ALGORITHMS

Name : Kartikey Sharma

Course: B. Sc. (H) Computer Science, II Year, IV Semester

College Roll No : CSC/21/18

University Roll No.21059570020

//1. i. Implement Insertion Sort (The program should report the number of comparisons)

//Test run the algorithm on 100 different inputs of sizes varying from
//30 to 1000. Count the number of comparisons and draw the graph.

```
#include<iostream>
#include<bits/stdc++.h>
using namespace std;
int y = 0;
void insertionsort(int arr[],int n){
  for(int i = 1;i<n;i++){
    int curr = arr[i];
    int prev = i-1;
    while(prev >= 0 && arr[prev]>curr){
      arr[prev+1] = arr[prev];
       prev--;
       y++;
    }
    arr[prev+1] = curr;
  }
}
void display(int array[], int size) {
  for(int i = 0; i<size; i++)
```

```
      cout << array[i] << " ";
   cout << endl;
}
void fun(int arr[],int s){
    int size = s;
    int u = 0;
    int b = 1000;
    arr[s];
 mt19937 num(random_device{}());
  uniform_int_distribution<int> dist(u,b);
for(int i= 0;i<s;i++){
    arr[i] = dist(num);
}
insertionsort(arr, size-1);     //(n-1) for last index
   cout << "Array after Sorting: ";
   display(arr, size);
}
int main(){
     int ub = 30;
   int lb = 1000;
   int size = 100;
  int u = 0;
  int arr[size] ;
  mt19937
```

```
 num(random_device{}());
 uniform_int_distribution<int> dist(ub, lb);
 for (auto& i : arr) {
  i = dist(num);
 }
  for (auto i : arr) {
   cout << i << " ";
 }
 cout << endl;
 int c = 0;
  cout<<"No. of comparisons = "<<y<<endl;
 ofstream myfile;
   myfile.open ("kartik.csv");
    myfile <<"Iteration No.,Array Size,Time ,Comparisons " <<" \n";
for(int j=0;j<100;j++){
   int s = arr[j];
   int ar[s];
   cout<<endl;
   cout<<"size of array  = "<<s<<endl;
   c++;
    clock_t time_req;
    time_req = clock();
  fun(ar,s);
 time_req = clock()- time_req;
```

```
    cout << "Processor time taken for iteration "<<j+1<<" : "
         << (float)time_req/CLOCKS_PER_SEC << " seconds" << endl;
  u = y - u;
   cout<<"No. of comparisons = "<<u<<endl;
    myfile <<j+1<<","<<s<<","<<(float)time_req/CLOCKS_PER_SEC<<","<<u<<" \n";
  u = y;}
myfile.close();
cout<<endl;
cout<<"number of inputs sorted = "<<c;
}
```

kartik.csv

Iteration No.	Array Size	Time	Comparisons
1	575	0.14	84053
2	735	0.15	141331
3	368	0.08	34059
4	918	0.19	217476
5	35	0.01	326
6	607	0.13	92527
7	876	0.19	194681
8	530	0.12	71871
9	961	0.2	235020
10	976	0.2	241619
11	194	0.05	9419
12	672	0.14	117017
13	337	0.07	29288
14	543	0.11	75017
15	710	0.14	133121
16	795	0.15	163947
17	277	0.06	19711
18	101	0.02	2453
19	587	0.12	86694
20	350	0.07	31100
21	358	0.08	32749
22	596	0.13	89141
23	261	0.05	17353
24	60	0.01	844
25	197	0.04	9486
26	931	0.19	220456

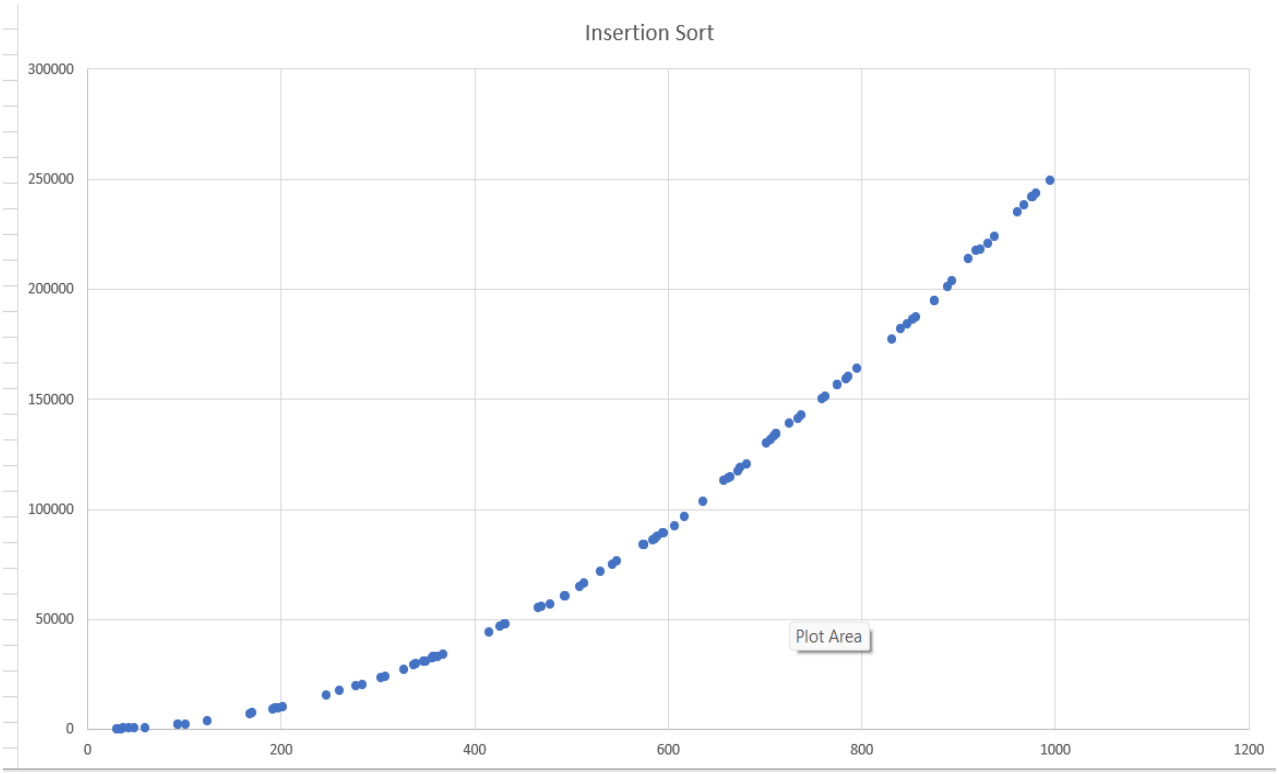
kartik.csv

Iteration No.	Array Size	Time	Comparisons
27	968	0.28	238246
28	49	0.01	579
29	664	0.16	114541
30	786	0.17	160191
31	493	0.11	60718
32	356	0.07	32225
33	94	0.02	2082
34	362	0.07	33136
35	43	0.01	454
36	432	0.09	47878
37	847	0.3	184169
38	911	0.19	213855
39	304	0.06	23375
40	662	0.14	113979
41	923	0.19	217862
42	584	0.12	85899
43	853	0.17	185951
44	37	0.01	378
45	574	0.13	83722
46	308	0.06	23800
47	893	0.18	203567
48	738	0.15	142613
49	31	0.01	238
50	712	0.14	134349
51	284	0.06	20356
52	469	0.1	55704

kartik.csv

Iteration No.	Array Size	Time	Comparisons
53	358	0.07	32749
54	247	0.05	15523
55	937	0.19	223776
56	876	0.18	194681
57	494	0.1	60765
58	726	0.3	139269
59	980	0.19	243408
60	889	0.18	201179
61	775	0.15	156627
62	356	0.07	32225
63	712	0.14	134349
64	681	0.14	120530
65	192	0.04	9242
66	617	0.12	96540
67	513	0.1	66564
68	94	0.02	2082
69	658	0.13	112892
70	775	0.15	156627
71	675	0.14	118662
72	832	0.18	177262
73	856	0.17	187285
74	170	0.03	7338
75	547	0.11	76323
76	32	0.01	256
77	327	0.07	27340
78	509	0.1	64995

