DESIGN & ANALYSIS OF ALGORITHMS

PRACTICAL FILE

SUBMITTED BY :- SANJANA KUMARI

ROLL NO – 20570015

COURSE :- B.Sc. COMPUTER SCIENCE (HONS)

Q1. Implement Insertion Sort

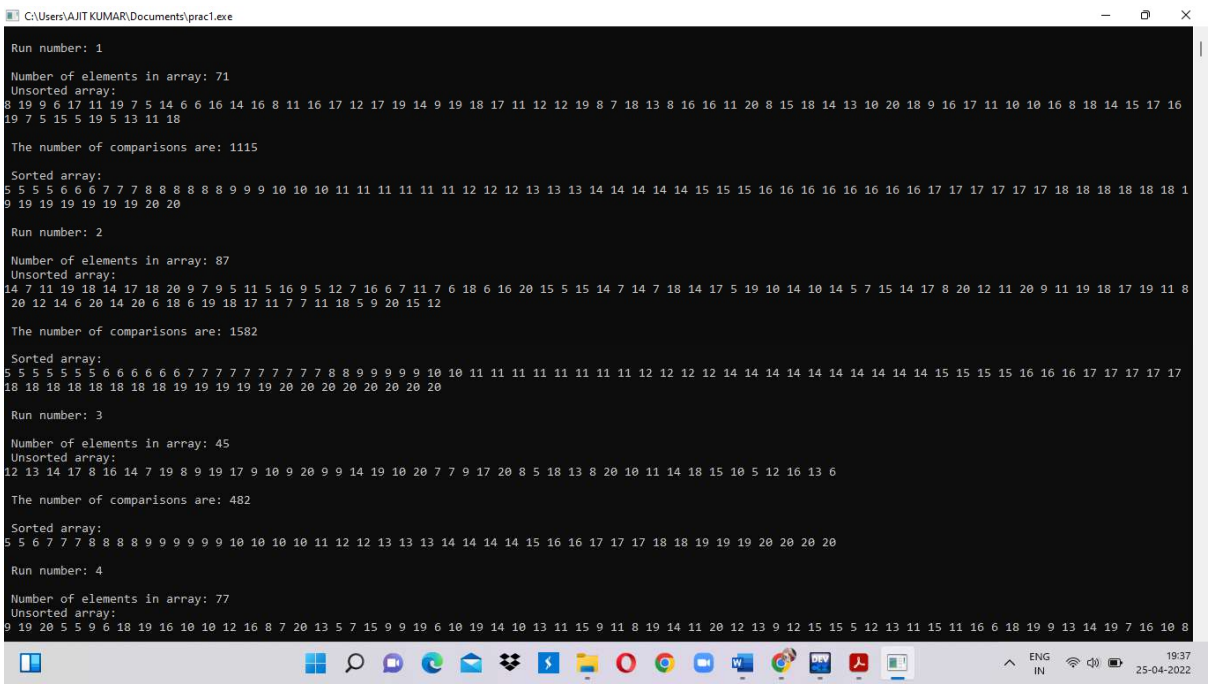
//code

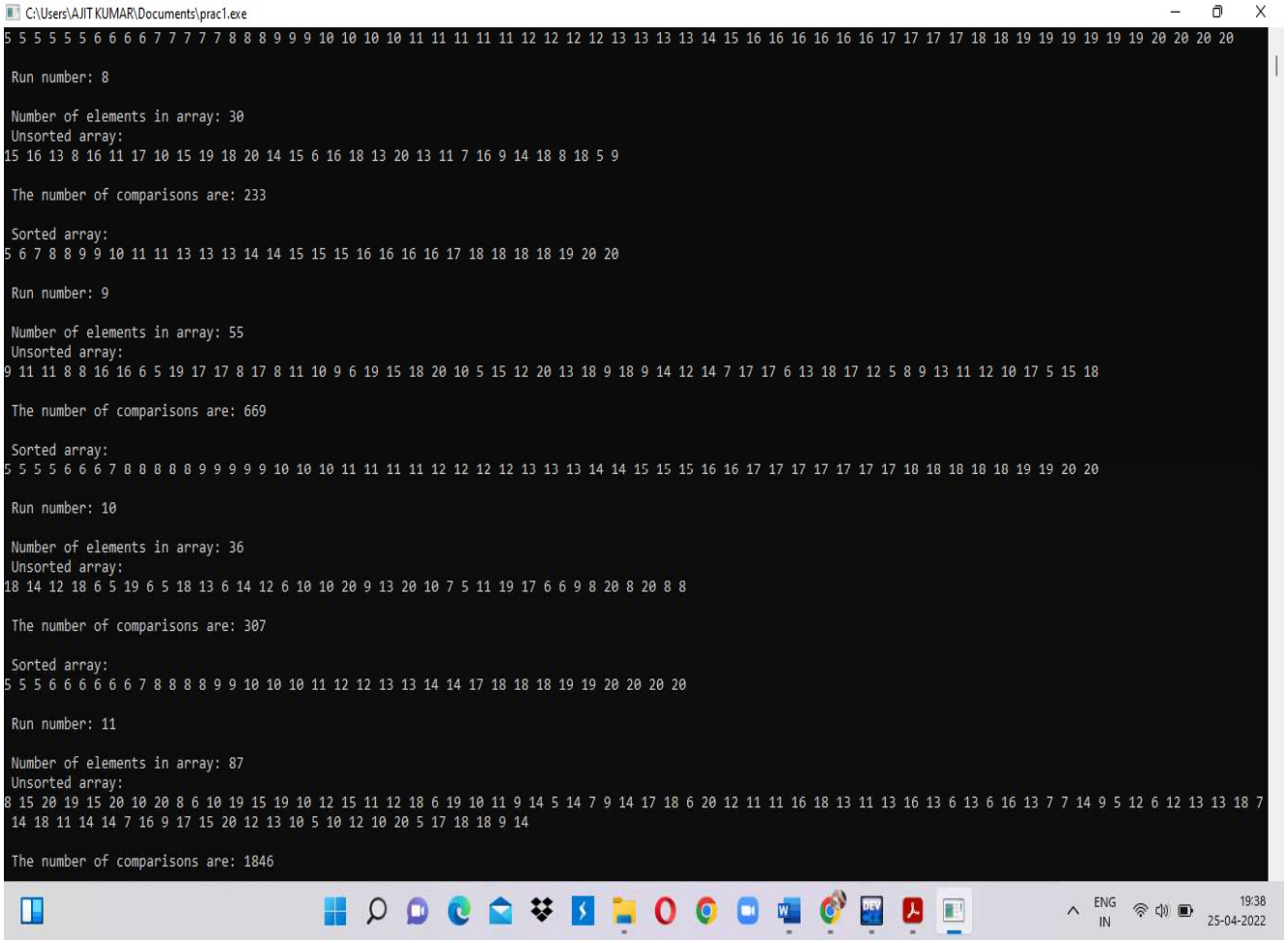
```
#include<iostream>
#include<cstdlib>
#include<ctime>
using namespace std;
int total_sizes[100], total_comparisons[100];
void InsertionSort(int array[], int n_ele, int pos)
{
int key, i, j, count=0;
for(j = 1; j < n_ele; j++)
{
key = array[j];
i = j - 1;
while( i > -1 && array[i] > key )
{
array[i+1] = array[i];
i = i - 1;
array[i + 1] = key;
count++;
}
}

cout<<"\n The number of comparisons are: "<<count<<endl;
total_comparisons[pos] = count;
}
int random_no(int lower, int upper)
{
```

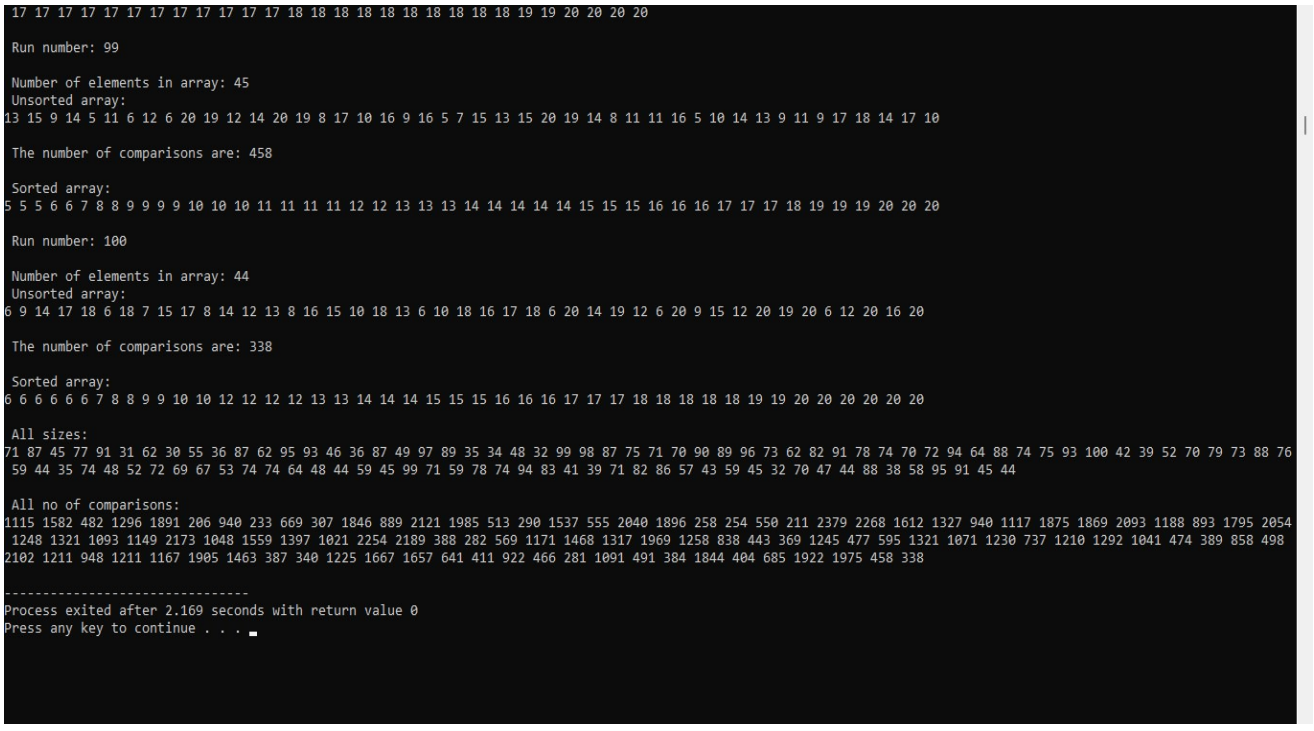
```
int num = ( rand() % (upper - lower + 1)) + lower;
return num;
}
void sort_elements(int pos)
{
int n = random_no(30, 100);
cout<<"\n Number of elements in array: "<<n;
int arr[n];
total_sizes[pos] = n;
cout<<"\n Unsorted array:"<<endl;
for(int i = 0; i < n; i++)
{
arr[i] = random_no(5, 20);
cout<<arr[i]<<" ";
}
cout<<endl;

InsertionSort(arr, n, pos);
cout<<"\n Sorted array:"<<endl;
for(int i = 0; i < n; i++)
cout<<arr[i]<<" ";
cout<<endl;
}
int main()
{
int x = 0;
while(x < 100)
{
cout<<"\n Run number: "<<x+1<<endl;
```

```
sort_elements(x);
x++;
}
cout<<"\n All sizes:"<<endl;
for(int i = 0; i < 100; i++)
cout<<total_sizes[i]<<" ";
cout<<endl;
cout<<"\n All no of comparisons:"<<endl;

for(int i = 0; i < 100; i++)
cout<<total_comparisons[i]<<" ";
cout<<endl;
return 0;
}
```

