Lab Work

University of Baltistan Skardu

Submitted to:

Ma’am Noreen Maryam

Submitted By:

M Nawaz

Registration No:

S23BSCS014

Department:

Computer Science

Date:

May 27,2024

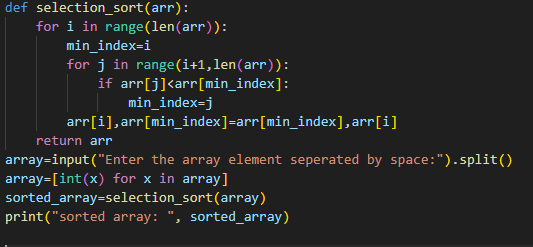
Section:

“A”

LAB#20

Q: Write a program that takes an unsorted array from the user and then sorts it by using selection sort. Explain each line of code in detail when you execute it.

Program:



Output:



Explanation:

Def- selection -sort ([3,6,1,8,2,4]):

n=len ([3,6,1,8,2,4])

for i in range(6): #i=0

minindex=0

for j in range(i+1,n):

if a[1]<a[0]:

6<3

X

for j in range(2,6):

if a[2]<a[0]

1<3

Minindex=2

for j in range(4,6):

if a[4]<a[2]

2<1

X

for j in range(5,6):

if a[5]<a[2]

4<1

X

a[0],a[2]=a[2],a[0]

a[0],a[2]=1,3

updated array:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 |
| 1 | 6 | 3 | 8 | 2 | 4 |

for i in range(6): #i=1

minindex=1

for j in range(1+1,n)

if a[2]<a[1]:

3<6

Minindex=2

for j in range(3,6):

if a[3]<a[2]

8<3

X

for j in range(4,6):

if a[4]<a[2]

2<3

True

Minindex=j

Minindex=4

for j in range(5,6):

if a[5]<a[4]

4<2

X

a[1],a[4]=a[4],a[1]

a[1],a[4]=2,6

updated array:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 |
| 1 | 2 | 3 | 8 | 6 | 4 |

for i in range(6): #i=3

minindex=3

for j in range(4,6):

if a[4]<a[3]:

6<8

minindex=4

for j in range(5,6):

if a[5]<a[4]:

4<6

Minindex=5

a[3],a[5]=a[5],a[3]

a[3],a[5]=4,8

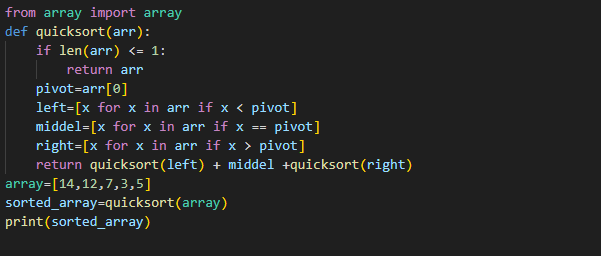
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 |
| 1 | 2 | 3 | 4 | 6 | 8 |

Thus the array is completely sorted in ascending order by the method of selection sort.

LAB#21

**Q.1:** Write a program to sort an unsorted array by using the quick sort algorithm consider first element as a pivot element.

Program:



Output:



Explanation:

**1. def quick\_sort(arr):**

**If len([14,12,7,3,5])<=1:**

**Return arr**

**Pivot=arr[0]**

**Pivot=14**

**Left= [12,7,3,5]**

**Middle= [14]**

**I return quick\_sort([12,7,3,5])+middle**

**1-left🡪quick\_sort([12,7,3,5])**

**Middle🡪14**

**quick\_sort([12,7,3,5])**

**1-left:**

**def quick\_sort(arr):**

**if len[12,7,3,5]<=1**

**X**

**Pivot=arr[0]**

**Left=7,3,5**

**Middle=12**

**Right=[]**

**II return quick\_sort[7,3,5]+middle+quick\_sort[]**

**1-left-left🡪quick\_sort[7,3,5]**

**1-left-middle🡪5**

**1-left-right🡪empty**

**1-left-left:**

**def quick\_sort(arr):**

**if len[7,3,5]<=1:**

**X**

**Pivot=arr[0]**

**Left=3,5**

**Middle=7**

**Right=[]**

**1-left-left-left=[3,5]**

**Midde=7**

**return quick\_sort[3,5]+middle+quick\_sort[]**

**1-left-left-left=[3,5]**

**def quick\_sort(arr):**

**if len[3,5]<=1:**

**X**

**Pivot=arr[0]**

**Left=[]**

**Middle=3**

**Right=[5]**

**1-left-left-left-right=5**

**return quick\_sort[]+middle+quick\_sort[5]**

**1-left-left-left-right=[5]**

**def quick\_sort(arr):**

**if len[5]<=1:**

**return 5**

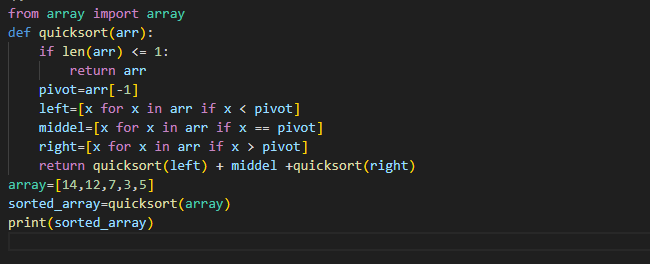
**I return quick\_sort([12,7,3,5])+14+quick\_sort()**

**return[3,5,7,12,14]**

**sorted\_a1=[3,5,7,12,14]**

**Q.2:** Write a program to sort an unsorted array by using the quick sort algorithm consider last element as a pivot element.