	79	348	0.19	30594
	80	124	0.03	3936
	81	431	0.08	47557
	82	702	0.14	129953
	83	575	0.11	84053
	84	636	0.14	103509
	85	706	0.15	131770
	86	763	0.16	151363
	87	589	0.12	87376
	88	478	0.1	56845
	89	340	0.06	29596
	90	784	0.15	159091
	91	995	0.2	249351
	92	759	0.15	150298
	93	595	0.12	88996
	94	977	0.2	241657
	95	415	0.09	44022
	96	202	0.04	9933
	97	427	0.09	46787
	98	466	0.1	54983
	99	168	0.04	7196
	100	841	0.29	181827



// ii. Implement Merge Sort (The program should report the number of comparisons)

```
#include<iostream>
#include<bits/stdc++.h>
#include <random>
using namespace std;
int y = 0;
void display(int arr[], int s) {
  for(int i = 0; i<s; i++)
    cout << arr[i] << " ";
  cout << endl;
}
```

```
void merge(int arr[], int si, int mid, int ei) {
  int i, j, k, nl, nr;
  nl = mid-si+1; nr = ei-mid;
  int larr[nl], rarr[nr];
  for(i = 0; i<nl; i++)
    larr[i] = arr[si+i];
  for(j = 0; j<nr; j++)
    rarr[j] = arr[mid+1+j];
  i = 0; j = 0; k = si;
  while(i < nl && j<nr) {
    if(larr[i] <= rarr[j]) {
      arr[k] = larr[i];
      i++;
    }else{
      arr[k] = rarr[j];
      j++;
    }
    k++;
   y++;
  }
  while(i<nl) {
    arr[k] = larr[i];
    i++; k++;
  }
```

```
  while(j<nr) {
    arr[k] = rarr[j];
    j++; k++;
  }
}
void mergeSort(int arr[], int si, int r) {
  int mid;
  if(si < r) {
    int mid = si+(r-si)/2;
    mergeSort(arr, si, mid);
    mergeSort(arr, mid+1, r);
    merge(arr, si, mid, r);
  }
}
void fun(int arr[],int s){
   int size = s;
   int u = 0;
   int b = 1000;
   arr[s];
 mt19937 pr(random_device{}());
  uniform_int_distribution<int> dist(u,b);
for(int i= 0;i<s;i++){
   arr[i] = dist(pr);
}
```

```
 mergeSort(arr, 0, size-1);
   cout << "Array after Sorting: ";
   display(arr, size);
}
int main() {
    int ub = 30;
   int lb = 1000;
   int size = 100;
  int u = 0;
  int arr[size] ;
  mt19937
   num(random_device{}());
  uniform_int_distribution<int> dist(ub, lb);
  for (auto& i: arr) {
    i = dist(num);
  }
   for (auto i: arr) {
    cout << i << " ";
  }
  cout << endl;
   int c = 0;
    ofstream myfile;
    myfile.open ("kartikey.csv");
     myfile <<"Iteration No.,Array Size,Time ,Comparisons " <<" \n";
```

```
    clock_t time_req;
for(int j=0;j<100;j++){
    int s = arr[j];
    int ar[s];
    cout<<endl;
    cout<<"size of array  = "<<s<<endl;
    c++;
     time_req = clock();
   fun(ar,s);
  time_req = clock()- time_req;
    cout << "Processor time taken for iteration "<<j+1<<" : "
         << (float)time_req/CLOCKS_PER_SEC << " seconds" << endl;
  u = y - u;
   cout<<"No. of comparisons = "<<u<<endl;
    myfile <<j+1<<","<<s<<","<<(float)time_req/CLOCKS_PER_SEC<<","<<u<<" \n";
  u = y;
}
myfile.close();
cout<<endl;
cout<<"number of inputs sorted = "<<c;
}
```

kartikey.csv

Iteration No.	Array Size	Time	Comparisons
1	575	0.07	4549
2	735	0.09	6070
3	368	0.05	2682
4	918	0.11	7868
5	35	0.01	139
6	607	0.07	4852
7	876	0.11	7482
8	530	0.07	4143
9	961	0.12	8307
10	976	0.12	8419
11	194	0.02	1214
12	672	0.07	5460
13	337	0.04	2397
14	543	0.06	4269
15	710	0.08	5823
16	795	0.12	6670
17	277	0.13	1894
18	101	0.01	543
19	587	0.08	4661
20	350	0.05	2513
21	358	0.05	2586
22	596	0.08	4731
23	261	0.03	1756
24	60	0.01	286
25	197	0.02	1242
26	931	0.11	8012