//output

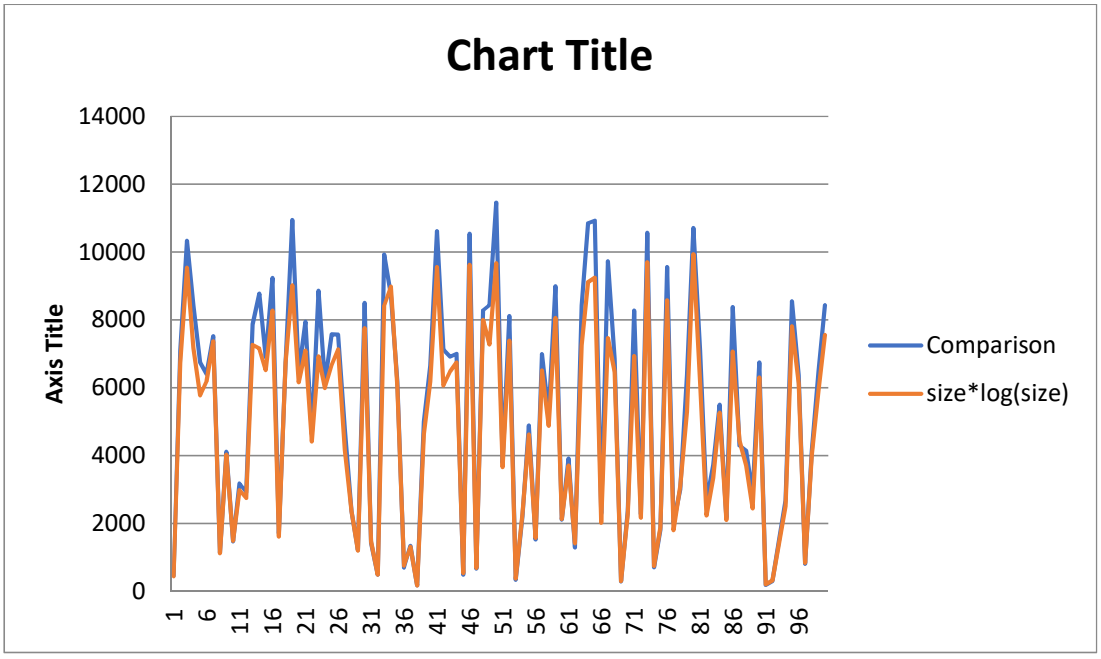



C:\Users\AJIT KUMAR\Documents\prac1.exe
5 5 5 5 5 5 6 6 6 6 7 7 7 7 7 8 8 8 9 9 9 10 10 10 10 11 11 11 11 11 12 12 12 12 13 13 13 13 14 15 16 16 16 16 16 16 17 17 17 17 18 18 19 19 19 19 19 19 20 20 20 20
Run number: 8
Number of elements in array: 30
Unsorted array:
15 16 13 8 16 11 17 10 15 19 18 20 14 15 6 16 18 13 20 13 11 7 16 9 14 18 8 18 5 9
The number of comparisons are: 233
Sorted array:
5 6 7 8 8 9 9 10 11 11 13 13 13 14 14 15 15 15 16 16 16 16 17 18 18 18 18 19 20 20
Run number: 9
Number of elements in array: 55
Unsorted array:
9 11 11 8 8 16 16 6 5 19 17 17 8 17 8 11 10 9 6 19 15 18 20 10 5 15 12 20 13 18 9 18 9 14 12 14 7 17 17 6 13 18 17 12 5 8 9 13 11 12 10 17 5 15 18
The number of comparisons are: 669
Sorted array:
5 5 5 5 6 6 6 7 8 8 8 8 8 9 9 9 9 9 10 10 10 11 11 11 11 12 12 12 12 13 13 13 14 14 15 15 15 16 16 17 17 17 17 17 17 17 18 18 18 18 18 19 19 20 20
Run number: 10
Number of elements in array: 36
Unsorted array:
18 14 12 18 6 5 19 6 5 18 13 6 14 12 6 10 10 20 9 13 20 10 7 5 11 19 17 6 6 9 8 20 8 20 8 8
The number of comparisons are: 307
Sorted array:
5 5 5 6 6 6 6 6 6 7 8 8 8 8 9 9 10 10 10 11 12 12 13 13 14 14 17 18 18 18 19 19 20 20 20 20
Run number: 11
Number of elements in array: 87
Unsorted array:
8 15 20 19 15 20 10 20 8 6 10 19 15 19 10 12 15 11 12 18 6 19 10 11 9 14 5 14 7 9 14 17 18 6 20 12 11 11 16 18 13 11 13 16 13 6 13 6 16 13 7 7 14 9 5 12 6 12 13 13 18 7
14 18 11 14 14 7 16 9 17 15 20 12 13 10 5 10 12 10 20 5 17 18 18 9 14
The number of comparisons are: 1846
ENG
IN
19:38
25-04-2022


17 17 17 17 17 17 17 17 17 17 17 17 18 18 18 18 18 18 18 18 18 18 19 19 20 20 20 20
Run number: 99
Number of elements in array: 45
Unsorted array:
13 15 9 14 5 11 6 12 6 20 19 12 14 20 19 8 17 10 16 9 16 5 7 15 13 15 20 19 14 8 11 11 16 5 10 14 13 9 11 9 17 18 14 17 10
The number of comparisons are: 458
Sorted array:
5 5 5 6 6 7 8 8 9 9 9 9 10 10 10 11 11 11 11 12 12 13 13 13 14 14 14 14 14 15 15 15 16 16 16 17 17 17 18 19 19 19 20 20 20
Run number: 100
Number of elements in array: 44
Unsorted array:
6 9 14 17 18 6 18 7 15 17 8 14 12 13 8 16 15 10 18 13 6 10 18 16 17 18 6 20 14 19 12 6 20 9 15 12 20 19 20 6 12 20 16 20
The number of comparisons are: 338
Sorted array:
6 6 6 6 6 6 7 8 8 9 9 10 10 12 12 12 12 13 13 14 14 14 15 15 15 16 16 16 17 17 17 18 18 18 18 18 19 19 20 20 20 20 20 20
All sizes:
71 87 45 77 91 31 62 30 55 36 87 62 95 93 46 36 87 49 97 89 35 34 48 32 99 98 87 75 71 70 90 89 96 73 62 82 91 78 74 70 72 94 64 88 74 75 93 100 42 39 52 70 79 73 88 76
59 44 35 74 48 52 72 69 67 53 74 74 64 48 44 59 45 99 71 59 78 74 94 83 41 39 71 82 86 57 43 59 45 32 70 47 44 88 38 58 95 91 45 44
All no of comparisons:
1115 1582 482 1296 1891 206 940 233 669 307 1846 889 2121 1985 513 290 1537 555 2040 1896 258 254 550 211 2379 2268 1612 1327 940 1117 1875 1869 2093 1188 893 1795 2054
1248 1321 1093 1149 2173 1048 1559 1397 1021 2254 2189 388 282 569 1171 1468 1317 1969 1258 838 443 369 1245 477 595 1321 1071 1230 737 1210 1292 1041 474 389 858 498
2102 1211 948 1211 1167 1905 1463 387 340 1225 1667 1657 641 411 922 466 281 1091 491 384 1844 404 685 1922 1975 458 338

Process exited after 2.169 seconds with return value 0
Press any key to continue . . .

//graph



Q2. Implement Heap Sort

(The program should report the number of comparisons)

//CODE

```
#include <iostream>
#include<cstdlib>
#include<ctime>
using namespace std;
int total_sizes[100], total_comparisons[100];
int PARENT(int i)
{
return (i/2);
}
int LEFT(int i)
{
return (2 * i);
}
int RIGHT(int i)
```

```
{
return (2 * i + 1);
}
void swap(int *a, int *b)

{
int temp = *a;
*a = *b;
*b = temp;
}
int Max_Heapify(int arr[], int i, int hsize)
{
int largest, count = 0;
int l = LEFT(i);
int r = RIGHT(i);
if( (l <= hsize) && (arr[l] > arr[i]) )
{
largest = l;
count++;
}
else
{
largest = i;
count++;
}
if( (r <= hsize) && (arr[r] > arr[largest]) )
{
largest = r;

count++;
}
```

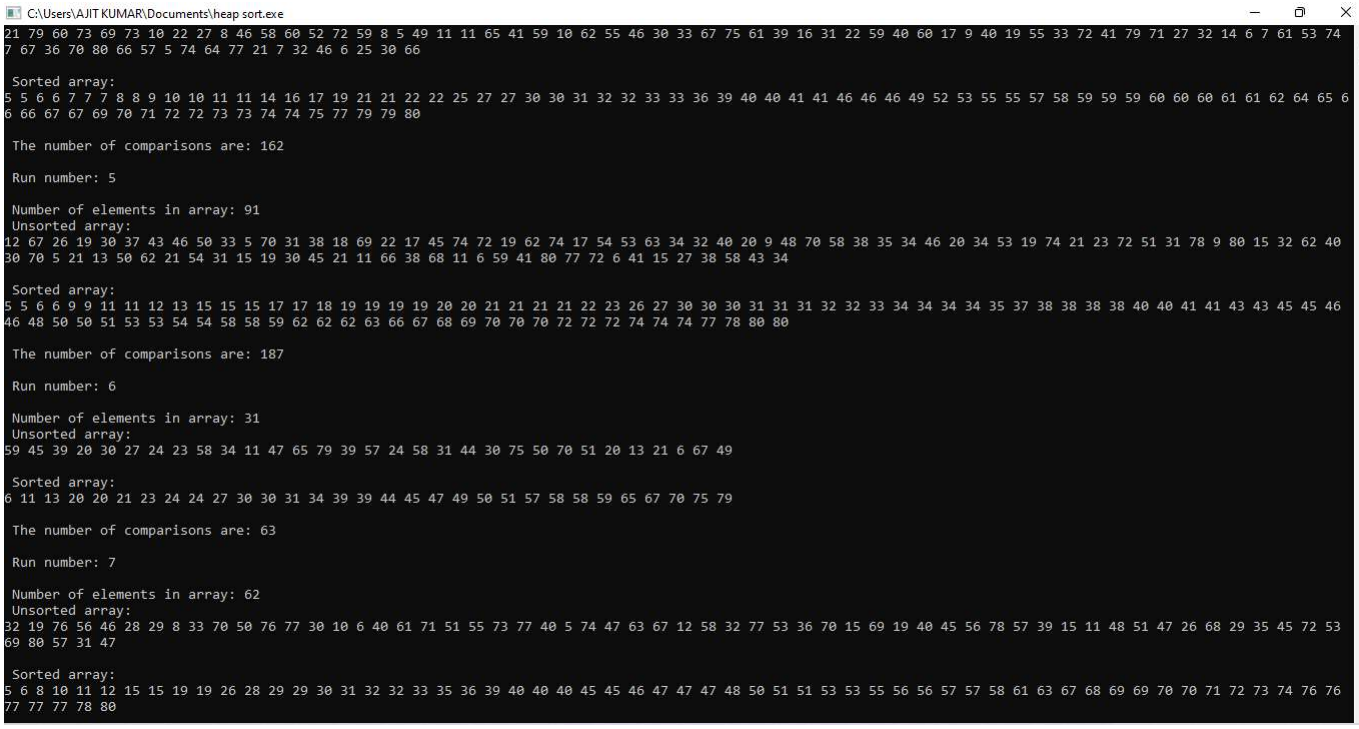
```
if(largest != i)
{
swap(arr[i], arr[largest]);
Max_Heapify(arr, largest, hsize);
}
return count;
}
int Build_Max_Heap(int arr[], int hsize, int n)
{
int count1[n], bmh_count = 0;
hsize = n;
for(int i = n/2; i >= 1; i--)
{
count1[i] = Max_Heapify(arr, i, hsize);
bmh_count += count1[i];
}
return bmh_count;
}
int HeapSort(int arr[], int hsize, int n)
{
int count2[n], hs_count = 0;

int b_count = Build_Max_Heap(arr, hsize, n);
for (int i = n; i >= 2; i--)
{
swap(arr[1], arr[i]);
hsize -= 1;
count2[i] = Max_Heapify(arr, 1, hsize);
hs_count += count2[i];
}
return (b_count + hs_count);
```