Program:



Output:



Explanation:

**1. def quick\_sort(arr):**

**If len([14,12,7,3,5])<=1:**

**Return arr**

**Pivot=arr[-1]**

**Pivot=5**

**Left=[3]**

**Middle=[5]**

**Right=[14,12,7]**

**I return quick\_sort([3])+middle+quick\_sort([14,12,7])**

**1-left🡪quick\_sort([3])**

**Middle🡪5**

**1-right🡪quick\_sort([14,12,7]**

**1-left:**

**def quick\_sort(arr):**

**if len[3]<=1:**

**return 3**

**Now:**

**I return quick\_sort[3]+middle+quick\_sort[]**

**quick\_sort([14,12,7])**

**1-right:**

**def quick\_sort(arr):**

**if len[14,12,7]<=1**

**X**

**Pivot=arr[-1]**

**Left=[]**

**Middle=7**

**Right=14,12**

**II return quick\_sort[]+7+quick\_sort[14,12]**

**1-right-left🡪empty**

**1-right-middle🡪7**

**1-right-right🡪14,12**

**1-right-right:**

**def quick\_sort(arr):**

**if len[14,12]<=1:**

**X**

**Pivot=arr[-1]**

**Left=[]**

**Middle=12**

**Right=14**

**1-right-right-right=14**

**1-right-right-middel=12**

**1-right-right-right:**

**def quick\_sort(arr):**

**if len[14]<=1:**

**return 14**

**now:**

**III return quick\_sort[]+7+quick\_sort[14,12]**

**Return [14,12]**

**We have:**

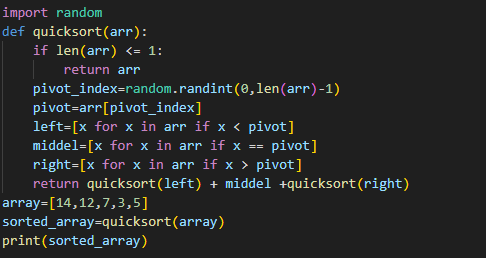
**1-right=[14,12]**

**I return quick\_sort([3])+middle+quick\_sort([7,12,14])**

**return[3,5,7,12,14]**

**sorted\_a1=[3,5,7,12,14]**

**Q.3:** Write a program to sort an unsorted array by using the quick sort algorithm consider any random element as a pivot element.



Output:



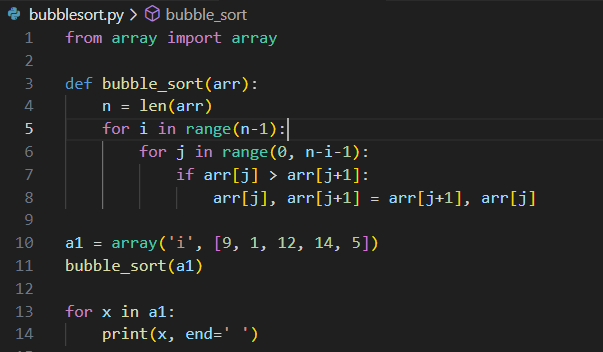
Explanation:

This program is written for random value of pivot. The value of pivot is randomly chosen but there is no difference on the output of the program.

LAB#22

Q: Explain the code (LAB#22 Example#1) in detail.

Program:



Output:



Explanation of code:

Array= [9,1,12,14,5]

N=len(arr) n=5

I=4 j= (0,3)

J=0

If arr[j] > arr[j+1]

arr[0] > arr[1]

9>1

arr[j],arr[j+1]=arr[j+1],arr[j]

arr[0],arr[1]=arr[1],arr[0]

arr[0],arr[1]=1,9

updated array

1,9,12,14,5

J=1

If arr[j] > arr[j+1]

arr[1] > arr[2]

9>12

X

J=2

If arr[j] > arr[j+1]

arr[2] > arr[3]

12>14

X

J=3

If arr[j] > arr[j+1]

arr[3] > arr[4]

14>5

arr[j],arr[j+1]=arr[j+1],arr[j]

arr[3],arr[4]=arr[4],arr[3]

arr[3],arr[4]=5,14

updated array

1,9,12,5,14

I=3 j= (0,2)

J=0

If arr[j] > arr[j+1]

arr[0] > arr[1]