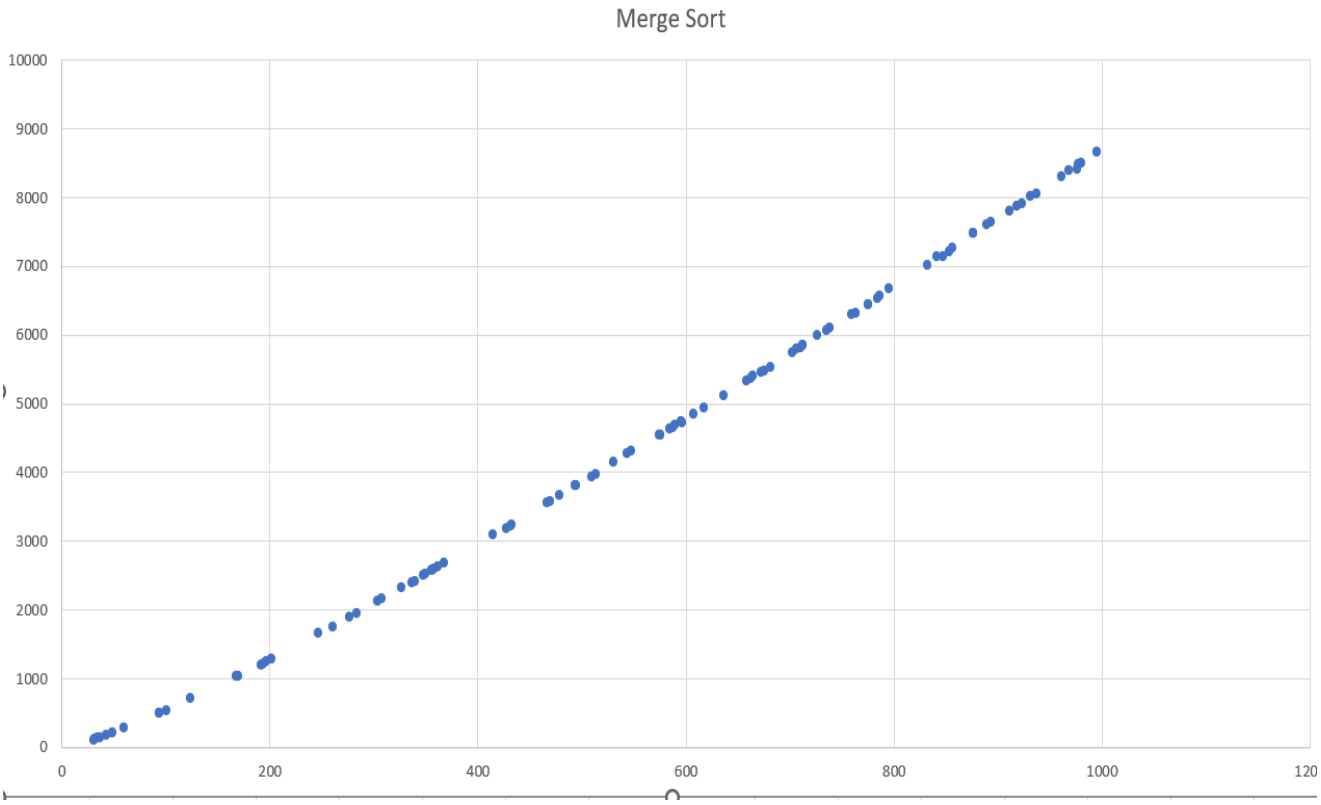
kartikey.csv

Iteration No.	Array Size	Time	Comparisons
27	968	0.11	8394
28	49	0.01	213
29	664	0.08	5401
30	786	0.09	6568
31	493	0.06	3812
32	356	0.04	2574
33	94	0.01	490
34	362	0.04	2626
35	43	0.01	178
36	432	0.05	3238
37	847	0.09	7146
38	911	0.1	7805
39	304	0.04	2126
40	662	0.08	5366
41	923	0.1	7915
42	584	0.07	4639
43	853	0.1	7221
44	37	0	144
45	574	0.13	4553
46	308	0.04	2157
47	893	0.1	7635
48	738	0.08	6094
49	31	0	113
50	712	0.08	5852
51	284	0.03	1956
52	469	0.05	3585

kartikey.csv

Iteration No.	Array Size	Time	Comparisons
53	358	0.04	2586
54	247	0.03	1660
55	937	0.11	8047
56	876	0.1	7482
57	494	0.06	3818
58	726	0.08	5993
59	980	0.11	8496
60	889	0.11	7607
61	775	0.09	6440
62	356	0.06	2574
63	712	0.11	5852
64	681	0.1	5536
65	192	0.03	1202
66	617	0.14	4945
67	513	0.07	3974
68	94	0.01	490
69	658	0.08	5336
70	775	0.09	6440
71	675	0.08	5473
72	832	0.1	7012
73	856	0.1	7259
74	170	0.02	1031
75	547	0.06	4314
76	32	0	118
77	327	0.04	2324
78	509	0.06	3941

	79	348	0.04	2504
	80	124	0.02	713
	81	431	0.05	3227
	82	702	0.08	5740
	83	575	0.07	4549
	84	636	0.08	5114
	85	706	0.08	5796
	86	763	0.09	6325
	87	589	0.07	4686
	88	478	0.06	3672
	89	340	0.04	2414
	90	784	0.15	6533
	91	995	0.12	8654
	92	759	0.08	6296
	93	595	0.07	4734
	94	977	0.11	8477
	95	415	0.05	3094
	96	202	0.02	1289
	97	427	0.05	3192
	98	466	0.05	3558
	99	168	0.02	1031
	100	841	0.1	7137



//same for heap sort

```
#include<iostream>
#include<bits/stdc++.h>
#include <random>
using namespace std;
int y = 0;
void heapify(int arr[],int i,int size){
 int left = 2*i + 1;
 int right = 2*i + 2;
 int maxidx = i;
 if(left<size && arr[left]>arr[maxidx]){
```

```
    maxidx = left;
    y++;
  }
  if(right<size && arr[right]>arr[maxidx]){
    maxidx = right;
    y++;
  }
  if( maxidx != i){
    int temp = arr[i];
    arr[i] = arr[maxidx];
    arr[maxidx] = temp;

heapify(arr,maxidx,size);
  }
}
void display(int arr[], int s) {
   for(int i = 0; i<s; i++)
      cout << arr[i] << " ";
   cout << endl;
}
void heapsort(int arr[],int n){
   // Build heap (rearrange array)
    for(int i=n/2;i>=0;i--){
        heapify(arr,i,n);
```

```
    }
     // One by one extract an element from heap and Moving current root to end
    for(int i=n-1;i>=0;i--){
      int temp = arr[0];
      arr[0] = arr[i];
      arr[i] = temp;
     // heapify on the reduced heap
        heapify(arr,0,i);
    }
}
void fun(int arr[],int s){
    int size = s;
    int u = 0;
    int b = 1000;
    arr[s];
    //generating random numbers and string them in the arr[s]
    // where s is the particular index of another 100 size array
 mt19937 pr(random_device{}());
  uniform_int_distribution<int> dist(u,b);
for(int i= 0;i<s;i++){
    arr[i] = dist(pr);
}
 heapsort(arr, size-1);
   cout << "Array after Sorting: ";
```

```
  display(arr, size);
}
int main(){
  int ub = 30;
   int lb = 1000;
   int size = 100;
  int u = 0;
  int arr[size] ;
//generating 100 random numbers and storing them in an array
  mt19937 num(random_device{}());
  uniform_int_distribution<int> dist(ub, lb);
  for (auto& i: arr) {
    i = dist(num);
  }
   for (auto i: arr) {
    cout << i << " ";
  }
  cout << endl;
   int c = 0;
    ofstream myfile;
    myfile.open ("kartikeyh.csv");
     myfile <<"Iteration No.,Time,Array Size ,Comparisons " <<" \n";
    clock_t time_req;
for(int j=0;j<100;j++){
```

```
    int s = arr[j];
    int ar[s];
    cout<<endl;
    cout<<"size of array  = "<<s<<endl;
    c++;
     time_req = clock();
   fun(ar,s);
  time_req = clock()- time_req;
    cout << "Processor time taken for iteration "<<j+1<<" : "
         << (float)time_req/CLOCKS_PER_SEC << " seconds" << endl;
  u = y - u;
   cout<<"No. of comparisons = "<<u<<endl;
    myfile <<j+1<<","<<(float)time_req/CLOCKS_PER_SEC<<","<<s<<","<<u<<" \n";
  u = y;
}
myfile.close();
cout<<endl;
cout<<"number of inputs sorted = "<<c;
}
```

