform some arithmetic algebraic operation to sort the array.

### **Pseudocode**

| 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 |
| --- |

### **Code in python**

| def CountingSort(*Arr*):  k = int(max(Arr)) - int(min(Arr)) + 1  count = [0 for i in range(k)]  output = [0 for i in range(len(Arr))]    for i in range(0, len(Arr)):  count[Arr[i]-int(min(Arr))] += 1  for i in range(1, len(count)):  count[i] += count[i-1]  for i in range(len(Arr)-1,-1,-1):  output[count[Arr[i]-int(min(Arr))]-1] = Arr[i]  count[Arr[i]-int(min(Arr))] -= 1    return output |
| --- |

### **Strengths**

* Time complexity is O(n+k)
* Can be used as a reliable algorithm for mathematical problems.
* Stable algorithm

### 

### **Weaknesses**

* It is recursive
* Worst case requires quadratic time
* Hard to implement requires complex logics.

### 

### **Time Complexity**

O(n+k)

## **Radix Sort:**

### **Description**

It sorts integers based on their individual values i.e. it is performed from least significant to most significant digit.

### **Pseudocode**

| Function RadixSort(list)  Minimum maximum element of list  Exponential 1  While maximum/exponential is greater than 0  CountingSort(list, exponential)  Multiply the exponential by 10  Return list |
| --- |

### **Code in python**

| def CountingSortWithExp(*Arr*, *exponential*):  count = [0 for i in range(10)]  output = [0 for i in range(len(Arr))]    for i in range(0, len(Arr)):  rad = Arr[i] // exponential  count[rad % 10] += 1    for i in range(1, 10):  count[i] += count[i-1]    i = len(Arr)-1  while i>= 0:  rad = Arr[i] // exponential  output[count[rad%10]-1] = Arr[i]  count[rad % 10] -= 1  i -= 1    for n in range(len(Arr)):  Arr[n] = output[n]    def RadixSort(*Arr*):  maximum = max(Arr)  exponential = 1  while maximum/exponential>0:  CountingSortWithExp(Arr, exponential)  exponential \*= 10 |
| --- |

### **Strengths**

* Fast when arrays are short
* Stable algorithm

### 

### **Weaknesses**

* Less flexible
* Constant for time complexity is greater
* Requires more space than other sorts i.e. quick sort

### 

### **Time Complexity**

O(nk)

## **Bucket Sort:**

### **Description**

In this sorting algorithm data is divided into buckets and then each bucket is sorted independently as fuction call itself recursively.

### **Pseudocode**

| 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. |
| --- |

### **Code in python**

| def BucketSort(*Arr*, *n*):  A = [[] for i in range(n)]    for j in Arr:  b = int(n \* j)  A[b].append(j)    for i in range(n):  A[i] = InsertionSort(A[i])    m = 0  for n in range(n):  for o in range(len(A[n])):  Arr[m] = A[n][o]  m += 1  return Arr |
| --- |

### **Strengths**

* It distributes data into smaller buckets i.e. sort smaller arrays than original array.
* Use it as external sorting algorithms as data divided into buckets so it can sort directly data from RAM
* Works well if large degree of parallelism available

### 

### **Weaknesses**

* Can’t apply it to all data types
* Sensitive to distribution of input values
* To get better performances we need other algorithms

### 

### **Time Complexity**

Time complexity of Bucket sort is O(n2)

# **Interfaces for project:**

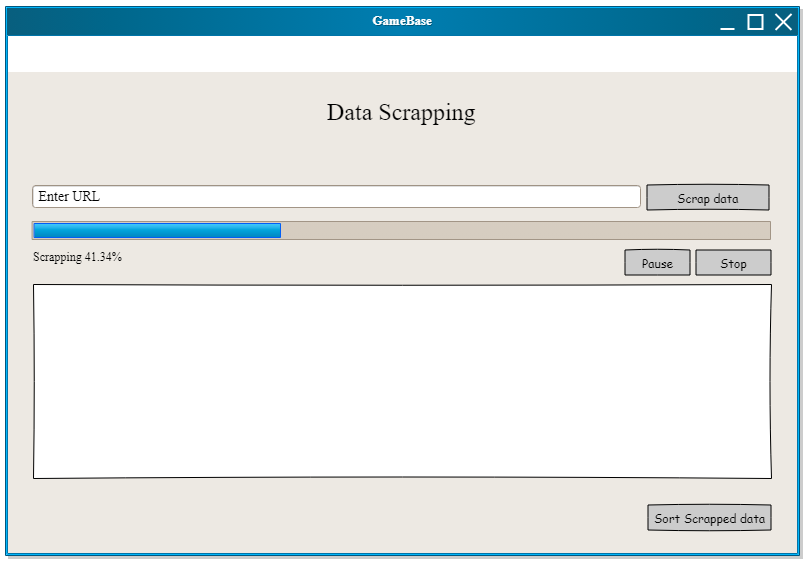
## 

## **Main Page**

**

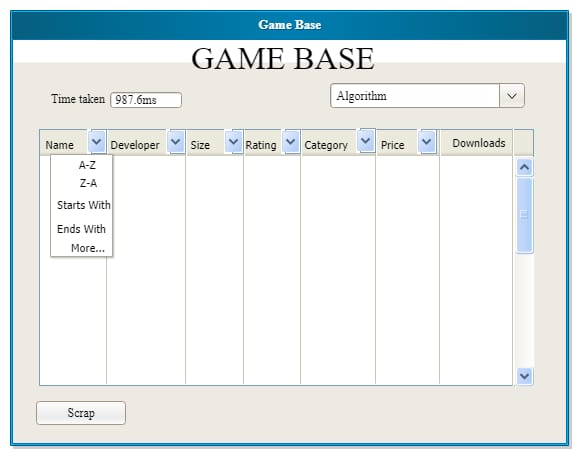
| UI Component Name | Type of UI component | Purpose of UI Component/Other details |
| --- | --- | --- |
| Terms | Link | To show the term and conditions window |
| Twitter/Facebook/telegram | Links | To open respective social media platform |
| Agree\_box | Check box | To confirm that user knows the terms and conditions of our system and agrees to them before using it |
| About panel | Panel | To graphically visualise the the description and importance of our system |
| Stary\_btn | Button | To get to the scraping screen |

## **Scrapping Page**



| UI Component Name | Type of UI component | Purpose of UI Component/Other details |
| --- | --- | --- |
| Progress | Progress bar | To visualize the scraping of Data through percentage |
| Text\_box | Rich text box | To visualize the data which is being scraped from website |
| Pause\_btn | Button | To pause scraping |
| Stop\_btn | Button | To stop scraping |
| Start\_btn | Button | To start scrapping from the website |
| Sort\_btn | Button | To go to the sorting screen |

## **Sorting Page**

**

| UI Component Name | Type of UI component | Purpose of UI Component/Other details |
| --- | --- | --- |
| Data table | Table | To present data in a sophisticated manner |
| Time box | Text box/label | To show the time taken by selected sorting algorithm |
| Algorithm selector | Combo box | To select an algorithm from given algorithm for sorting |
| Name filter | Combo box | To select and apply filter on name column |
| Developer filter | Combo box | To select and apply filter on developer column |
| Size filter | Combo box | To select and apply filter on size column |
| Rating filter | Combo box | To select and apply filter on rating column |
| Category filter | Combo box | To select and apply filter on category column |
| Price filter | Combo box | To select and apply filter on price column |
| Scrap\_btn | Button | To get back to scraping window |