1>9

X

J=1

If arr[j] > arr[j+1]

arr[1] > arr[2]

9>12

X

J=2

If arr[j] > arr[j+1]

arr[2] > arr[3]

12>5

arr[j],arr[j+1]=arr[j+1],arr[j]

arr[2],arr[3]=arr[3],arr[2]

arr[3],arr[4]=5,12

updated array

1,9,5,12,14

I=2 j= (0,1)

J=0

If arr[j] > arr[j+1]

arr[0] > arr[1]

1>9

X

J=1

If arr[j] > arr[j+1]

arr[1] > arr[2]

9>5

arr[j],arr[j+1]=arr[j+1],arr[j]

arr[1],arr[2]=arr[2],arr[1]

arr[1],arr[2]=5,9

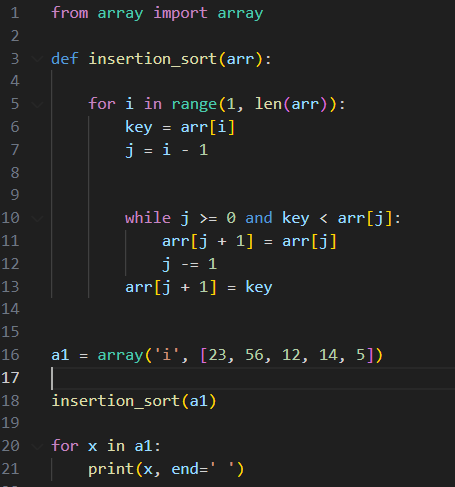
updated array

1,5,9,12,14

Thus the array is sorted in ascending order.

LAB#24

Q: Write a complete explanation of the above code (Example#1 lab#24).



Result:



Explanation of code:

arr=[23,56,12,14,5]

i=(1,5)

i=1

key=56

j=0

if 0>=0 and 56<23

X

arr[1]=56

updated array

23,56,12,14,5

i=2

key=12

j=1

if 1>=0 and 12<56

arr[2]=arr[1]

arr[2]=56

j=0

if 0>=0 and 12<23

arr[1]=arr[0]

arr[1]=23

j=-1

if -1>=0 and 12<23

X

arr[0]=12

updated array

12,23,56,14,5

i=3

key=14

j=2

if 2>=0 and 14<56

arr[3]=arr[2]

arr[3]=56

j=1

if 1>=0 and 14<23

arr[2]=arr[1]

arr[2]=23

j=0

if 0>=0 and 14<12

X

arr[1]=14

updated array

12,14,23,56,5

i=4

key=5

j=3

if 3>=0 and 5<56

arr[4]=arr[3]

arr[4]=56

j=2

if 2>=0 and 5<23

arr[3]=arr[2]

arr[3]=23

j=1

if 1>=0 and 5<14

arr[2]=arr[1]

arr[2]=14

j=0

if 0>=0 and 5<12

arr[1]=arr[0]

arr[1]=12

j=-1

if -1>=0 and 5<12

X

arr[0]=5

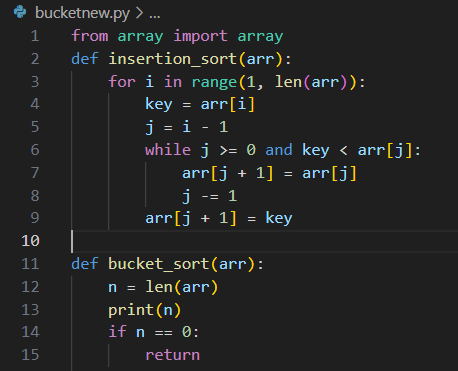
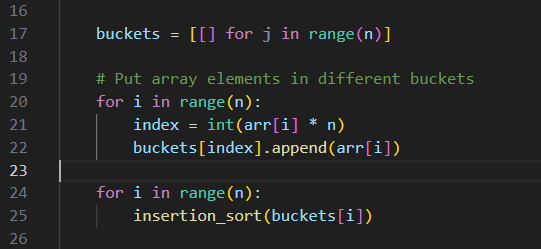
updated array

5,12,14,23,56

Thus the array is sorted in ascending order.

LAB#25

**Q.1:** Perform the remaining steps in example#1(LAB#25).

A computer screen shot of a program code

Description automatically generated

Explanation of the code:

a1=array(‘d’,[0.38,0.90,0.47,0.69,0.52,0.88,0.71,0.18])

bucket\_sort(a1)

def bucket\_sort(a1):

n=len(a1)

n=9

if n==0:

return

buckets=[[ ] for j in range(n)] #n=9



for i in range(9): #i=0

index=int(arr[i]\*n)

index=int(arr[0]\*9)

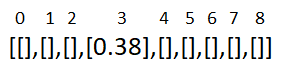
index=int(0.38\*9)

index=int(3.42)

index=3

buckets[index].append(arr[0])

buckets[3].append(0.38)



for i in range(9): #i=1

index=int(arr[i]\*n)