kartikeyh.csv

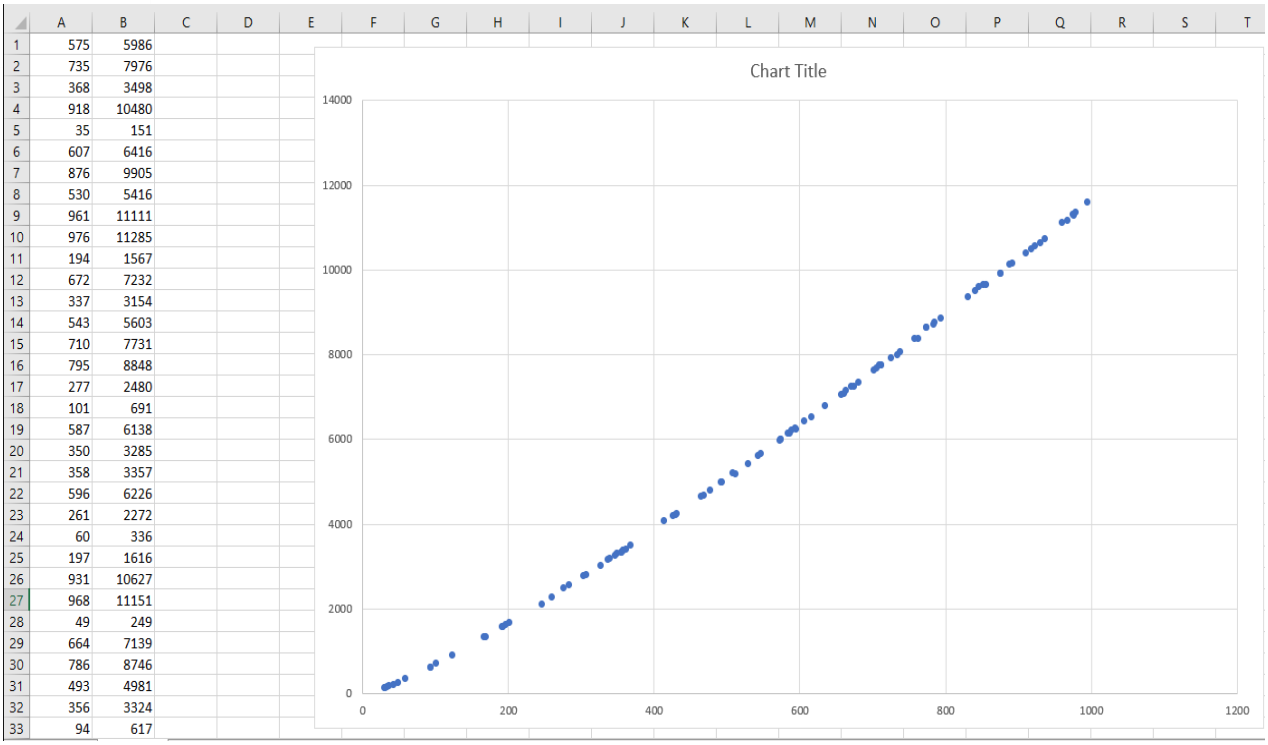
Iteration No.	Time	Array Size	Comparisons
1	0.15	575	5986
2	0.26	735	7976
3	0.25	368	3498
4	0.28	918	10480
5	0.02	35	151
6	0.14	607	6416
7	0.15	876	9905
8	0.1	530	5416
9	0.18	961	11111
10	0.18	976	11285
11	0.04	194	1567
12	0.13	672	7232
13	0.06	337	3154
14	0.1	543	5603
15	0.14	710	7731
16	0.14	795	8848
17	0.06	277	2480
18	0.02	101	691
19	0.1	587	6138
20	0.07	350	3285
21	0.07	358	3357
22	0.1	596	6226
23	0.15	261	2272
24	0.01	60	336
25	0.04	197	1616
26	0.17	931	10627
27	0.17	968	11151
28	0.02	49	249
29	0.12	664	7139
30	0.14	786	8746
31	0.1	493	4981
32	0.06	356	3324
33	0.02	94	617

kartikeyh.csv

Iteration No.	Time	Array Size	Comparisons
34	0.06	362	3396
35	0.01	43	210
36	0.07	432	4227
37	0.16	847	9586
38	0.17	911	10393
39	0.06	304	2758
40	0.12	662	7064
41	0.18	923	10556
42	0.1	584	6133
43	0.17	853	9641
44	0.01	37	175
45	0.19	574	5957
46	0.06	308	2801
47	0.15	893	10141
48	0.14	738	8044
49	0.01	31	119
50	0.13	712	7732
51	0.05	284	2542
52	0.09	469	4660
53	0.06	358	3357
54	0.05	247	2101
55	0.18	937	10724
56	0.16	876	9905
57	0.09	494	4984
58	0.13	726	7906
59	0.18	980	11338
60	0.16	889	10108
61	0.14	775	8618
62	0.06	356	3324
63	0.13	712	7732
64	0.12	681	7342
65	0.04	192	1558
66	0.2	617	6508
67	0.09	513	5179

kartikeyh.csv

Iteration No.	Time	Array Size	Comparisons
68	0.02	94	617
69	0.12	658	7036
70	0.14	775	8618
71	0.12	675	7246
72	0.16	832	9348
73	0.15	856	9639
74	0.02	170	1323
75	0.1	547	5653
76	0.01	32	132
77	0.06	327	3020
78	0.1	509	5187
79	0.06	348	3244
80	0.02	124	884
81	0.08	431	4215
82	0.13	702	7610
83	0.1	575	5986
84	0.12	636	6782
85	0.14	706	7672
86	0.14	763	8372
87	0.1	589	6209
88	0.09	478	4790
89	0.16	340	3166
90	0.14	784	8691
91	0.18	995	11596
92	0.14	759	8369
93	0.11	595	6263
94	0.17	977	11276
95	0.07	415	4061
96	0.03	202	1660
97	0.07	427	4176
98	0.09	466	4647
99	0.03	168	1317
100	0.14	841	9488



	A	B
1	575	5986
2	735	7976
3	368	3498
4	918	10480
5	35	151
6	607	6416
7	876	9905
8	530	5416
9	961	11111
10	976	11285
11	194	1567
12	672	7232
13	337	3154
14	543	5603
15	710	7731
16	795	8848
17	277	2480
18	101	691
19	587	6138
20	350	3285
21	358	3357
22	596	6226
23	261	2272
24	60	336
25	197	1616
26	931	10627
27	968	11151
28	49	249
29	664	7139
30	786	8746
31	493	4981
32	356	3324
33	94	617

Q3)//implementing randomized quicksort :-

```
#include<iostream>
#include<bits/stdc++.h>
using namespace std;
int y=0;
int partition(int arr[],int left,int right){
    int p = arr[right];
    int i = left-1; //holding elements smaller then pivot
    for(int j=left;j<right;j++){
      if( arr[j]<=p){
       i++;
       y++;
```

```
        int temp = arr[j];
        arr[j] = arr[i];
        arr[i] = temp;
       }
     }
      i++;
        int temp = p;
        arr[right] = arr[i];
        arr[i] = temp;
        return i;
        }
int partition_r(int arr[], int left, int right)
{
   // Generate a random number in between
   // low .. high
   srand(time(NULL));
   int random = left + rand() % (right - left);

   // Swap A[random] with A[high]
   swap(arr[random], arr[right]);

   return partition(arr, left, right);
}
```

```
void display(int arr[], int s) {
  for(int i = 0; i<s; i++)
     cout << arr[i] << " ";
  cout << endl;
}
void quicksort(int arr[],int left,int right){
   if(left>=right){
       return;
   }
  int pivot_index =  partition_r(arr,left,right);
  //cout<<pivot_index<<endl;
   quicksort(arr,left,pivot_index-1);
   quicksort(arr,pivot_index+1,right);
  }
  void fun(int arr[],int s){
   int size = s;
   int u = 0;
   int b = 1000;
   arr[s];
   //generating random numbers and string them in the arr[s]
   // where s is the particular index of another 100 size array
mt19937 pr(random_device{}());
 uniform_int_distribution<int> dist(u,b);
for(int i= 0;i<s;i++){
```

```
  arr[i] = dist(pr);
}
quicksort(arr,0,s-1);
  cout << "Array after Sorting: ";
  display(arr, size);
}
int main(){
  int ub = 30;
  int lb = 1000;
  int size = 100;
 int u = 0;
 int arr[size] ;
//generating 100 random numbers and storing them in an array
 mt19937 num(random_device{}());
 uniform_int_distribution<int> dist(ub, lb);
 for (auto& i: arr) {
   i = dist(num); }
  for (auto i: arr) {
   cout << i << " "; }
 cout << endl;
  int c = 0;
   ofstream myfile;
   myfile.open ("quick.csv");
    myfile <<"Iteration No.,Time,Array Size ,Comparisons " <<" \n";
```

```
  clock_t time_req;
for(int j=0;j<100;j++){
  int s = arr[j];
  int ar[s];
  cout<<endl;
  cout<<"size of array  = "<<s<<endl;
  c++;
   time_req = clock();
 fun(ar,s);
time_req = clock()- time_req;
  cout << "Processor time taken for iteration "<<j+1<<" : "
      << (float)time_req/CLOCKS_PER_SEC << " seconds" << endl;
u = y - u;
 cout<<"No. of comparisons = "<<u<<endl;
  myfile <<j+1<<","<<(float)time_req/CLOCKS_PER_SEC<<","<<s<<","<<u<<" \n";
u = y; }
myfile.close();
cout<<endl;
cout<<"number of inputs sorted = "<<c; }
```

quick.csv

Iteration No.	Time	Array Size	Comparisons
1	0.42	575	2785
2	0.3	735	4154
3	0.1	368	1818
4	0.19	918	5267
5	0.01	35	129
6	0.12	607	2802
7	0.16	876	4097
8	0.1	530	2209
9	0.19	961	6120
10	0.17	976	5312
11	0.04	194	736
12	0.12	672	3294
13	0.06	337	1849
14	0.1	543	2374
15	0.13	710	3998
16	0.14	795	4562
17	0.06	277	1289
18	0.02	101	338
19	0.11	587	2602
20	0.06	350	1794
21	0.06	358	1453
22	0.1	596	4115
23	0.05	261	1134
24	0.01	60	198
25	0.04	197	753
26	0.17	931	5863