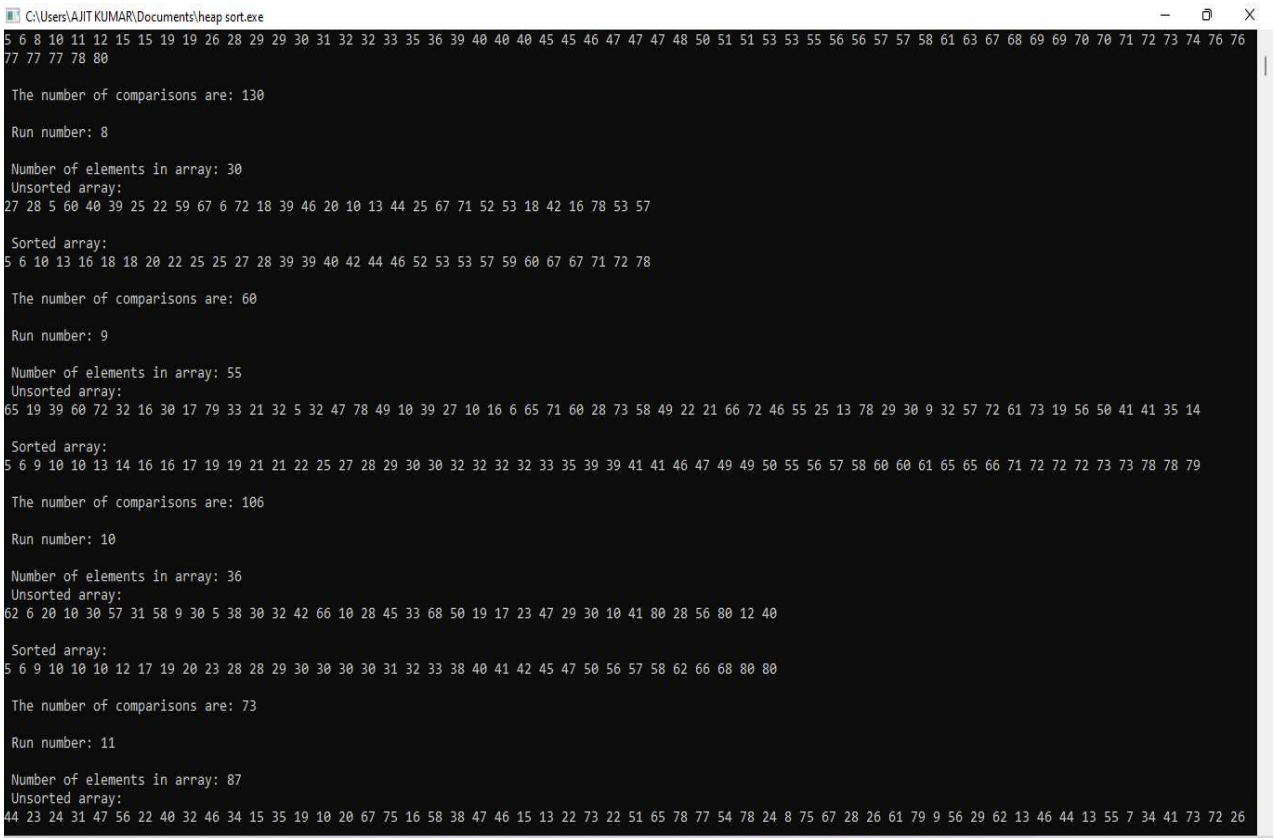
```
}
int random_no(int lower, int upper)
{
int num = ( rand() % (upper - lower + 1)) + lower;
return num;
}
void sort_elements(int pos)
{
int n = random_no(30, 100);
cout<<"\n Number of elements in array: "<<n;
int arr[n];
int hsize = n;
total_sizes[pos] = n;

cout<<"\n Unsorted array:"<<endl;
for(int i = 1; i <= n; i++)
{
arr[i] = random_no(5, 80);
cout<<arr[i]<<" ";
}
cout<<endl;
int count = HeapSort(arr, hsize, n);
cout<<"\n Sorted array:"<<endl;
for(int i = 1; i <= n; i++)
cout<<arr[i]<<" ";
cout<<endl;
cout<<"\n The number of comparisons are: "<<count<<endl;
total_comparisons[pos] = count;
}
int main()
{
```

```
int x = 0;
while(x < 100)
{
cout<<"\n Run number: "<<x+1<<endl;
sort_elements(x);

x++;
}
cout<<"\n All sizes:"<<endl;
for(int i = 1; i <= 100; i++)
cout<<total_sizes[i]<<" ";
cout<<endl;
cout<<"\n All number of comparisons:"<<endl;
for(int i = 1; i <= 100; i++)
cout<<total_comparisons[i]<<" ";
cout<<endl;
return 0;
}
```

//OUTPUT

```
 Run number: 1

 Number of elements in array: 71
 Unsorted array:
80 31 57 22 73 7 27 63 73 10 30 30 36 10 40 36 15 44 45 20 17 31 6 69 75 22 25 39 64 56 71 32 67 50 5 60 8 16 19 56 28 15 14 46 33 22 40 62 33 16 61 63 6 54 20 64 18 22
 23 9 76 79 23 65 15 77 55 41 5 15 22

 Sorted array:
5 5 6 6 7 8 9 10 10 14 15 15 15 15 16 16 17 18 19 20 20 22 22 22 22 22 23 23 25 27 28 30 30 31 31 32 33 33 36 36 39 40 40 41 44 45 46 50 54 55 56 56 57 60 61 62 63 63 6
4 64 65 67 69 71 73 73 75 76 77 79 80

 The number of comparisons are: 144

 Run number: 2

 Number of elements in array: 87
 Unsorted array:
46 55 79 39 22 78 57 26 12 73 47 65 65 31 9 28 17 77 68 7 80 70 35 11 35 58 54 50 36 8 23 61 55 74 31 62 47 10 22 25 77 43 58 30 74 10 25 51 71 46 21 16 72 24 75 8 49 7
1 15 74 69 63 15 28 24 76 18 34 56 18 52 54 22 66 35 58 49 43 23 75 39 78 65 69 68 71 64

 Sorted array:
7 8 8 9 10 10 11 12 15 15 16 17 18 18 21 22 22 22 23 23 24 24 25 25 26 28 28 30 31 31 34 35 35 35 36 39 39 43 43 46 46 47 47 49 49 50 51 52 54 54 55 55 56 57 58 58 58 6
1 62 63 64 65 65 65 66 68 68 69 69 70 71 71 71 72 73 74 74 74 75 75 76 77 77 78 78 79 80

 The number of comparisons are: 176

 Run number: 3

 Number of elements in array: 45
 Unsorted array:
56 41 22 45 28 12 18 35 35 36 61 31 5 41 10 69 24 65 17 78 27 58 28 51 51 65 69 44 16 9 78 29 68 28 70 27 10 18 19 26 21 16 52 9 10

 Sorted array:
5 9 9 10 10 10 12 16 16 17 18 18 19 21 22 24 26 27 27 28 28 28 29 31 35 35 36 41 41 44 45 51 51 52 56 58 61 65 65 68 69 69 70 78 78

 The number of comparisons are: 89

 Run number: 4

 Number of elements in array: 77
 Unsorted array:
21 79 60 73 69 73 10 22 27 8 46 58 60 52 72 59 8 5 49 11 11 65 41 59 10 62 55 46 30 33 67 75 61 39 16 31 22 59 40 60 17 9 40 19 55 33 72 41 79 71 27 32 14 6 7 61 53 74
```

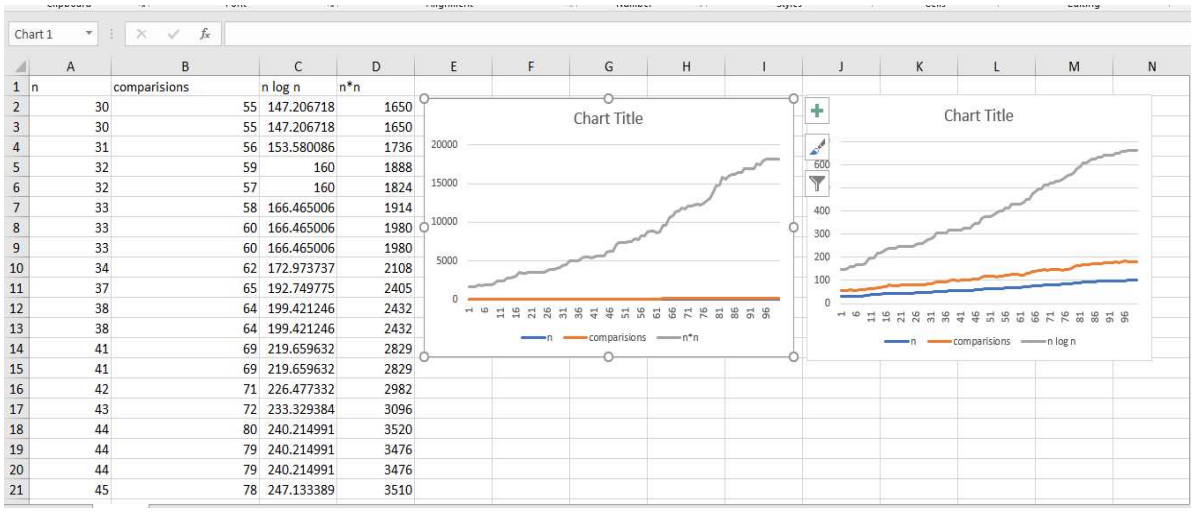

C:\Users\AJIT KUMAR\Documents\heap sort.exe
21 79 60 73 69 73 10 22 27 8 46 58 60 52 72 59 8 5 49 11 11 65 41 59 10 62 55 46 30 33 67 75 61 39 16 31 22 59 40 60 17 9 40 19 55 33 72 41 79 71 27 32 14 6 7 61 53 74
7 67 36 70 80 66 57 5 74 64 77 21 7 32 46 6 25 30 66
Sorted array:
5 5 6 6 7 7 7 8 8 9 10 10 11 11 14 16 17 19 21 21 22 22 25 27 27 30 30 31 32 32 33 33 36 39 40 40 41 41 46 46 46 49 52 53 55 55 57 58 59 59 59 60 60 60 61 61 62 64 65 6
6 66 67 67 69 70 71 72 72 73 73 74 74 75 77 79 79 80
The number of comparisons are: 162
Run number: 5
Number of elements in array: 91
Unsorted array:
12 67 26 19 30 37 43 46 50 33 5 70 31 38 18 69 22 17 45 74 72 19 62 74 17 54 53 63 34 32 40 20 9 48 70 58 38 35 34 46 20 34 53 19 74 21 23 72 51 31 78 9 80 15 32 62 40
30 70 5 21 13 50 62 21 54 31 15 19 30 45 21 11 66 38 68 11 6 59 41 80 77 72 6 41 15 27 38 58 43 34
Sorted array:
5 5 6 6 9 9 11 11 12 13 15 15 15 17 17 18 19 19 19 20 20 21 21 21 21 22 23 26 27 30 30 30 31 31 31 32 32 33 34 34 34 34 35 37 38 38 38 38 40 40 41 41 43 43 45 45 46
46 48 50 50 51 53 53 54 54 58 58 59 62 62 62 63 66 67 68 69 70 70 70 72 72 72 74 74 74 77 78 80 80
The number of comparisons are: 187
Run number: 6
Number of elements in array: 31
Unsorted array:
59 45 39 20 30 27 24 23 58 34 11 47 65 79 39 57 24 58 31 44 30 75 50 70 51 20 13 21 6 67 49
Sorted array:
6 11 13 20 20 21 23 24 24 27 30 30 31 34 39 39 44 45 47 49 50 51 57 58 58 59 65 67 70 75 79
The number of comparisons are: 63
Run number: 7
Number of elements in array: 62
Unsorted array:
32 19 76 56 46 28 29 8 33 70 50 76 77 30 10 6 40 61 71 51 55 73 77 40 5 74 47 63 67 12 58 32 77 53 36 70 15 69 19 40 45 56 78 57 39 15 11 48 51 47 26 68 29 35 45 72 53
69 80 57 31 47
Sorted array:
5 6 8 10 11 12 15 15 19 19 26 28 29 29 30 31 32 32 33 35 36 39 40 40 40 45 45 46 47 47 47 48 50 51 51 53 53 55 56 56 57 57 58 61 63 67 68 69 69 70 70 71 72 73 74 76 76
77 77 77 78 80