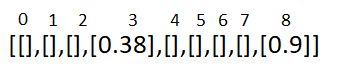
index=int(arr[1]\*9)

index=int(0.9\*9)

index=int(8.1)

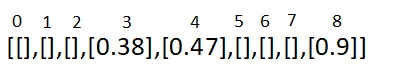
index=8

buckets[index].append(arr[1])

 buckets[8].append(0.9)

for i in range(9): #i=2

index=int(arr[i]\*n)

 index=int(arr[2]\*9)

index=int(0.47\*9)

index=int(4.23)

index=4

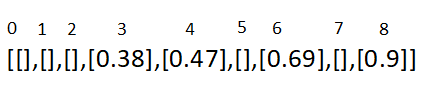
buckets[index].append(arr[2])

buckets[4].append(0.47)

for i in range(9): #i=3

index=int(arr[i]\*n)

index=int(arr[3]\*9)

 index=int(0.69\*9)

index=int(6.21)

index=6

buckets[index].append(arr[3])

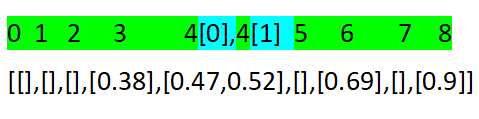
buckets[6].append(0.69)

for i in range(9): #i=4

index=int(arr[i]\*n)

index=int(arr[4]\*9)

index=int(0.52\*9)

 index=int(4.68)

index=4

buckets[index].append(arr[4])

buckets[4].append(0.52)

for i in range(9): #i=5

index=int(arr[i]\*n)

index=int(arr[5]\*9)

A close-up of numbers

Description automatically generated index=int(0.88\*9)

index=int(7.92)

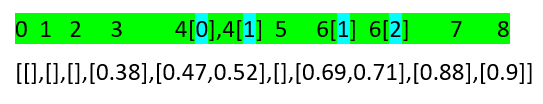
index=7

buckets[index].append(arr[5])

buckets[7].append(0.88)

for i in range(9): #i=6

index=int(arr[i]\*n)

 index=int(arr[6]\*9)

index=int(0.71\*9)

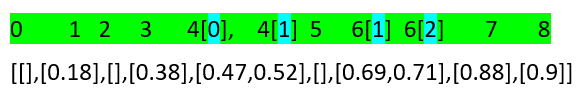
index=int(6.39)

index=6

buckets[index].append(arr[6])

buckets[6].append(0.71)

for i in range(9): #i=7

 index=int(arr[i]\*n)

index=int(arr[7]\*9)

index=int(0.18\*9)

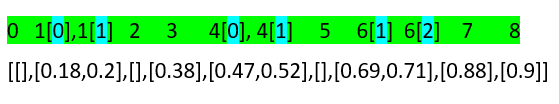
index=int(1.62)

index=1

buckets[index].append(arr[7])

buckets[1].append(0.18)

for i in range(9): #i=8

 index=int(arr[i]\*n)

index=int(arr[8]\*9)

index=int(0.2\*9)

index=int(1.8)

index=1

buckets[index].append(arr[8])

buckets[1].append(0.20)

for I in range (9)

insertion\_sort(buckets[i])

i=1

insertion\_sort(0.18,0.20)

i=(0,1)

i=0

key=0.18

j=-1

-1>=0 and 0.18<0

Arr[0]=0.18

I=1

Key=0.20

J=0

0>=0 and 0.20<0

Arr[1]=0.20

Follow this step on other array and the array is sorted in ascending order.

**Q.2:** Create an array of size 10. Instead of using elements from 0 to 1, use elements in the array greater than 1. Then apply the code from example #1 (LAB #25). If you encounter any errors, discuss the error and identify the reason behind it.

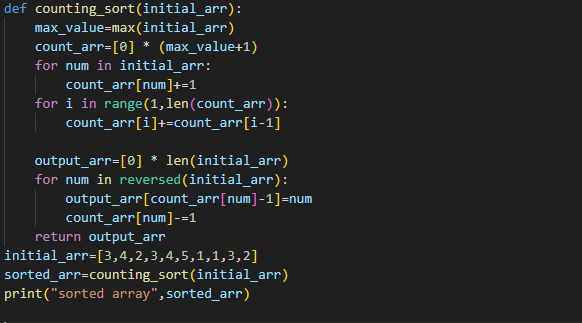
Program:

If we create an array of length 10 and the value in the array is greater than 1 then the output of this program is list index out of range. It means that the input value is not within the expected or valid range.

So list index out of range is an error message indicating that the input value is not within the valid range that the program can handle.

LAB#26

Q.1: Take an unsorted array, apply the counting sort algorithm to sort it, and then explain the code in detail, visualizing each step.