⚠ 4

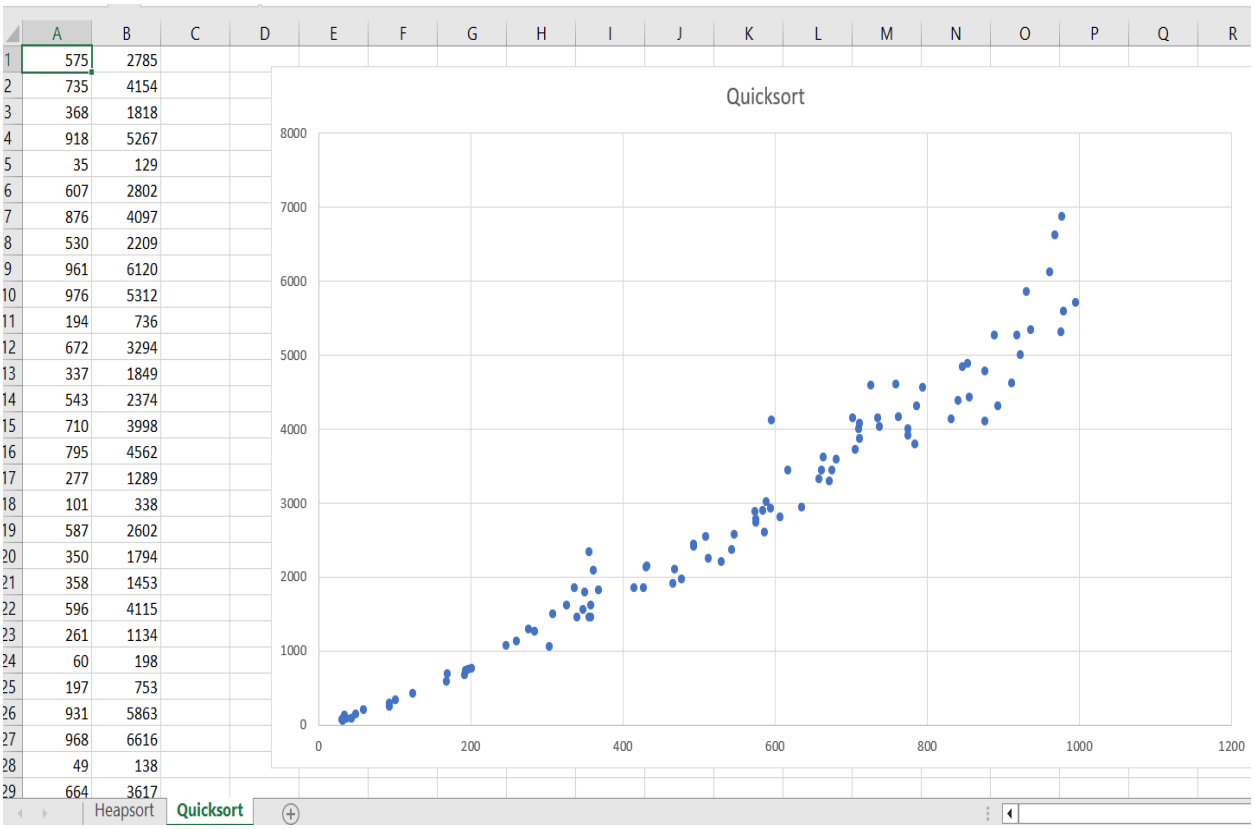
quick.csv

Iteration No.	Time	Array Size	Comparisons
53	0.06	358	1618
54	0.12	247	1077
55	0.16	937	5343
56	0.16	876	4787
57	0.09	494	2410
58	0.13	726	4589
59	0.18	980	5596
60	0.16	889	5271
61	0.14	775	3914
62	0.06	356	2344
63	0.13	712	3868
64	0.12	681	3586
65	0.03	192	668
66	0.11	617	3439
67	0.09	513	2248
68	0.02	94	298
69	0.12	658	3326
70	0.14	775	4006
71	0.11	675	3441
72	0.14	832	4132
73	0.24	856	4432
74	0.03	170	686
75	0.1	547	2568
76	0	32	62
77	0.06	327	1618
78	0.1	509	2549

mergesort.cpp | heapsort.cpp | heapoperations.cpp

quick.csv

Iteration No.	Time	Array Size	Comparisons
75	0.1	547	2568
76	0	32	62
77	0.06	327	1618
78	0.1	509	2549
79	0.06	348	1555
80	0.02	124	430
81	0.07	431	2135
82	0.12	702	4144
83	0.11	575	2741
84	0.11	636	2947
85	0.12	706	3728
86	0.14	763	4157
87	0.1	589	3018
88	0.09	478	1968
89	0.06	340	1451
90	0.15	784	3794
91	0.18	995	5705
92	0.14	759	4610
93	0.11	595	2929
94	0.18	977	6876
95	0.14	415	1853
96	0.04	202	761
97	0.07	427	1859
98	0.08	466	1906
99	0.03	168	586
100	0.15	841	4380