C:\Users\AJIT KUMAR\Documents\heap sort.exe
5 6 8 10 11 12 15 15 19 19 26 28 29 29 30 31 32 32 33 35 36 39 40 40 40 45 45 46 47 47 47 48 50 51 51 53 53 55 56 56 57 57 58 61 63 67 68 69 69 70 70 71 72 73 74 76 76
77 77 77 78 80
The number of comparisons are: 130
Run number: 8
Number of elements in array: 30
Unsorted array:
27 28 5 60 40 39 25 22 59 67 6 72 18 39 46 20 10 13 44 25 67 71 52 53 18 42 16 78 53 57
Sorted array:
5 6 10 13 16 18 18 20 22 25 25 27 28 39 39 40 42 44 46 52 53 53 57 59 60 67 67 71 72 78
The number of comparisons are: 60
Run number: 9
Number of elements in array: 55
Unsorted array:
65 19 39 60 72 32 16 30 17 79 33 21 32 5 32 47 78 49 10 39 27 10 16 6 65 71 60 28 73 58 49 22 21 66 72 46 55 25 13 78 29 30 9 32 57 72 61 73 19 56 50 41 41 35 14
Sorted array:
5 6 9 10 10 13 14 16 16 17 19 19 21 21 22 25 27 28 29 30 30 32 32 32 32 33 35 39 39 41 41 46 47 49 49 50 55 56 57 58 60 60 61 65 65 66 71 72 72 72 73 73 78 78 79
The number of comparisons are: 106
Run number: 10
Number of elements in array: 36
Unsorted array:
62 6 20 10 30 57 31 58 9 30 5 38 30 32 42 66 10 28 45 33 68 50 19 17 23 47 29 30 10 41 80 28 56 80 12 40
Sorted array:
5 6 9 10 10 10 12 17 19 20 23 28 28 29 30 30 30 30 31 32 33 38 40 41 42 45 47 50 56 57 58 62 66 68 80 80
The number of comparisons are: 73
Run number: 11
Number of elements in array: 87
Unsorted array:
44 23 24 31 47 56 22 40 32 46 34 15 35 19 10 20 67 75 16 58 38 47 46 15 13 22 73 22 51 65 78 77 54 78 24 8 75 67 28 26 61 79 9 56 29 62 13 46 44 13 55 7 34 41 73 72 26

```
C:\Users\AJIT KUMAR\Documents\heap sort.exe
44 23 24 31 47 56 22 40 32 46 34 15 35 19 10 20 67 75 16 58 38 47 46 15 13 22 73 22 51 65 78 77 54 78 24 8 75 67 28 26 61 79 9 56 29 62 13 46 44 13 55 7 34 41 73 72 26
48 57 45 38 19 46 10 79 26 54 15 36 69 9 15 36 24 25 10 37 26 8 34 68 17 37 74 74 73 30

 Sorted array:
7 8 8 9 9 10 10 10 13 13 13 15 15 15 15 16 17 19 19 20 22 22 22 23 24 24 24 25 26 26 26 28 29 30 31 32 34 34 34 35 36 36 37 37 38 38 40 41 44 44 45 46 46 46 46 47 47
 48 51 54 54 55 56 56 57 58 61 62 65 67 67 68 69 72 73 73 73 74 74 75 75 77 78 78 79 79

 The number of comparisons are: 176

 Run number: 12

 Number of elements in array: 62
 Unsorted array:
12 58 66 9 78 71 10 78 80 24 10 57 65 58 65 32 15 61 64 61 48 19 75 47 60 64 5 52 22 20 11 12 47 60 58 5 9 11 24 18 72 21 55 41 71 54 40 24 80 27 51 80 61 49 12 9 42 60
 42 65 71 63

 Sorted array:
5 5 9 9 9 10 10 11 11 12 12 12 15 18 19 20 21 22 24 24 24 27 32 40 41 42 42 47 47 48 49 51 52 54 55 57 58 58 58 60 60 60 61 61 61 63 64 64 65 65 65 66 71 71 71 72 75 78
 78 80 80 80

 The number of comparisons are: 129

 Run number: 13

 Number of elements in array: 95
 Unsorted array:
67 12 6 22 65 58 77 70 53 51 59 30 56 9 5 25 12 75 52 38 8 64 36 53 70 36 22 33 40 44 41 60 26 70 20 38 50 73 23 5 69 22 15 15 69 15 11 50 80 79 13 69 16 57 40 28 7 66
51 54 22 57 21 16 20 38 71 9 64 67 51 32 54 68 74 58 77 45 20 33 15 24 58 35 41 65 73 63 19 34 38 33 22 35 25

 Sorted array:
5 5 6 7 8 9 9 11 12 12 13 15 15 15 15 16 16 19 20 20 20 21 22 22 22 22 22 23 24 25 25 26 28 30 32 33 33 33 34 35 35 36 36 38 38 38 38 40 40 41 41 44 45 50 50 51 51 51 5
2 53 53 54 54 56 57 57 58 58 58 59 60 63 64 64 65 65 66 67 67 68 69 69 69 70 70 70 71 73 73 74 75 77 77 79 80

 The number of comparisons are: 193

 Run number: 14

 Number of elements in array: 93
 Unsorted array:
78 78 5 73 38 39 71 53 45 77 63 43 68 44 72 71 62 38 40 38 11 11 59 43 55 30 22 65 5 29 47 46 63 44 71 12 49 77 33 10 38 58 17 21 33 43 67 39 68 54 69 52 33 8 10 11 70
70 25 71 54 51 21 11 12 36 70 7 48 13 52 36 64 62 58 61 9 10 10 15 46 35 15 42 31 78 7 49 10 12 38 76 42

 Sorted array:
```

```
5 7 7 7 7 7 8 9 10 11 12 12 14 14 14 14 16 17 17 18 19 19 20 21 22 22 23 24 26 26 29 29 29 30 30 30 32 33 33 33 34 35 36 37 38 38 38 38 40 43 46 46 46 46 49 50 51 52 54
 54 54 55 58 58 59 60 61 64 64 64 65 66 69 69 69 70 71 71 71 72 72 73 73 74 74 74 75 77 77 79 80

 The number of comparisons are: 185

 Run number: 99

 Number of elements in array: 45
 Unsorted array:
49 7 13 30 37 47 54 52 22 68 43 8 6 20 23 40 29 66 44 33 72 49 67 63 69 67 64 7 18 36 19 27 20 13 50 10 49 13 55 37 13 70 26 33 46

 Sorted array:
6 7 7 8 10 13 13 13 13 18 19 20 20 22 23 26 27 29 30 33 33 36 37 37 40 43 44 46 47 49 49 49 50 52 54 55 63 64 66 67 67 68 69 70 72

 The number of comparisons are: 86

 Run number: 100

 Number of elements in array: 44
 Unsorted array:
22 73 14 53 78 14 18 15 35 65 56 22 44 73 24 64 55 58 34 17 38 38 46 20 49 6 10 8 78 71 56 70 8 33 67 32 28 11 48 62 44 8 20 36

 Sorted array:
6 8 8 8 10 11 14 14 15 17 18 20 20 22 22 24 28 32 33 34 35 36 38 38 44 44 46 48 49 53 55 56 56 58 62 64 65 67 70 71 73 73 78 78

 The number of comparisons are: 83

 All sizes:
87 45 77 91 31 62 30 55 36 87 62 95 93 46 36 87 49 97 89 35 34 48 32 99 98 87 75 71 70 90 89 96 73 62 82 91 78 74 70 72 94 64 88 74 75 93 100 42 39 52 70 79 73 88 76 59
 44 35 74 48 52 72 69 67 53 74 74 64 48 44 59 45 99 71 59 78 74 94 83 41 39 71 82 86 57 43 59 45 32 70 47 44 88 38 58 95 91 45 44 0

 All number of comparisons:
176 89 162 187 63 130 60 106 73 176 129 193 191 96 71 179 98 198 185 68 69 101 66 197 196 173 152 152 152 187 185 202 157 133 173 184 159 155 138 146 187 132 182 159 15
4 190 200 82 79 100 148 166 150 179 154 120 89 65 149 92 105 153 138 137 104 145 152 130 90 90 119 85 204 148 129 165 150 191 170 84 75 149 172 176 122 84 120 89 63 147
 90 88 176 82 118 188 185 86 83 0

--------------------------------
Process exited after 2.608 seconds with return value 0
Press any key to continue . . .
```

//GRAPH



	A	B	C	D
1	n	comparisions	n log n	n*n
2	30	55	147.206718	1650
3	30	55	147.206718	1650
4	31	56	153.580086	1736
5	32	59	160	1888
6	32	57	160	1824
7	33	58	166.465006	1914
8	33	60	166.465006	1980
9	33	60	166.465006	1980
10	34	62	172.973737	2108
11	37	65	192.749775	2405
12	38	64	199.421246	2432
13	38	64	199.421246	2432
14	41	69	219.659632	2829
15	41	69	219.659632	2829
16	42	71	226.477332	2982
17	43	72	233.329384	3096
18	44	80	240.214991	3520
19	44	79	240.214991	3476
20	44	79	240.214991	3476
21	45	78	247.133389	3510