Program:  
 

Output:



Explanation of the code:

Initial\_arr=[3,4,2,3,4,5,1,1,3,2]

Sorted\_arr=counting\_sort(initial\_arr)

def counting\_sort([3,4,2,3,4,5,1,1,3,2]):

max\_val=max(initial\_arr)

max\_val=5

count\_arr=[0]\*(max\_val+1)

count\_arr=[0]\*6

count\_arr=[0,0,0,0,0,0]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 0 | 0 | 0 | 0 | 0 |
|  | 1 | 1 | 1 | 1 |  |
|  | 2 | 2 | 2 | 2 | 1 |
|  |  |  | 3 |  |  |
| 0 | 2 | 2 | 3 | 2 | 1 |

Count\_arr🡪

for num in initial\_arr:

Count\_arr[num]=Count\_arr[num]+1

Count\_arr[3]=Count\_arr[3]+1=0+1=1

Count\_arr[4]=Count\_arr[4]+1=0+1=1

Count\_arr[2]=Count\_arr[2]+1=0+1=1

Count\_arr[3]=Count\_arr[3]+1=1+1=2

Count\_arr[4]=Count\_arr[4]+1=1+1=2

Count\_arr[5]=Count\_arr[5]+1=0+1=1

Count\_arr[1]=Count\_arr[1]+1=0+1=1

Count\_arr[1]=Count\_arr[1]+1=1+1=2

Count\_arr[3]=Count\_arr[3]+1=2+1=3

Count\_arr[2]=Count\_arr[2]+1=1+1=2

Count\_arr=

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 2 | 2 | 3 | 2 | 1 |

Now, store the result of cumulative sum in the Count\_arr:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 2 | 2 | 3 | 2 | 1 |
| 0 | 2 | 4 | 7 | 9 | 10 |

Count\_arr 🡪

for i in range(1,len(Count\_arr)):

Count\_arr[i]=Count\_arr[i]+Count\_arr[i-1]

#i=1 Count\_arr[1]=Count\_arr[1]+Count\_arr[1-1]

= Count\_arr[1]+Count\_arr[0]

=2+0=2

#i=2 Count\_arr[2]=Count\_arr[2]+Count\_arr[2-1]

= Count\_arr[2]+Count\_arr[1]

=2+2=4

#i=3 Count\_arr[3]=Count\_arr[3]+Count\_arr[3-1]

= Count\_arr[3]+Count\_arr[2]

=3+4=7

#i=4 Count\_arr[4]=Count\_arr[4]+Count\_arr[4-1]

= Count\_arr[4]+Count\_arr[3]

=2+7=9

#i=5 Count\_arr[5]=Count\_arr[5]+Count\_arr[5-1]

= Count\_arr[5]+Count\_arr[4]

=9+1=10

Updated Count\_arr:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 2 | 4 | 7 | 9 | 10 |

Output=[0]\*len(initial\_arr)

=[0]\*10

Output=[0,0,0,0,0,0,0,0,0,0]

Now, we have:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 3 | 4 | 2 | 3 | 4 | 5 | 1 | 1 | 3 | 2 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 2 | 4 | 7 | 9 | 10 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Initial\_arr

Count\_arr

Output\_arr

for num in reversed(initial\_arr):