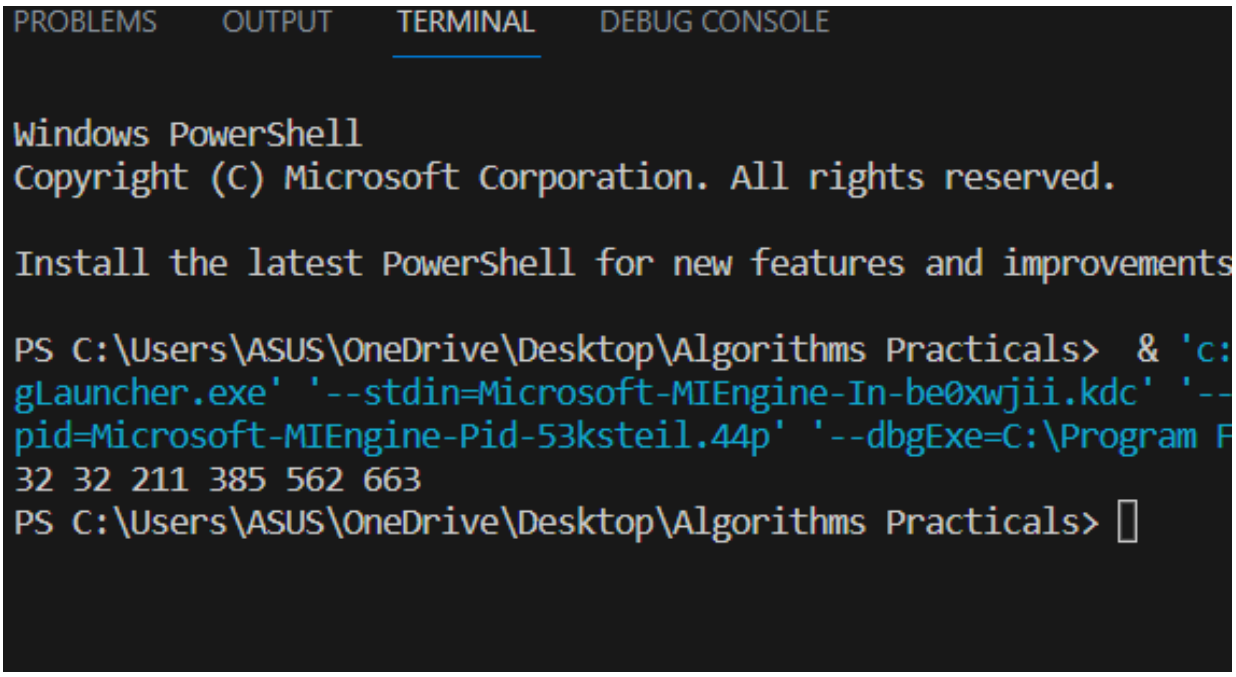


Q4) Implement Radix Sort :-

```
#include<iostream>

using namespace std;


int getmax(int arr[],int n){
   int m = arr[0];
 for(int i=1;i<n;i++){
   if(arr[i]>m){
       m = arr[i];
   }
 }
```

```
  return m;
}

void countsort(int arr[],int n,int p){
    int b[n];
   int c[10] = {0};
   for(int i=0;i<n;i++){
    ++c[(arr[i]/p)%10];
   }
   for(int i=1;i<=n;i++){
    c[i]=c[i]+c[i-1];
   }
   for(int i=n-1;i>=0;i--){
     b[--c[(arr[i]/p)%10]] = arr[i];
   }
   for(int i=0;i<n;i++){
    arr[i]=b[i];

   }

}

void radixsort(int arr[],int n){
    int m = getmax(arr,n);
```

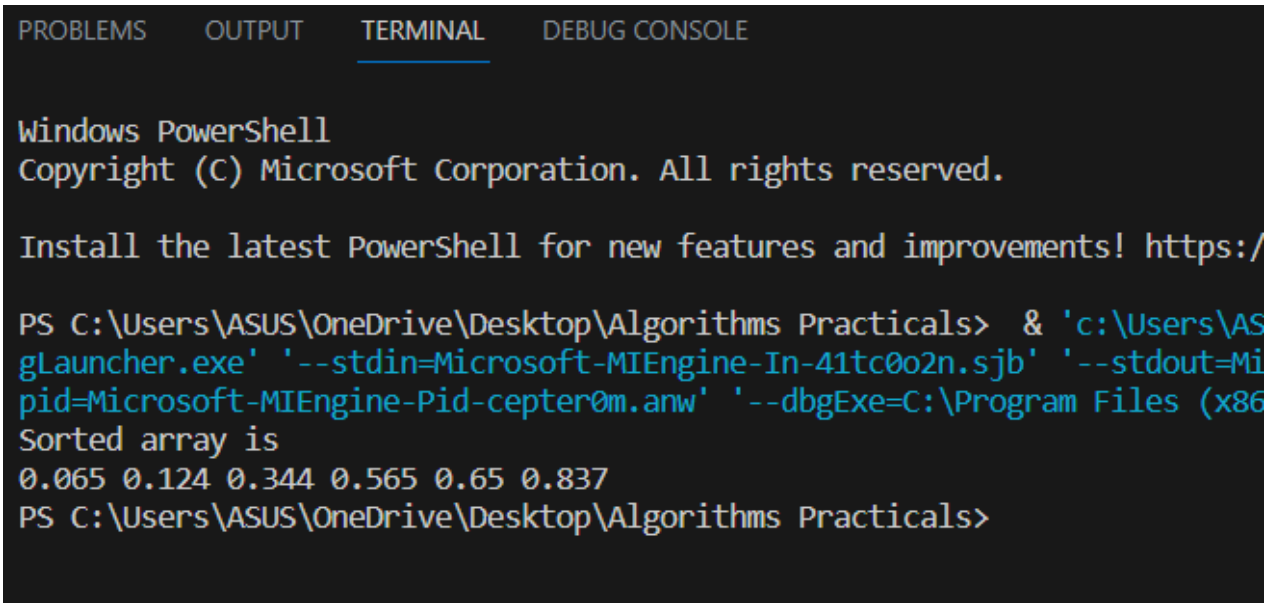
```
    for(int i=1;m/i>0;i*=10){
        countsort(arr,n,i);
    }
    for(int i=0;i<n;i++){
    cout<<arr[i]<<" ";
   }
}
int main(){
   int arr[] = {663,562,385,211,906,328};
   int n = 6;
   radixsort(arr,n);
}
```



Q5) Implement Bucket Sort :-

```
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;
void bucketSort(float arr[], int n)
{
   vector<float> b[n];
   for (int i = 0; i < n; i++) {
      int bi = n * arr[i];
      b[bi].push_back(arr[i]);
   }
   for (int i = 0; i < n; i++)
      sort(b[i].begin(), b[i].end());
   int index = 0;
   for (int i = 0; i < n; i++)
      for (int j = 0; j < b[i].size(); j++)
         arr[index++] = b[i][j];
}
int main()
{
   float arr[]
      = { 0.837, 0.565, 0.65, 0.124, 0.065, 0.344 };
   int n = sizeof(arr) / sizeof(arr[0]);
```

```
    bucketSort(arr, n);

    cout << "Sorted array is \n";
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    return 0;
}
```



Q6) Implement Randomized Select :-

```
#include<iostream>
#include<time.h>
#include<ctime>
using namespace std;
```

```
// Creating the Exchange function :
void exchange(int &a,int &b){
   int temp=a;
   a=b;
   b=temp;
}


// function to select a random number between two given numbers :
/*int Random(int p,int r){
   srand(time(NULL));
   int x= rand()%r;
   return x;
}*/
// Creating the Partition Function :
int Partiton(int A[], int p,int r){
int x=A[r];
int i=(p-1);
for(int j=p;j<=r-1;j++){
if(A[j] <=x){
   i=i+1;
   exchange(A[i],A[j]);
```

```
}
}
exchange(A[i+1],A[r]);
return (i+1);
}

// Creating the random partiton function :
int Randomized_Partiton(int A[],int p,int r){
   int i=r;
   exchange(A[r],A[i]);
   return Partiton(A,p,r);
}

// Randomized Select :
int Randomized_Select(int A[],int p,int r,int i){
   if(p==r)
      return A[p];
   int q=Randomized_Partiton(A,p,r);
   int k=q-p+1;
   if (i==k){
      return A[q];
   }
```

```
    else if (i<k){
        return Randomized_Select(A,p,q-1,i);
    }
    else{
        return Randomized_Select(A,q+1,r,i-k);
    }
}
//Driver Code :
int main(){
  int A[]={5,2,7,9,6,12,4,3,16,14};
  cout<<"\nThe array is :  ";
  for(int e:A){
   cout<<e<<"  ";
  }
  cout<<endl;
   cout<<"The 1 st order statistics is :
"<<Randomized_Select(A,0,9,1)<<endl;
   cout<<"The 3 th order statistics is :
"<<Randomized_Select(A,0,9,3)<<endl;
   cout<<"The 10 th order statistics is
:"<<Randomized_Select(A,0,9,10)<<endl<<endl;
   return 0;
}
```

```
The array is :  5  2  7  9  6  12  4  3  16  14
The 1 st order statistics is : 2
The 3 th order statistics is : 4
The 10 th order statistics is :16

PS C:\Users\ASUS\OneDrive\Desktop\Algorithms Practicals>
```