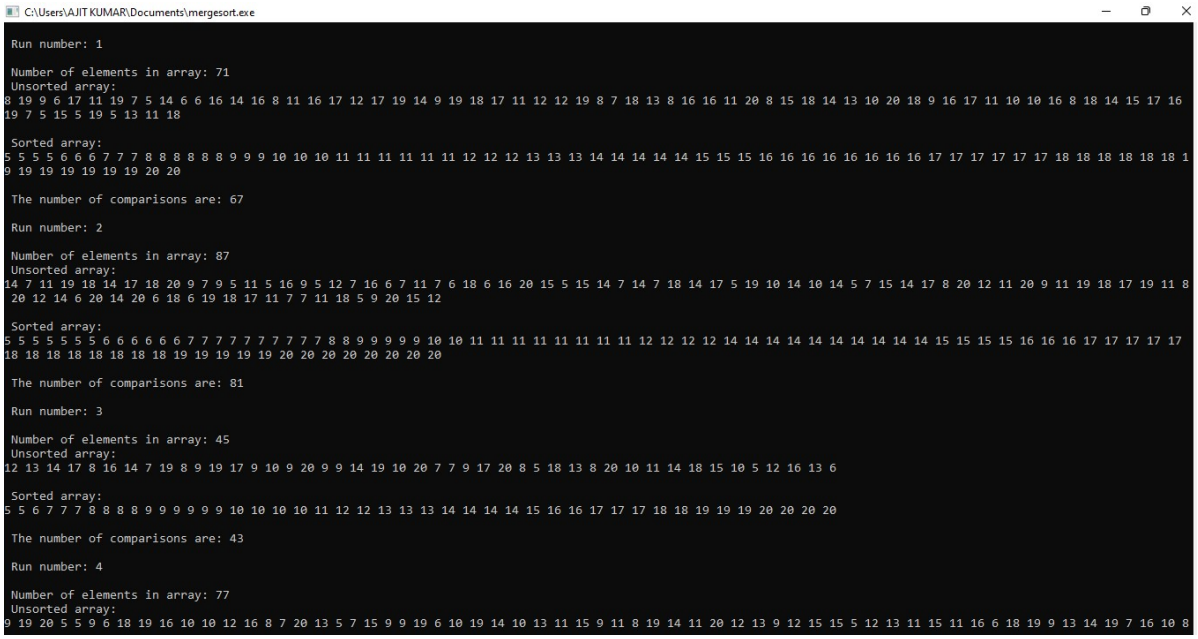
Q4. Implement Merge Sort

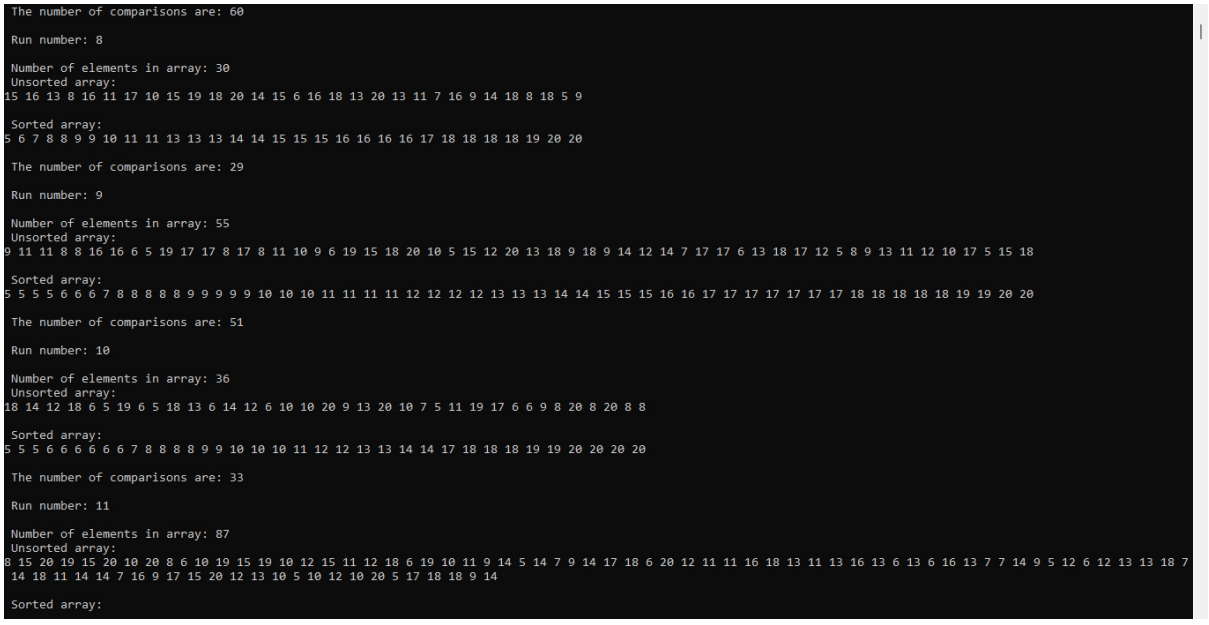
```
#include<iostream>
#include<cstdlib>
#include<ctime>
using namespace std;
int total_sizes[100], total_comparisons[100];
int merge(int arr[], int f, int m, int l)
{
int count;
int n1 = m - f + 1;
int n2 = l - m;
int L[n1], R[n2];
for(int i = 0; i < n1; i++ )
L[i] = arr[f + i];
for(int j = 0; j < n1; j++ )
R[j] = arr[m + 1 + j];
int i = 0, j = 0, k = f; // Indices of 3 sub arrays.

while (i < n1 && j < n2)
{
if (L[i] <= R[j])
{
arr[k] = L[i];
i++;
}
else
{
arr[k] = R[j];
```

```
j++;
}
k++;
count++;
}
// Copy the remaining elements
while (i < n1)
{
arr[k] = L[i];
i++;
k++;
}
while (j < n2)

{
arr[k] = R[j];
j++;
k++;
}
return count;
}
int mergesort(int arr[], int f, int l)
{
int count1;
if(f < l)
{
int m = (f + l)/2;
mergesort(arr, f, m);
mergesort(arr, m+1, l);
```

```
count1 = merge(arr, f, m, l);
}
return count1;
}
int random_no(int lower, int upper)
{
int num = ( rand() % (upper - lower + 1)) + lower;
return num;
}

void sort_elements(int pos)
{
int size = random_no(30, 100);
cout<<"\n Number of elements in array: "<<size;
int arr[size];
total_sizes[pos] = size;
cout<<"\n Unsorted array:"<<endl;
for(int i = 0; i < size; i++)
{
arr[i] = random_no(5, 20);
cout<<arr[i]<<" ";
}
cout<<endl;
int count = mergesort(arr, 0, size - 1);
cout<<"\n Sorted array:"<<endl;
for(int i = 0; i < size; i++)
cout<<arr[i]<<" ";
cout<<endl;
cout<<"\n The number of comparisons are: "<<count<<endl;
```

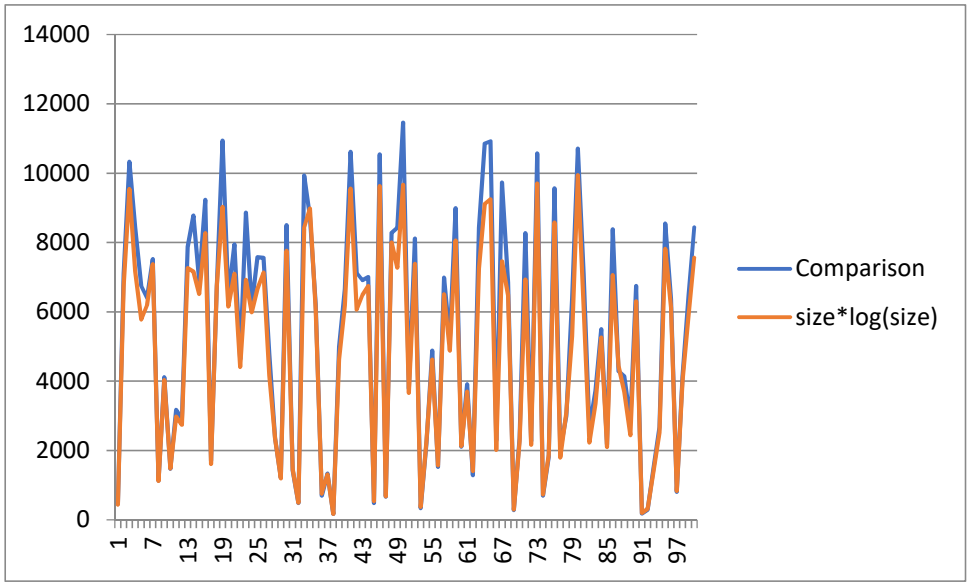
```
total_comparisons[pos] = count;


}
int main()
{
int x = 0;
while(x < 100)
{
cout<<"\n Run number: "<<x+1<<endl;
sort_elements(x);
x++;
}
cout<<"\n All sizes:"<<endl;
for(int i = 0; i < 100; i++)
cout<<total_sizes[i]<<" ";
cout<<endl;
cout<<"\n All no of comparisons:"<<endl;
for(int i = 0; i < 100; i++)
cout<<total_comparisons[i]<<" ";
cout<<endl;
return 0;
}
//output
```


C:\Users\AJIT KUMAR\Documents\mergesort.exe
Run number: 1
Number of elements in array: 71
Unsorted array:
8 19 9 6 17 11 19 7 5 14 6 6 16 14 16 8 11 16 17 12 17 19 14 9 19 18 17 11 12 12 19 8 7 18 13 8 16 16 11 20 8 15 18 14 13 10 20 18 9 16 17 11 10 10 16 8 18 14 15 17 16
19 7 5 15 5 19 5 13 11 18
Sorted array:
5 5 5 5 6 6 6 7 7 7 8 8 8 8 8 8 9 9 9 10 10 10 11 11 11 11 11 11 12 12 12 13 13 13 14 14 14 14 14 15 15 15 16 16 16 16 16 16 16 16 17 17 17 17 17 17 18 18 18 18 18 18 1
9 19 19 19 19 19 19 20 20
The number of comparisons are: 67
Run number: 2
Number of elements in array: 87
Unsorted array:
14 7 11 19 18 14 17 18 20 9 7 9 5 11 5 16 9 5 12 7 16 6 7 11 7 6 18 6 16 20 15 5 15 14 7 14 7 18 14 17 5 19 10 14 10 14 5 7 15 14 17 8 20 12 11 20 9 11 19 18 17 19 11 8
20 12 14 6 20 14 20 6 18 6 19 18 17 11 7 7 11 18 5 9 20 15 12
Sorted array:
5 5 5 5 5 5 5 6 6 6 6 6 6 7 7 7 7 7 7 7 7 7 7 8 8 9 9 9 9 9 10 10 11 11 11 11 11 11 11 12 12 12 12 14 14 14 14 14 14 14 14 14 14 15 15 15 15 16 16 16 17 17 17 17 17
18 18 18 18 18 18 18 18 19 19 19 19 19 20 20 20 20 20 20 20 20
The number of comparisons are: 81
Run number: 3
Number of elements in array: 45
Unsorted array:
12 13 14 17 8 16 14 7 19 8 9 19 17 9 10 9 20 9 9 14 19 10 20 7 7 9 17 20 8 5 18 13 8 20 10 11 14 18 15 10 5 12 16 13 6
Sorted array:
5 5 6 7 7 7 8 8 8 8 9 9 9 9 9 9 9 10 10 10 10 11 12 12 13 13 13 14 14 14 14 15 16 16 17 17 18 18 19 19 19 20 20 20 20
The number of comparisons are: 43
Run number: 4
Number of elements in array: 77
Unsorted array:
9 19 20 5 5 9 6 18 19 16 10 10 12 16 8 7 20 13 5 7 15 9 9 19 6 10 19 14 10 13 11 15 9 11 8 19 14 11 20 12 13 9 12 15 15 5 12 13 11 15 11 16 6 18 19 9 13 14 19 7 16 10 8


The number of comparisons are: 60
Run number: 8
Number of elements in array: 30
Unsorted array:
15 16 13 8 16 11 17 10 15 19 18 20 14 15 6 16 18 13 20 13 11 7 16 9 14 18 8 18 5 9
Sorted array:
5 6 7 8 8 9 9 10 11 11 13 13 13 14 14 15 15 15 16 16 16 17 18 18 18 18 19 20 20
The number of comparisons are: 29
Run number: 9
Number of elements in array: 55
Unsorted array:
9 11 11 8 8 16 16 6 5 19 17 17 8 17 8 11 10 9 6 19 15 18 20 10 5 15 12 20 13 18 9 18 9 14 12 14 7 17 17 6 13 18 17 12 5 8 9 13 11 12 10 17 5 15 18
Sorted array:
5 5 5 5 6 6 6 7 8 8 8 8 8 9 9 9 9 9 9 10 10 10 11 11 11 11 12 12 12 12 13 13 13 14 14 15 15 15 16 16 17 17 17 17 17 17 17 18 18 18 18 18 19 19 20 20
The number of comparisons are: 51
Run number: 10
Number of elements in array: 36
Unsorted array:
18 14 12 18 6 5 19 6 5 18 13 6 14 12 6 10 10 20 9 13 20 10 7 5 11 19 17 6 6 9 8 20 8 20 8 8
Sorted array:
5 5 5 6 6 6 6 6 6 7 8 8 8 8 9 9 10 10 10 11 12 12 13 13 14 14 17 18 18 18 19 19 20 20 20 20
The number of comparisons are: 33
Run number: 11
Number of elements in array: 87
Unsorted array:
8 15 20 19 15 20 10 20 8 6 10 19 15 19 10 12 15 11 12 18 6 19 10 11 9 14 5 14 7 9 14 17 18 6 20 12 11 11 16 18 13 11 13 16 13 6 13 6 16 13 7 7 14 9 5 12 6 12 13 13 18 7
14 18 11 14 14 7 16 9 17 15 20 12 13 10 5 10 12 10 20 5 17 18 18 9 14
Sorted array:


C:\Users\AJIT KUMAR\Documents\mergesort.exe
Unsorted array:
13 15 9 14 5 11 6 12 6 20 19 12 14 20 19 8 17 10 16 9 16 5 7 15 13 15 20 19 14 8 11 11 16 5 10 14 13 9 11 9 17 18 14 17 10
Sorted array:
5 5 5 6 6 7 8 8 9 9 9 9 10 10 10 11 11 11 12 12 13 13 13 14 14 14 14 15 15 15 16 16 16 17 17 17 18 19 19 19 20 20 20
The number of comparisons are: 44
Run number: 100
Number of elements in array: 44
Unsorted array:
6 9 14 17 18 6 18 7 15 17 8 14 12 13 8 16 15 10 18 13 6 10 18 16 17 18 6 20 14 19 12 6 20 9 15 12 20 19 20 6 12 20 16 20
Sorted array:
6 6 6 6 6 6 7 8 8 9 9 10 10 12 12 12 12 13 13 14 14 14 15 15 15 16 16 16 17 17 17 18 18 18 18 18 19 19 20 20 20 20 20 20
The number of comparisons are: 34
All sizes:
71 87 45 77 91 31 62 30 55 36 87 62 95 93 46 36 87 49 97 89 35 34 48 32 99 98 87 75 71 70 90 89 96 73 62 82 91 78 74 70 72 94 64 88 74 75 93 100 42 39 52 70 79 73 88 76
59 44 35 74 48 52 72 69 67 53 74 74 64 48 44 59 45 99 71 59 78 74 94 83 41 39 71 82 86 57 43 59 45 32 70 47 44 88 38 58 95 91 45 44
All no of comparisons:
67 81 43 74 89 29 60 29 51 33 85 59 93 92 44 33 84 47 90 87 33 30 45 30 93 96 82 74 69 67 89 86 93 69 60 80 88 76 72 65 69 91 63 85 71 74 89 96 40 34 51 69 75 66 87 75
57 42 34 73 47 49 71 67 66 52 72 72 63 46 43 58 43 95 70 57 76 72 90 78 40 38 67 80 80 55 42 57 43 30 68 44 42 87 34 54 93 88 44 34
Process exited after 2.114 seconds with return value 0
Press any key to continue . . .

//graph



Q4. Implement Radix Sort

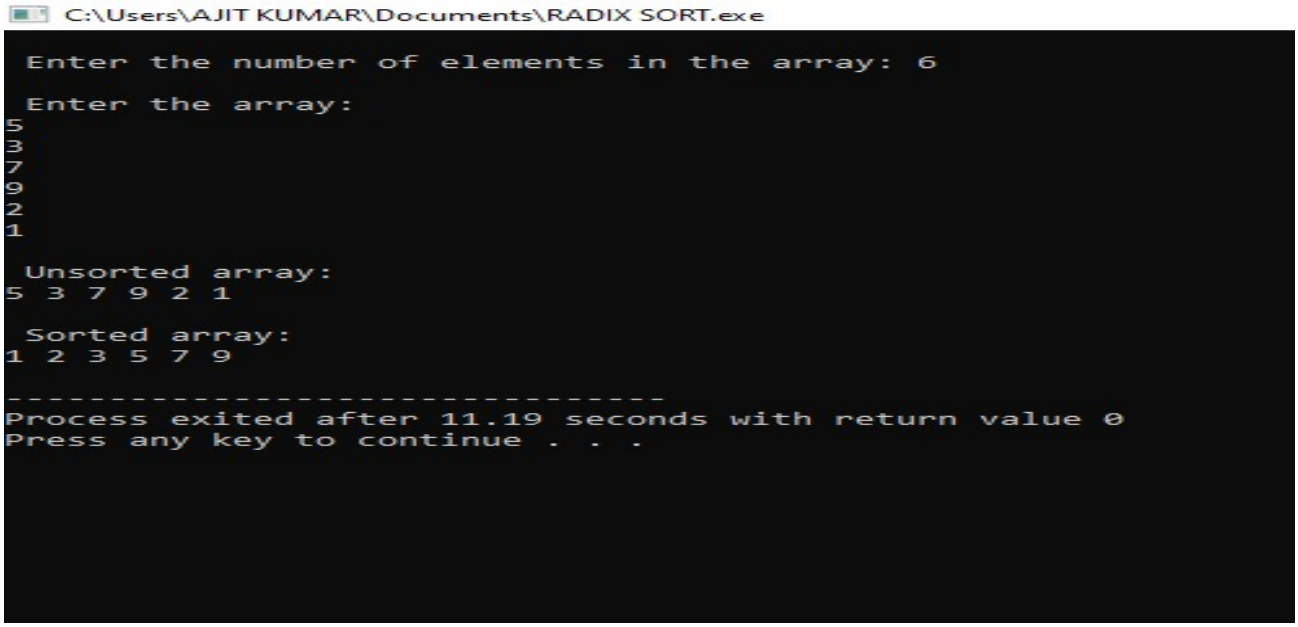
//CODE

```
#include <iostream>

using namespace std;

void Counting_Sort(int A[], int n, int d, int k)

{

int B[n], C[k];

for(int i = 0; i <= k; i++)

C[i] = 0;

for(int i = 0; i < n; i++)

C[(A[i] / d) % 10]++;

for(int i = 1; i <= k; i++)

C[i] += C[i-1];

for(int j = n; j >= 1; j--)

{

B[C[A[j / d] % 10]] = A[j];

C[A[j / d] % 10] -= 1;
```

```
}
cout<<"\n Sorted array:"<<endl;
for(int i = 1; i <= n; i++)
cout<<B[i]<<" ";

cout<<endl;
}
void RadixSort(int A[], int n)
{
int k = 0;
for(int i = 1; i <= n; i++)
{
if(A[i] > k)
k = A[i];
}
for(int d = 1; k/d > 0; d *= 10)
Counting_Sort(A, n, d, k);
}
int main()
{
int n, k = 0;
cout<<"\n Enter the number of elements in the array: ";
cin>>n;
int A[n];
cout<<"\n Enter the array:"<<endl;
for(int i = 1; i <= n; i++)

{
cin>>A[i];
if(A[i] > k)
k = A[i];
```

```
}
cout<<"\n Unsorted array:"<<endl;
for(int i = 1; i <= n; i++)
cout<<A[i]<<" ";
cout<<endl;
RadixSort(A, n);
return 0;
}
```

//OUTPUT



Q5. Implement Bucket Sort

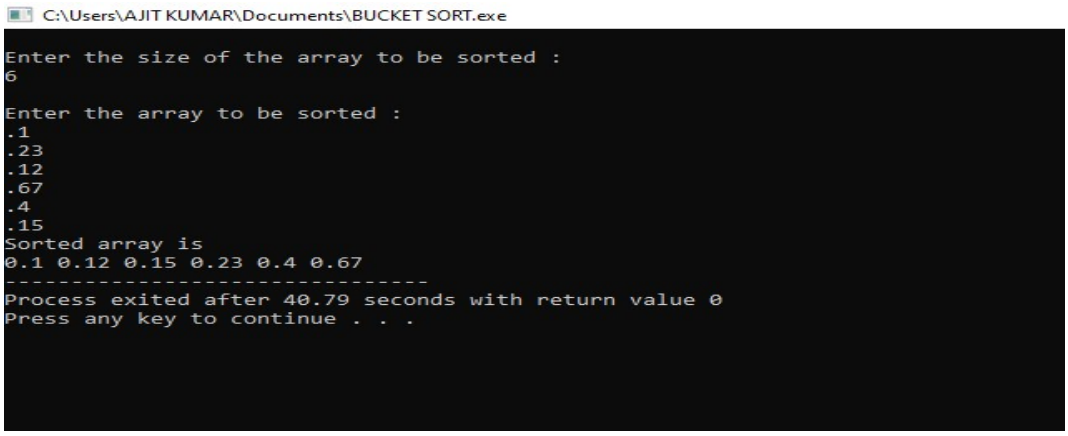
//CODE

```
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;

// Function to sort arr[] of
```

```
// size n using bucket sort
void bucketSort(float arr[], int n)
{

	// 1) Create n empty buckets
	vector<float> b[n];

	// 2) Put array elements
	// in different buckets
	for (int i = 0; i < n; i++) {
		int bi = n * arr[i]; // Index in bucket
		b[bi].push_back(arr[i]);
	}

	// 3) Sort individual buckets
	for (int i = 0; i < n; i++)
		sort(b[i].begin(), b[i].end());

	// 4) Concatenate all buckets into arr[]
	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()
```

```
{
	int size ;
	cout<<"\nEnter the size of the array to be sorted : \n";
	cin>>size;



	float arr[size];
	cout<<"\nEnter the array to be sorted : \n";
	for(int i=0;i<size;i++)
	{
		cin>>arr[i];
	}

	bucketSort(arr, size);

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

//OUTPUT



Q6. Implement Randomized select.

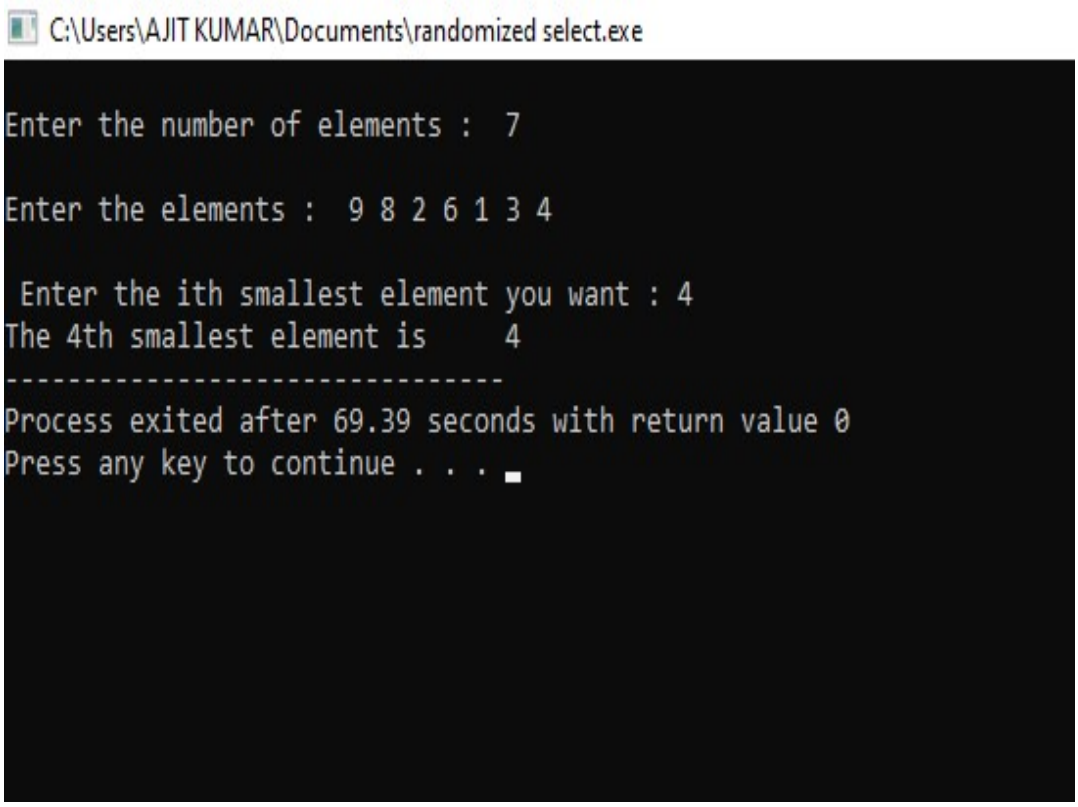
```
#include<iostream>
#include<stdlib.h>
using namespace std;

int randomno(int lower,int upper)
{

	int num=(rand()%(upper-lower+1))+lower;
	return num;