#num=2

Output\_arr[Count\_arr[2]-1]=2

Output\_arr[4-1]=2

Output\_arr[3]=2

Count\_arr[2]=Count\_arr[2]-1

=4-1

Count\_arr[2]=3

#num=3

Output\_arr[Count\_arr[ 3]-1]=3

Output\_arr[7-1]=3

Output\_arr[6]=3

Count\_arr[3]=Count\_arr[3]-1

=7-1

Count\_arr[3]=6

#num=1

Output\_arr[Count\_arr[1]-1]=1

Output\_arr[2-1]=1

Output\_arr[1]=1

Count\_arr[1]=Count\_arr[1]-1

=2-1

Count\_arr[1]=1

#num=1

Output\_arr[Count\_arr[1]-1]=1

Output\_arr[1-1]=1

Output\_arr[0]=1

Count\_arr[1]=Count\_arr[1]-1

=1-1

Count\_arr[1]=0

#num=5

Output\_arr[Count\_arr[5]-1]=5

Output\_arr[10-1]=5

Output\_arr[9]=5

Count\_arr[5]=Count\_arr[5]-1

=10-1

Count\_arr[5]=9

#num=4

Output\_arr[Count\_arr[4]-1]=4

Output\_arr[9-1]=4

Output\_arr[8]=4

Count\_arr[4]=Count\_arr[4]-1

=9-1

Count\_arr[4]=8

#num=3

Output\_arr[Count\_arr[3]-1]=3

Output\_arr[6-1]=3

Output\_arr[5]=3

Count\_arr[3]=Count\_arr[3]-1

=6-1

Count\_arr[3]=5

#num=2

Output\_arr[Count\_arr[2]-1]=2

Output\_arr[3-1]=2

Output\_arr[2]=2

Count\_arr[2]=Count\_arr[2]-1

=3-1

Count\_arr[1]=2

#num=4

Output\_arr[Count\_arr[4]-1]=4

Output\_arr[8-1]=4

Output\_arr[7]=4

Count\_arr[4]=Count\_arr[4]-1

=8-1

Count\_arr[4]=7

#num=3

Output\_arr[Count\_arr[3]-1]=3

Output\_arr[5-1]=3

Output\_arr[4]=3

Count\_arr[3]=Count\_arr[3]-1

=5-1

Count\_arr[2]=4

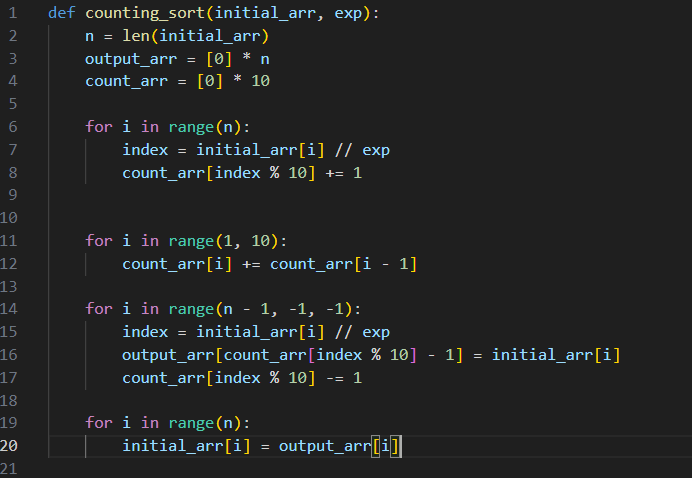
Return Output\_arr

[1,1,2,2,3,3,3,4,,5]

LAB#27

Q.1: Consider the below code (Lab#27 example#1)then explain the code in detail, visualizing each step.

Program:



A computer screen with text

Description automatically generated

Output:



Explanation:

Radix\_sort=[2050,1,14,100]

Def radix\_sort(2050,1,14,100)

Max\_value=2050

Exp=1

While 2050//1 >0:

Counting\_sort([2050,1,14,100],1)

N=4

Output\_arr=[0,0,0,0]

Count\_arr=[0,0,0,0,0,0,0,0,0,0]

For I in range(0,3)

I=0

Index=2050//1

Index=2050

Count\_arr[2050%10]+=1

Count\_arr[0]+=1

I=1

Index=1//1

Index=1

Count\_arr[0%10]+=1

Count\_arr[1]+=1

I=2

Index=14//1

Index=14

Count\_arr[14%10]+=1

Count\_arr[4]+=1

I=3

Index=100//1

Index=100

Count\_arr[100%10]+=1

Count\_arr[0]+=1

Updated count\_arr=[2,1,0,0,1,0,0,0,0,0]

For I in range(1,10)

Count\_arr[i]=count\_arr[i]+count\_arr[i-1]

Count\_arr[1]=1+2

Count\_arr[1]=3

Count\_arr[2]=0+3

Count\_arr[2]=3

Count\_arr[3]=0+3

Count\_arr[3]=3

Count\_arr[4]=1+3

Count\_arr[4]=4

Count\_arr[5]=0+4

Count\_arr[5]=4

Count\_arr[6]=0+4

Count\_arr[6]=4

Count\_arr[7]=0+4

Count\_arr[7]=4

Count\_arr[8]=0+4

Count\_arr[8]=4

Count\_arr[9]=0+4

Count\_arr[9]=4

Updated count\_arr=[2,3,3,3,4,4,4,4,4,4]

For I in range(0,2)

I=0

Index=2050//1

Index=2050

Output\_arr[count\_arr[index%10]-1]=initial\_arr[i]

Output\_arr[count\_arr[0]-1=initial\_arr[0]

Output\_arr[2-1]=initial\_arr[0]

Output\_arr[1]=2050

Count\_arr[0]-=1

I=1

Index=1//1

Index=1

Output\_arr[count\_arr[index%10]-1]=initial\_arr[i]

Output\_arr[count\_arr[1]-1=initial\_arr[1]

Output\_arr[3-1]=initial\_arr[1]

Output\_arr[2]=1

Count\_arr[2]-=1

I=0

Index=2050//1

Index=2050

Output\_arr[count\_arr[index%10]-1]=initial\_arr[i]

Output\_arr[count\_arr[0]-1=initial\_arr[0]

Output\_arr[2-1]=initial\_arr[0]

Output\_arr[1]=2050

Count\_arr[2]-=1

I=0

Index=2050//1

Index=2050

Output\_arr[count\_arr[index%10]-1]=initial\_arr[i]

Output\_arr[count\_arr[0]-1=initial\_arr[0]

Output\_arr[2-1]=initial\_arr[0]