Q7) Implement Breadth-First Search in a graph :-

```
#include <bits/stdc++.h>
#include <iostream>
#include<algorithm>
#include <iomanip>
using namespace std;
const int s = 100;
void bfs(queue<int>b){
   while(!b.empty()){
      cout<<b.front()<<" ";
      b.pop();
   }
}
void visited(vector<int>& v,queue<int>& b,int admatrrix[][s],int size){
   int c=0;
   for(int i=1;i<=size;i++){
      for (int j=1;j<=size;j++)
```

```
        if(admatrrix[i][j]!=0 && c==0){

            v.push_back(i);
            b.push(i);
            v.push_back(j);
            b.push(j);
            c++;
        }
       else  if(admatrrix[i][j]!=0){
            int c=count(v.begin(),v.end(),j);
            if(c==0){
               v.push_back(j);
               b.push(j);
            }
        }
    }

}
int main()
{
    int size, size1, size2;

    cout << "Enter the number of verteces";
    cin >> size;
```

```
cout << endl;
int admattrix[s][s];
for (int i = 1; i <= size; i++)
{
   for (int j = 1; j <= size; j++)
   {
      admattrix[i][j] = 0;
   }
}
cout << "Enter the number of edges ";
cin >> size1;
int s1, s2, w;

int w1, u, v;
queue<int>b;
vector<int>v1;
cout << "Enter the vertex1 and vertex2";
for (int i = 1; i <= size1; i++)
{
   cin >> u >> v;

   admattrix[u][v] = 1;
   admattrix[v][u] = 1;
```

```
    }
    cout<<"\n adjacency matrix \n";
    cout << " ";
    for (int i = 1; i <= size; i++)
    {
        cout << setw(4) << " " << i;
    }
    cout << endl;
    for (int i = 1; i <= size; i++)
    {
        cout << i;
        for (int j = 1; j <= size; j++)
        {
            cout << setw(5) << admattrix[i][j];
        }
        cout << endl;
    }cout<<"\nbfs of graph :  ";
visited(v1,b,admattrix,size);

bfs(b);
    return 0;
}
```

```
Enter the number of verteces4

Enter the number of edges 4
Enter the vertex1 and vertex22 3

4

5
6
7
8
9

 adjacency matrix
     1    2    3    4
1    0    0    0    0
2    0    0    1    0
3    0    1    0    0
4    0    0    0    0

bfs of graph :  2 3
```

Q8) Implement Depth-First Search in a graph :-

```
#include <bits/stdc++.h>

#include <iostream>

#include<algorithm>

#include <iomanip>

using namespace std;

const int s = 100;

void dfs(queue<int>b){

   while(!b.empty()){

      cout<<b.front()<<" ";
```

```
        b.pop();
    }
}
void visited(vector<int>& v,queue<int>& b,stack<int>& s1,int admatrrix[][s],int
size){
    int c=0;
    for(int i=1;i<=size;i++){
        for (int j=1;j<=size;j++){
        if(admatrrix[i][j]!=0 && c==0){

            s1.push(i);
            s1.push(j);
            v.push_back(i);
            b.push(i);
            b.push(j);
            v.push_back(j);

            i=j;
            j=0;
            c++;
        }
       else  if(admatrrix[i][j]!=0){
            int c=count(v.begin(),v.end(),j);
            if(c==0){
```

```
            v.push_back(j);

            b.push(j);
            s1.push(j);
            i=j;
            j=0;
        }
    }
    if(j==size && !s1.empty()){

        int p=s1.top();
        s1.pop();
        i=p;
        j=0;
    }
  }
  }

}
int main()
{
  int size, size1, size2;

  cout << "Enter the number of verteces";
```

```
cin >> size;
cout << endl;
int admattrix[s][s];
for (int i = 1; i <= size; i++)
{
   for (int j = 1; j <= size; j++)
   {
      admattrix[i][j] = 0;
   }
}
cout << "Enter the number of edges ";
cin >> size1;
int w1, u, v;
queue<int>b;
vector<int>v1;
stack<int>s1;
cout << "Enter the vertex1 and vertex2";
for (int i = 1; i <= size1; i++)
{
   cin >> u >> v;

   admattrix[u][v] = 1;
   admattrix[v][u] = 1;
```

```
    }
    cout<<"\n adjacency matrix \n";
    cout << " ";
    for (int i = 1; i <= size; i++)
    {
       cout << setw(4) << " " << i;
    }
    cout << endl;
    for (int i = 1; i <= size; i++)
    {
       cout << i;
       for (int j = 1; j <= size; j++)
       {
          cout << setw(5) << admattrix[i][j];
       }
       cout << endl;
    }cout<<"\nDFS of graph :  ";
visited(v1,b,s1,admattrix,size);
     dfs(b);

    return 0;
}
```

```
Enter the number of verteces3

Enter the number of edges 4
Enter the vertex1 and vertex22
3
4
5
1
6
7
4

 adjacency matrix
     1    2    3
1    0    0    0
2    0    0    1
3    0    1    0
```

Q9) Write a program to determine the minimum spanning tree of a graph using both Prims and Kruskals algorithm :-

Prims :-

```
#include <iostream>

using namespace std;


class Edge

{

public:

  int src;
```

```
  int dest;
  int weight;
};

class Subset
{
public:
  int p;
  int rank;
};

int compEdges(const void *a, const void *b)
{
  return ((Edge *)(a))->weight > ((Edge *)(b))->weight;
}

class Graph
{
public:
  int V, E;
  Edge *edges;
  Subset *subsets;

  Graph(int V, int E)
```

```
{
  this->V = V;
  this->E = E;
  this->edges = new Edge[E];

  for (int i = 0; i < E; i++)
  {
    int src, dest, weight;
    cout << "Edge " << (i + 1)
        << "\n=============\n";
    cout << "Source Node: ";
    cin >> src;
    cout << "Destination Node: ";
    cin >> dest;
    cout << "Edge Weight: ";
    cin >> weight;
    cout << endl;

    if (src < 1 || src > V || dest < 1 || dest > V)
    {
      cout << "Invalid Node" << endl;
      exit(-1);
    }
```

```
    this->edges[i].src = src - 1;
    this->edges[i].dest = dest - 1;
    this->edges[i].weight = weight;
  }
}

void makeSet()
{
  this->subsets = new Subset[(this->V * sizeof(Subset))];

  for (int v = 0; v < this->V; ++v)
  {
    this->subsets[v].p = v;
    this->subsets[v].rank = 0;
  }
}

int findSet(int i)
{
  if (this->subsets[i].p != i)
  {
    this->subsets[i].p = this->findSet(this->subsets[i].p);
  }
```

```
  return this->subsets[i].p;
}

void link(int x, int y)
{
  if (this->subsets[x].rank > this->subsets[y].rank)
  {
    this->subsets[y].p = x;
  }
  else
  {
    this->subsets[x].p = y;

    if (this->subsets[x].rank == this->subsets[y].rank)
    {
      this->subsets[y].rank++;
    }
  }
}

void Union(int x, int y)
{
  this->link(this->findSet(x), this->findSet(y));
}
```

```
void KruskalMST()
{
  int e = 0, i = 0;
  Edge next, result[this->V];

  qsort(this->edges, this->E, sizeof(Edge), compEdges);

  this->makeSet();

  while (e < this->V - 1 && i < this->E)
  {
    next = this->edges[i++];

    int x = this->findSet(next.src);
    int y = this->findSet(next.dest);

    if (x != y)
    {
      result[e++] = next;
      this->Union(x, y);
    }
  }
  qsort(result, this->V - 1, sizeof(Edge), compEdges);
```

```
    cout << "Edges in Minimum Spanning Tree:"
        << "\n==============================\n";

    for (i = 0; i < e; ++i)
    {
      cout << "Edge (" << (result[i].src + 1)
          << ", " << (result[i].dest + 1)
          << ") ==> " << result[i].weight
          << endl;
    }

    return;
  }
};
int main()
{
  int V, E;
  cout << "Enter Number of Vertices: ";
  cin >> V;
  cout << "Enter Number of Edges: ";
  cin >> E;
  cout << endl;
  Graph graph(V, E);
```

```
 graph.KruskalMST();
 return 0;
}
```

```
Enter Number of Vertices: 3
Enter Number of Edges: 4

Edge 1
=============
Source Node: 2
Destination Node: 4
Edge Weight: 3

Invalid Node
```

Krushkals :-

```
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;
#define edge pair<int, int>
class Graph {
  private:
 vector<pair<int, edge> > G;  // graph
 vector<pair<int, edge> > T;  // mst
 int *parent;
 int V;  // number of vertices/nodes in graph
  public:
 Graph(int V);
 void AddWeightedEdge(int u, int v, int w);
```

```
 int find_set(int i);
 void union_set(int u, int v);
 void kruskal();
 void print();
};
Graph::Graph(int V) {
 parent = new int[V];

 //i 0 1 2 3 4 5
 //parent[i] 0 1 2 3 4 5
 for (int i = 0; i < V; i++)
  parent[i] = i;

 G.clear();
 T.clear();
}
void Graph::AddWeightedEdge(int u, int v, int w) {
 G.push_back(make_pair(w, edge(u, v)));
}
int Graph::find_set(int i) {
 if (i == parent[i])
  return i;
 else
  return find_set(parent[i]);
```

```
}

void Graph::union_set(int u, int v) {
  parent[u] = parent[v];
}
void Graph::kruskal() {
  int i, uRep, vRep;
  sort(G.begin(), G.end());
  for (i = 0; i < G.size(); i++) {
    uRep = find_set(G[i].second.first);
    vRep = find_set(G[i].second.second);
    if (uRep != vRep) {
      T.push_back(G[i]);
      union_set(uRep, vRep);
    }
  }
}
void Graph::print() {
  cout << "Edge :"
     << " Weight" << endl;
  for (int i = 0; i < T.size(); i++) {
    cout << T[i].second.first << " - " << T[i].second.second << " : "
       << T[i].first;
    cout << endl;
```

```
  }
}
int main() {
  Graph g(6);
  g.AddWeightedEdge(0, 1, 4);
  g.AddWeightedEdge(0, 2, 4);
  g.AddWeightedEdge(1, 2, 2);
  g.AddWeightedEdge(1, 0, 4);
  g.AddWeightedEdge(2, 0, 4);
  g.AddWeightedEdge(2, 1, 2);
  g.AddWeightedEdge(2, 3, 3);
  g.AddWeightedEdge(2, 5, 2);
  g.AddWeightedEdge(2, 4, 4);
  g.AddWeightedEdge(3, 2, 3);
  g.AddWeightedEdge(3, 4, 3);
  g.AddWeightedEdge(4, 2, 4);
  g.AddWeightedEdge(4, 3, 3);
  g.AddWeightedEdge(5, 2, 2);
  g.AddWeightedEdge(5, 4, 3);
  g.kruskal();
  g.print();
  return 0;
}
```

```
PS C:\Users\ASUS\OneDrive\Desktop\Algorithms Practicals>  & 'c:\
gLauncher.exe' '--stdin=Microsoft-MIEngine-In-pagtxtld.ly0' '--s
pid=Microsoft-MIEngine-Pid-mssqeeqt.a3p' '--dbgExe=C:\Program Fi
Edge : Weight
1 - 2 : 2
2 - 5 : 2
2 - 3 : 3
3 - 4 : 3
0 - 1 : 4
PS C:\Users\ASUS\OneDrive\Desktop\Algorithms Practicals>
```

Q10) Write a program to solve the weighted interval scheduling problem :-

```
#include<iostream>

using namespace std;

#define MAX 20

int M[MAX];


struct Interval

{

   int startTime;

   int finishTime;

   int weight;

};


class WIS{

   Interval I[MAX];
```

```
    public:
    int n;
    WIS(){
      for(int i=0;i<=MAX;i++)
      M[i]= 0;
    }
    void sortIntervals();
    int mComputeOpt(int);
    void input();
    int p(int);
};

int WIS::p(int j)
{
    for(int i=j-1;i>0;i--){
      if(I[i].finishTime <= I[j].startTime)
      return i;
    }
    return 0;
}
void WIS::input(){
    cout<<"enter number of intervals: ";
    cin>>n;
    cout<<"Enter the starting time ,finishing rime and weight value for the
intervals : ";
```

```
    cout<<"\n\nSi Fi Vi\n";
    for(int i=1;i<=n;i++){
        cin>>I[i].startTime;
        cin>>I[i].finishTime;
        cin>>I[i].weight;
    }
}

void WIS::sortIntervals(){
    int i,flag=1;
    Interval temp;

    for(i=1;(i<=n)&&flag;i++){
        flag =0 ;
        for(int j=1;j<n;j++){
            if(I[j+1].finishTime < I[j].finishTime){
                temp = I[j];
                I[j]=I[j+1];
                I[j+1]=temp;
                flag = 1;
            }
        }
    }
    for(i=1;i<=n;i++){
```

```
            for(int j=i+1;j<=n;j++){
                if(I[i].finishTime == I[j].finishTime &&  I[i].startTime>I[j].startTime)
                {
                    temp=I[i];
                    I[i] = I[j];
                    I[j]=temp;
                }
            }
        }
        cout<<"I<i>\t\ts<i>\t\tF<i>\t\tv<i>\n";
        for(int i=1;i<=n;i++){

cout<<"\t\t"<<i<<"\t\t"<<I[i].startTime<<"\t\t"<<I[i].finishTime<<"\t\t"<<I[i].weig
ht<<"\n";
        }
    }


    int WIS::mComputeOpt(int j){
        if(j==0){
            return 0;
        }else if(M[j]){
            return M[j];
        }
        else{
            M[j]=max((I[j].weight+mComputeOpt(p(j))),mComputeOpt(j-1));
```

```
    }
    return M[j];
}

int main(){
    WIS job;
    job.input();
    cout<<"\nSorted input: ";
    job.sortIntervals();
     cout<<endl;
     for(int i=1;i<=job.n;i++)
     cout<<"opt["<<i<<"]\t";

     cout<<endl;
     for(int i=1;i<=job.n;i++){
       cout<<job.mComputeOpt(i)<<"\t";
       if(i==job.n){
          cout<<endl;
          cout<<"max = "<<job.mComputeOpt(i);
       }
     }

     return 0;
}
```

```
enter number of intervals: 4
Enter the starting time ,finishing rime and weight value for the intervals :

Si Fi Vi
1 2 3
2 4 5
3 4 5
4 6 7

Sorted input: I<i>            s<i>          F<i>          v<i>
              1               1             2             3
              2               2             4             5
              3               3             4             5
              4               4             6             7

opt[1]  opt[2]  opt[3]  opt[4]
3       8       8       15
max = 15
```

Q11) Write a program to solve the 0-1 knapsack problem :-

```
#include <bits/stdc++.h>

using namespace std;

int max(int a, int b) { return (a > b) ? a : b; }

int knapSack(int W, int wt[], int val[], int n)

{

if (n == 0 || W == 0)

 return 0;

if (wt[n - 1] > W)

return knapSack(W, wt, val, n - 1);

else

return max(val[n - 1] + knapSack(W - wt[n - 1], wt, val, n - 1),knapSack(W, wt, val, n - 1));

}

int main()
```

```
{
int profit[] = { 90, 20, 40 };
int weight[] = { 10, 30, 30 };
int W = 50;
int n = sizeof(profit) / sizeof(profit[0]);
cout << knapSack(W, weight, profit, n);
return 0;
}
```

```
PS C:\Users\ASUS\OneDrive\Desktop\Algorithms Practicals>  & 'c:\Users\ASUS\.vscode\extensi
gLauncher.exe' '--stdin=Microsoft-MIEngine-In-c3wthtwk.5ku' '--stdout=Microsoft-MIEngine-
pid=Microsoft-MIEngine-Pid-kw04dmzq.rkz' '--dbgExe=C:\Program Files (x86)\mingw-w64\i686-
130
PS C:\Users\ASUS\OneDrive\Desktop\Algorithms Practicals>
```