}

int PARTITION(int A[],int p,int r)
{
	int x,temp;
	x=A[r];
	int i=p-1;

	for(int j=p;j<=r-1;j++)
	{
		if(A[j]<=x)
		{

			i=i+1;
			temp=A[i];
			A[i]=A[j];
			A[j]=temp;
```

```
                }
        }

        temp=A[i+1];
        A[i+1]=A[r];
        A[r]=temp;


        return (i+1);
}


int randomizedpartition(int A[],int p,int r)
{
         int T;

        int i=randomno(p,r);
        T=A[r];
        A[r]=A[i];
        A[i]=T;

        return PARTITION(A,p,r);
}


int randomizedselect(int A[],int p,int r,int i)
{
        if(p==r)
        {
                return A[p];
```

```
        }

        int q,k;
        q=randomizedpartition(A,p,r);
        k=q-p+1;

        if(i==k)
        {
                return A[q];
        }
        else if(i<k)
        {
                return randomizedselect(A,p,q-1,i);
        }

        else
        {
                return randomizedselect(A,q+1,r,i-k);
        }
}
int main()
{
        int n, i;
        int small;
        int smallele;

        cout<<"\nEnter the number of elements :  ";
        cin>>n;
        int A[n];
```

```
        cout<<"\nEnter the elements :  ";
        for(i = 1; i <= n; i++)
        {
        cin>>A[i];
        }

        cout<<"\n Enter the ith smallest element you want : ";
        cin>>small;

        smallele=randomizedselect(A,1,n,small);

        cout<<"The "<<small<<"th smallest element is\t"<<smallele;
        return 0;
}
```

//output



Q7. Implement Breadth-First Search in a graph .

//CODE

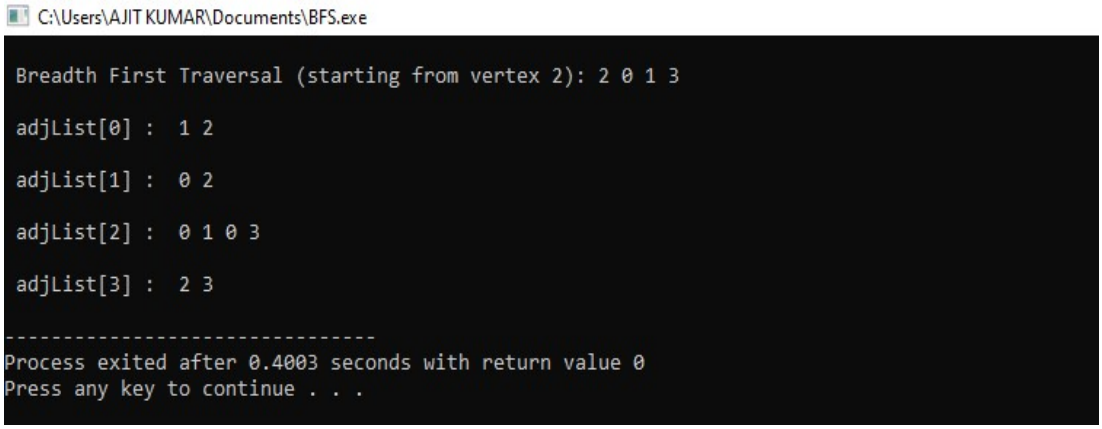
```
#include<iostream>
#include <list>
using namespace std;
class Graph
{
int V; // No. of vertices
list<int> *adjLists; // Pointer to an array containing adjacency lists.
public:
Graph(int V);
void addEdge(int src, int dest);
void BFS(int s);
list<int> getNodes(int v);
void showlist(list <int> g);
};
// Create a graph with given vertices, and maintain an adjacency list.
Graph::Graph(int V)
{
this->V = V;

adjLists = new list<int>[V]; // Creating 'V' no. of lists.
}
// Function to add an edge to graph.
```

```
void Graph::addEdge(int src, int dest)
{
adjLists[src].push_back(dest);
adjLists[dest].push_back(src); // The graph is undirected.
}
// Prints BFS traversal from a given source s.
void Graph::BFS(int s)
{
// Mark all the vertices as not Discovered.
bool *Discovered = new bool[V];
for(int i = 0; i < V; i++)
Discovered[i] = false;
// Create a queue for BFS.
list<int> queue;
// Mark the current node as Discovered and enqueue it.
Discovered[s] = true;

queue.push_back(s);
// 'i' will be used to get all adjacent vertices of a vertex.
list<int> :: iterator i;
while(!queue.empty())
{
// Dequeue a vertex from queue and print it.
s = queue.front();
cout << s << " ";
queue.pop_front();
```

```
// Get all adjacent vertices of the dequeued vertex s.
// If a adjacent has not been Discovered, then mark it Discovered and enqueue it.
for (i = adjLists[s].begin(); i != adjLists[s].end(); ++i)
{
if (!Discovered[*i])
{
Discovered[*i] = true;
queue.push_back(*i);
}
}
}
}
list<int> Graph::getNodes(int v)
{

return(adjLists[v]);
}
void Graph::showlist(list <int> l)
{
list <int> :: iterator it;
l.unique();
for(it = l.begin(); it != l.end(); it++)
cout<<" "<<*it;
cout<<endl;
}
int main()
```

```
{
Graph g(4);
g.addEdge(0, 1);
g.addEdge(0, 2);
g.addEdge(1, 2);
g.addEdge(2, 0);
g.addEdge(2, 3);
g.addEdge(3, 3);
cout << "\n Breadth First Traversal " << "(starting from vertex 2): ";
g.BFS(2);
cout<<endl;
for(int i = 0; i < 4; i++)

{
cout<<"\n adjList["<<i<<"] : ";
g.showlist(g.getNodes(i));
}
return 0;
}
```

//OUTPUT



Q8. Implement Depth-First Search in a graph .

//CODE

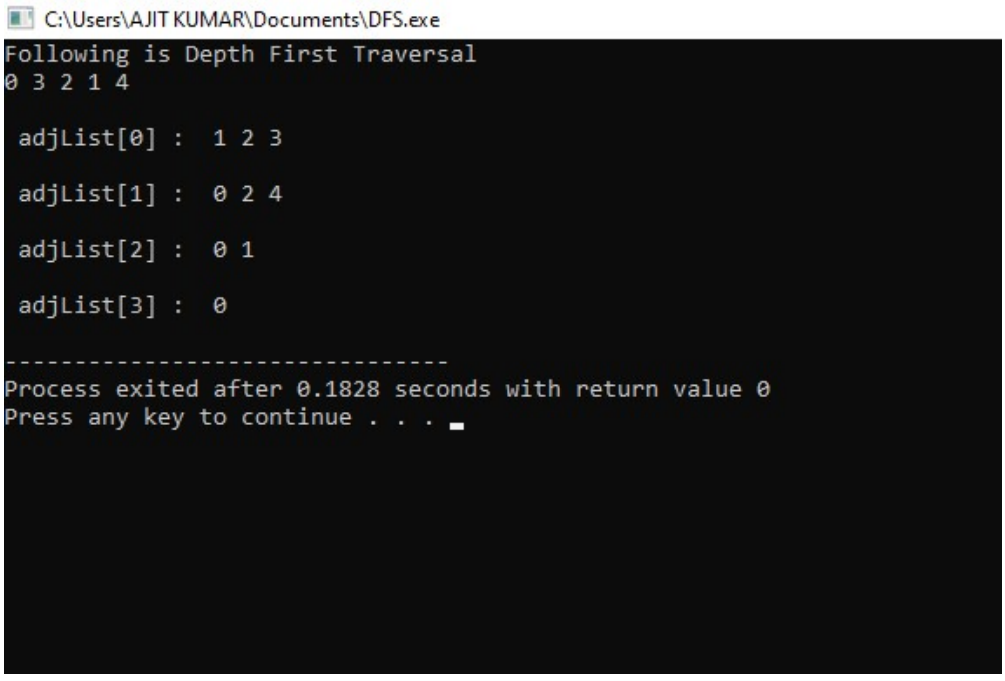
```
#include<bits/stdc++.h>
using namespace std;
class Graph
{
int V; // No. of vertices
list<int> *adjLists; // adjacency lists
public:
Graph(int V);
void addEdge(int src, int dest);
void DFS(int s);
list<int> getNodes(int v);
void showlist(list <int> g);
};
// Create a graph with given vertices, and maintain an adjacency list.
Graph::Graph(int V)
{
this->V = V;
adjLists = new list<int>[V];

}
// Function to add an edge to graph.
void Graph::addEdge(int src, int dest)
```

```
{
adjLists[src].push_back(dest);
adjLists[dest].push_back(src); // The graph is undirected.
}
// Prints DFS traversal from a given source s.
void Graph::DFS(int s)
{
// Initially mark all verices as not visited
vector<bool> visited(V, false);
// Create a stack for DFS
stack<int> stack;
// Push the current source node.
stack.push(s);
// 'i' will be used to get all adjacent vertices of a vertex.
list<int> :: iterator i;

while (!stack.empty())
{
// Pop a vertex from stack and print it
s = stack.top();
stack.pop();
// Stack may contain same vertex twice.
// So we need to print the popped item only if it is not visited.
if (!visited[s])
{
cout << s << " ";
```

```
visited[s] = true;
}
// Get all adjacent vertices of the popped vertex s
// If a adjacent has not been visited, then push it to the stack.
for (i = adjLists[s].begin(); i != adjLists[s].end(); ++i)
if (!visited[*i])
stack.push(*i);
}
}
list<int> Graph::getNodes(int v)
{
return(adjLists[v]);
}

void Graph::showlist(list <int> l)
{
list <int> :: iterator it;
l.unique();
for(it = l.begin(); it != l.end(); it++)
cout<<" "<<*it;
cout<<endl;
}
int main()
{
Graph g(5);
g.addEdge(1, 0);
```

```
g.addEdge(0, 2);
g.addEdge(2, 1);
g.addEdge(0, 3);
g.addEdge(1, 4);
cout << "Following is Depth First Traversal\n";
g.DFS(0);
cout<<endl;
for(int i = 0; i < 4; i++)
{
cout<<"\n adjList["<<i<<"] : ";
g.showlist(g.getNodes(i));
}
return 0;
}
```

//OUTPUT



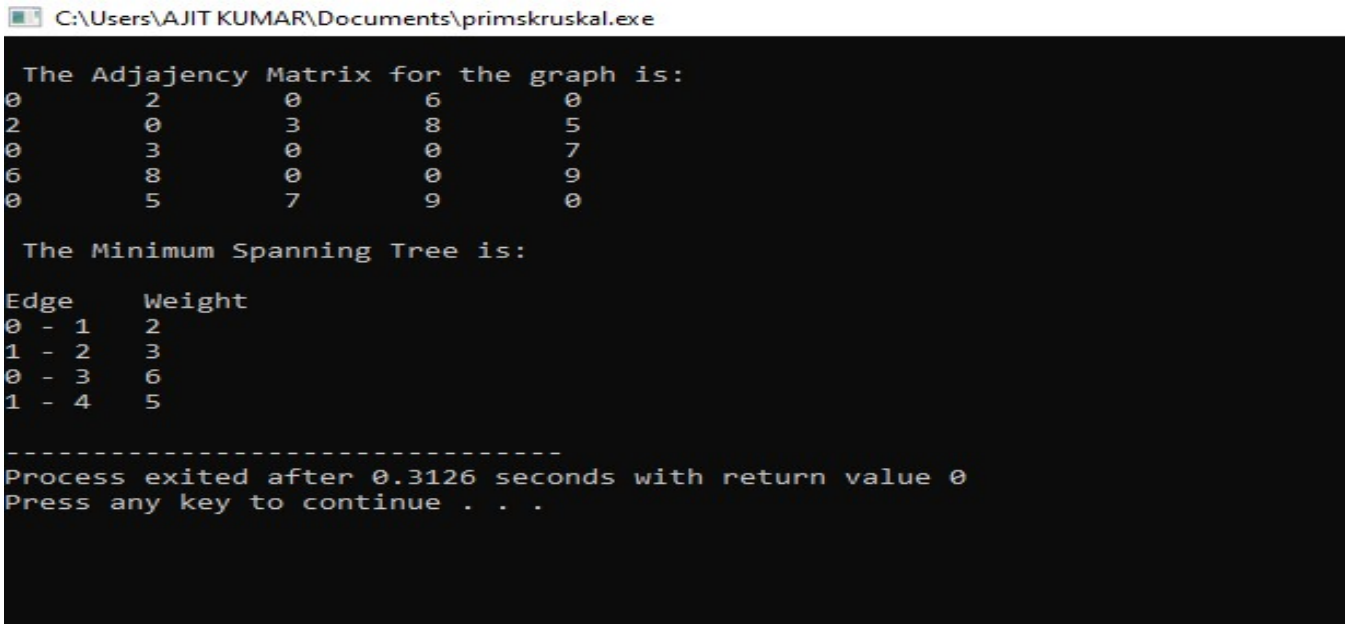
Question – 9 Write a program to determine the minimum spanning tree of a graph using both Prims and Kruskals algorithm.