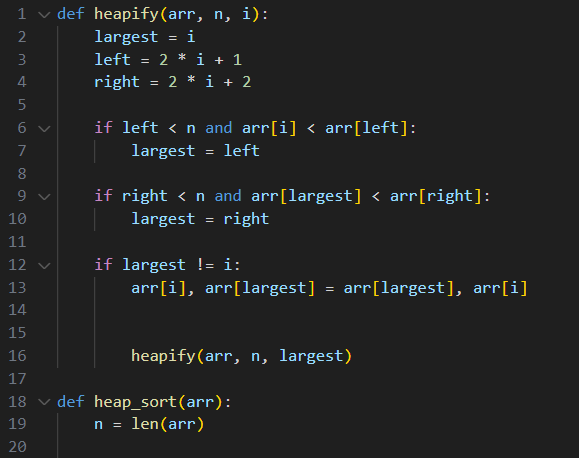
Output\_arr[1]=2050

Count\_arr[2]-=1

LAB#28

Q.1: Perform a detail dry run of the below code(LAB#28 Q.1) with a visual explanation for each step.

Solution:



A screen shot of a computer program

Description automatically generated

Result:



Explanation of code:

Heap\_sort(15,5,20,1,17,10,30)

def heap\_sort(15,5,20,1,17,10,30)

n=7

for I in range(7//2-1)

I in range(0,2)

I=0

Heapify([15,5,20,1,17,10,30],7,0)

Largest=0

Left=1

Right=2

If 1<7 and 15<5:

Largest=1

If 2<7 and 5<20

Largest=2

If 2!=0

Arr[0],arr[2]=arr[2],arr[0]

Arr[0],arr[2]=20,15

Updated array=[20,5,15,1,17,10,30]

Heapify([20,5,15,1,17,10,30],7,2)

Largest=2

Left=5

Right=6

If 5<7 and 15<10:

X

If 6<7 and 15<30

Largest=6

If 6!=2

Arr[2],arr[6]=arr[6],arr[2]

Arr[2],arr[6]=30,15

Updated array=[20,5,30,1,17,10,15]

Heapify([20,5,30,1,17,10,15],7,6)

Largest=6

Left=13

Right=14

If 13<7 and 15<0

X

If 14<7 and 15<0

X

If 6!=6

X

I=1

Heapify([20,5,30,1,17,10,15],7,1)

Largest=1

Left=3

Right=4

If 3<7 and 5<1:

X

If 4<7 and 5<17

Largest=3

If 3!=1

Arr[1],arr[3]=arr[3],arr[1]

Arr[1],arr[3]=1,5

Updated array=[20,1,15,5,17,10,

Heapify([20,1,15,5,17,10,30],7,3)

Largest=3

Left=5

Right=6

If 5<7 and 5<10:

Largest=5

If 6<7 and 10<30

Largest=6

If 6!=3

Arr[3],arr[6]=arr[6],arr[3]

Arr[3],arr[6]=30,5

Updated array=[20,5,30,1,17,10,5]

Heapify([20,5,30,1,17,10,15],7,6)

Largest=6

Left=13

Right=14

If 13<7 and 15<0

X

If 14<7 and 15<0

X

If 6!=6

X

I=2

Heapify([20,5,30,1,17,10,15],7,2)

Largest=2

Left=5

Right=6

If 5<7 and 30<10:

X

If 6<7 and 30<15

X

If 2!=2

X

Same as it is the program execute and the array is sorted in ascending order and the last output is

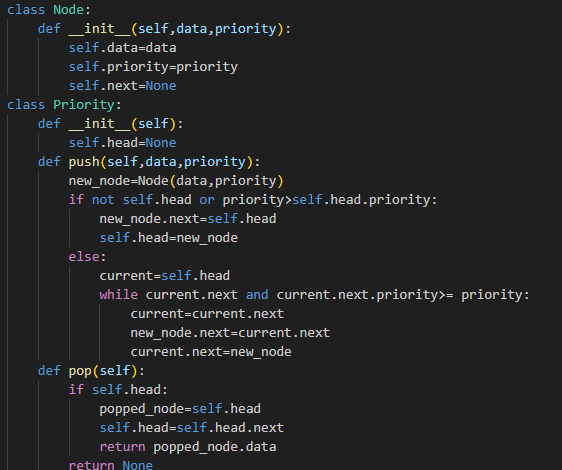
[1,5,10,15,17,20,30]

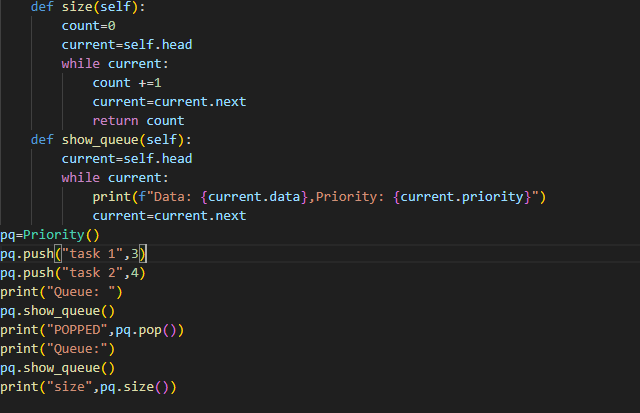
LAB#29

Q#1: Write a program to create a priority queue by using linked list.