```
#include<iostream>
using namespace std;
#define V 5
int minKey(int key[], bool mstSet[])
{
int min=INT_MAX, min_index;
for (int v = 0; v < V; v++)
if (mstSet[v] == false && key[v] < min)
min = key[v], min_index = v;
return min_index;
}
void printMST(int parent[], int graph[V][V])
{
cout<<"Edge \tWeight\n";
for (int i = 1; i < V; i++)
cout<<parent[i]<<" - "<<i<<" \t"<<graph[i][parent[i]]<<" \n";

}
void primMST(int graph[V][V])
{
int parent[V];
int key[V];
bool mstSet[V];
for (int i = 0; i < V; i++)
key[i] = INT_MAX, mstSet[i] = false;
key[0] = 0;
parent[0] = -1;
```

```
for (int count = 0; count < V - 1; count++)
{
int u = minKey(key, mstSet);
mstSet[u] = true;
for (int v = 0; v < V; v++)
if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v])
parent[v] = u, key[v] = graph[u][v];
}
printMST(parent, graph);

}
int main()
{
int graph[V][V] = { { 0, 2, 0, 6, 0 },
{ 2, 0, 3, 8, 5 },
{ 0, 3, 0, 0, 7 },
{ 6, 8, 0, 0, 9 },
{ 0, 5, 7, 9, 0 }};
cout<<"\n The Adjajency Matrix for the graph is:"<<endl;
for(int i=0;i<V;i++)
{
for(int j=0;j<V;j++)
{
cout<<graph[i][j]<<"\t";
}
cout<<"\n";
}
cout<<"\n The Minimum Spanning Tree is:"<<endl<<endl;
primMST(graph);
return 0;
```

}

//output



Question – 10 Write a program to solve the weighted interval scheduling problem .

```
#include<iostream>

#include<algorithm>

using namespace std;


int computepj(int i,int S[],int FT[])

{

   int c=0;


        for(int j=1;j<i;j++)

        {


                if(FT[j]<=S[i])

                {

                  c= j ;
```

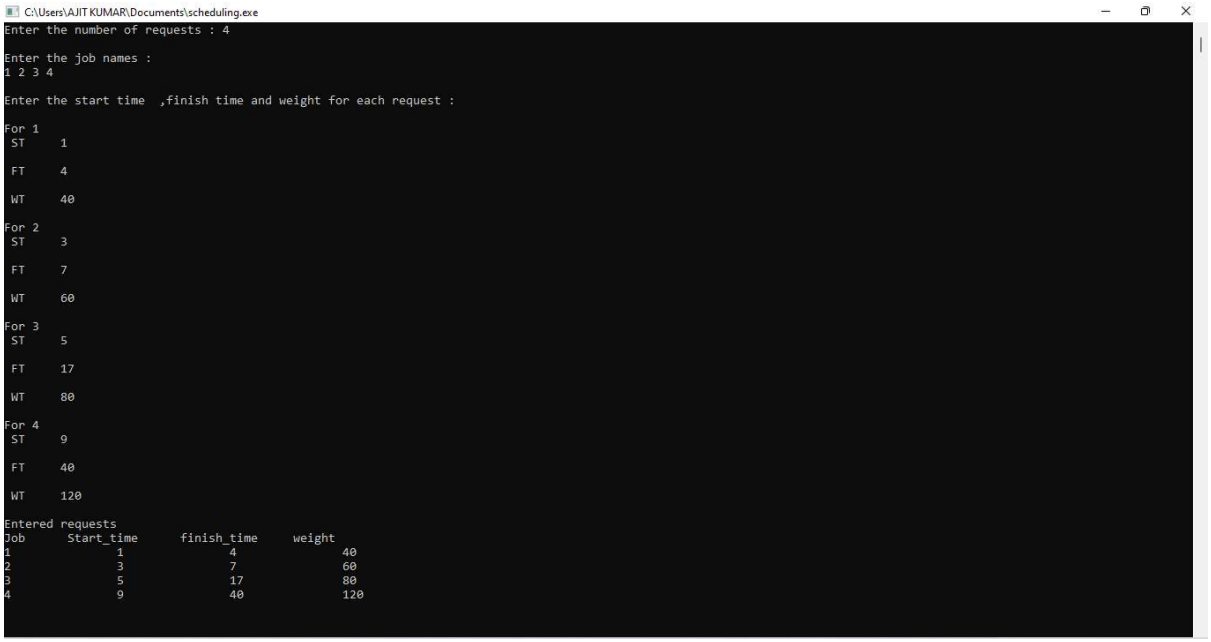
```
                }

        }

         return c;
}


int Computeopt(int j,int S[],int FT[],int Wt[])
{
        if(j==0)
        return 0;

        else
         return
                max(Wt[j]+Computeopt(computepj(j,S,FT),S,FT,Wt),
                Computeopt(j-1,S,FT,Wt));
}


int main()
{

int num;
cout<<"Enter the number of requests : ";
cin>>num;

int J[num];    // job no.
int S[num];   // start time
```

```
int FT[num];    // finish time
int Wt[num];     // weight
int t;
cout<<"\nEnter the job names : \n";

for(int i=1;i<=num;i++)
{
cin>>J[i];
}


cout<<"\nEnter the start time  ,finish time and weight for each request : \n";
for(int i=1;i<=num;i++)
{
cout<<"\nFor "<<i<<"\n ST\t";
cin>>S[i];

cout<<"\n FT\t";
cin>>FT[i];
cout<<"\n WT\t";
cin>>Wt[i];
}
cout<<"\nEntered requests\n";
cout<<"Job\t Start_time\t finish_time\t weight\n";
for(int i=1;i<=num;i++)
{
cout<<J[i]<<"\t\t"<<S[i]<<"\t\t"<<FT[i]<<"\t\t"<<Wt[i]<<"\t\t\n";
}
```

```
bool flag = true ;
//to sort the jobs in the order of increasing finish time
for(int i=1;i<=num;i++)
{
for(int j=1;j<=num-i;j++)
{

 if(FT[j]>FT[j+1])
 {
          flag = false ;
 t=FT[j];
 FT[j]=FT[j+1];
 FT[j+1]=t;

 t=S[j];
 S[j]=S[j+1];
 S[j+1]=t;

 t=J[j];
 J[j]=J[j+1];
 J[j+1]=t;

 t=Wt[j];
 Wt[j]=Wt[j+1];
 Wt[j+1]=t;
 }


}
```

```
}

int pj;

if(flag == false)        // will execute only if the we have sorted the data according to finish time
{

        cout<<"After sorting\n";
        cout<<"Job\t Start_time\t finish_time\t weight\n";
        for(int i=1;i<=num;i++)
        {
        cout<<J[i]<<"\t\t"<<S[i]<<"\t\t"<<FT[i]<<"\t\t"<<Wt[i]<<"\t\t\n";
        }
}

cout<<"\n\n";
for(int i=1;i<=num;i++)
{
        cout<<"p("<<i<<")\t";
        pj=computepj(i,S,FT);
        cout<<pj<<"\n";
}

cout<<"\nOptimal value "<<Computeopt(num,S,FT,Wt);

return 0 ;
}
```

//output

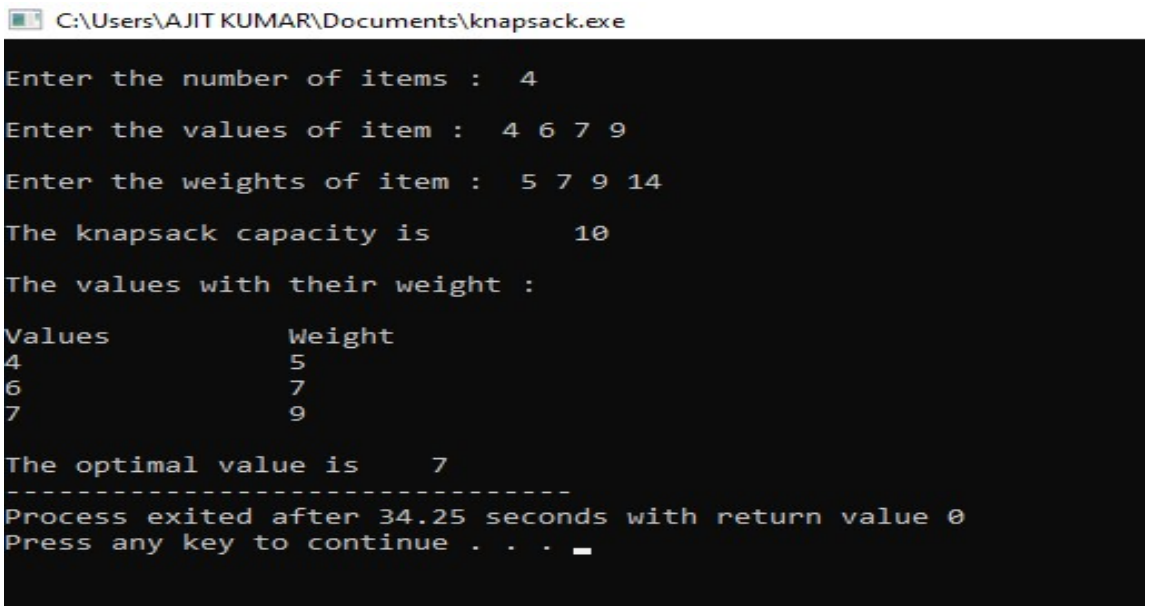


Question – 11 Write a program to solve the 0-1 knapsack problem.

```
#include<iostream>

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)   // base case
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 n ;

        int W = 10;
                cout<<"\nEnter the number of items :  ";
                cin>>n ;
                int val[n], wt[n];
        cout<<"\nEnter the values of item :  ";
        for(int i =0 ; i<n ;i++)
        {
                cin>>val[i];
        }
```

```
        cout<<"\nEnter the weights of item :  ";
        for(int i =0 ; i<n ;i++)
        {
                cin>>wt[i];
        }



cout<<"\nThe knapsack capacity is \t"<<W;
cout<<"\n\nThe values with their weight :\n";
cout<<"\nValues\t\tWeight\t\n";
for(int i=0;i<3;i++)
{
cout<<val[i]<<"\t\t"<<wt[i]<<"\n";
}

cout<<"\nThe optimal value is \t"<<knapSack(W, wt, val, n);
return 0;
}
//output
```


C:\Users\AJIT KUMAR\Documents\knapsack.exe
Enter the number of items : 4
Enter the values of item : 4 6 7 9
Enter the weights of item : 5 7 9 14
The knapsack capacity is 10
The values with their weight :
Values Weight
4 5
6 7
7 9
The optimal value is 7
Process exited after 34.25 seconds with return value 0
Press any key to continue . . .