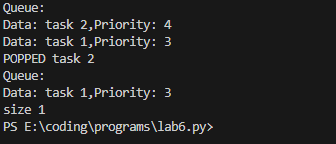
Your program must include all the functions that we used in Example #1 from Lab #29, such as push (), pop (), and others.

Program:





Output:

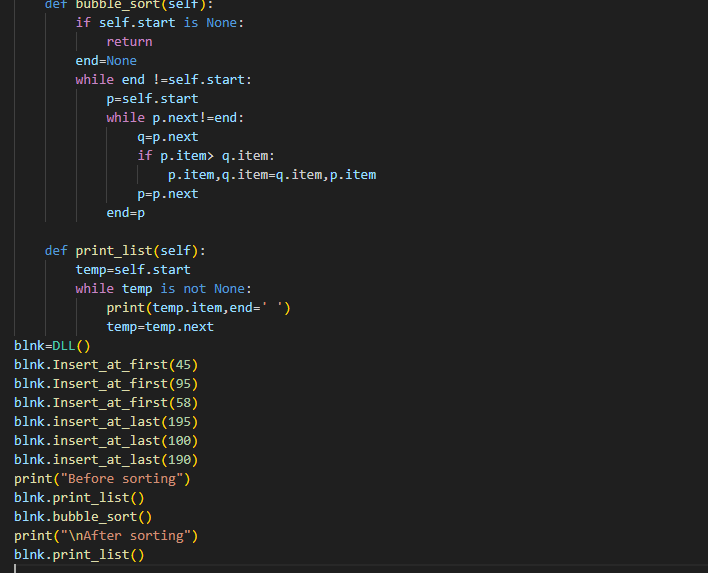


LAB#30

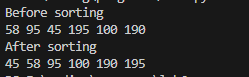
Q#1: Write a program to create a doubly linked list, then apply a sorting algorithm to arrange the item in ascending order.

Program:





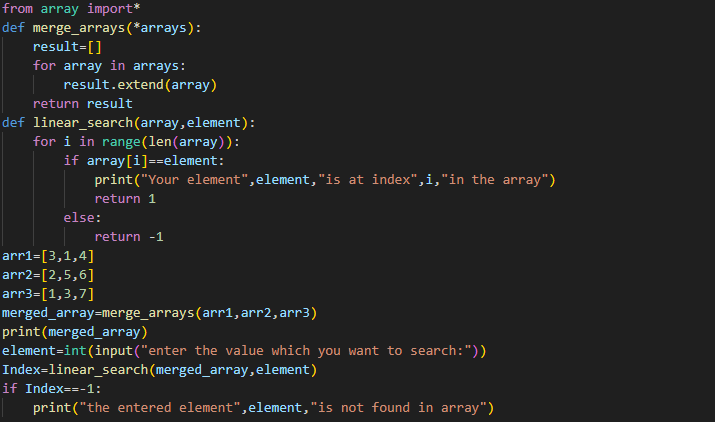
OUTPUT:



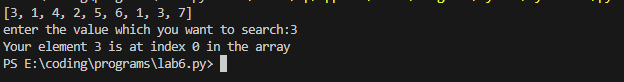
LAB#31

Q#1: Write a program to merge different unsorted arrays then apply linear search algorithm to search a particular element from that array.

Program:

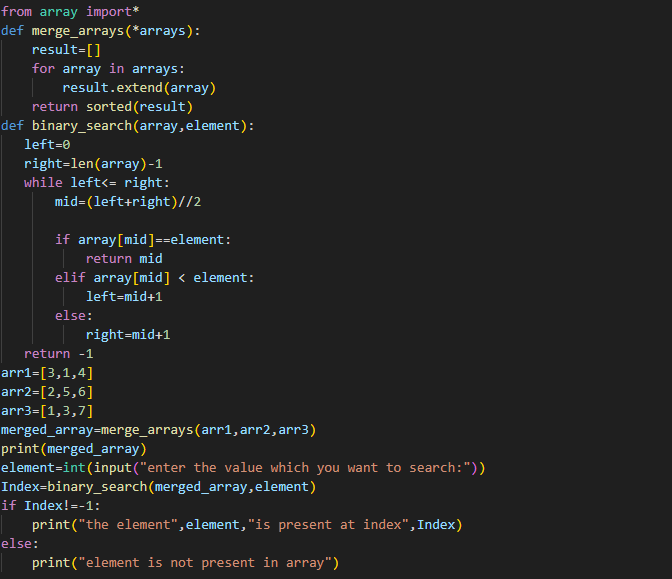


Output:



Q#2: Write a program to merge different unsorted arrays then sorted the array in ascending order and then apply the binary search algorithm to search a particular element from that array.

Program:



Output:

