|  |  |  |
| --- | --- | --- |
| **CLASS:** M.Sc. CS |  | **SEM:** I(2022-2023) |
| **SUBJECT:** Analysis of Algorithm and Researching Computing |  | **PAPER:** I |
| **NAME:** Ankit Ramprakash Prajapati |  | **ROLL NO.:** 521 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | **INDEX** |  |  |
| **NO** | **DATE** | **TITLE** | **PAGE NO.** | **SIGN** |
|  |  |  |  |  |
| 1 |  | Randomized Selection Algorithm | 2 |  |
| 2 |  | Heap Sort Algorithm | 4 |  |
| 3 |  | Radix Sort Algorithm | 6 |  |
| 4 |  | Bucket Sort Algorithm | 8 |  |
| 5 |  | Floyd - Warshall Algorithm | 10 |  |
| 6 |  | Counting Sort Algorithm | 12 |  |
| 7 |  | Set Covering Problem | 14 |  |
| 8 |  | Subset Sum Problem | 16 |  |

**Practical 1**

**Q.1) Write a Program for Randomized Selection Algorithm**

from random import randrange

def partition(x, pivot\_index = 0):

i = 0

if pivot\_index !=0: x[0],x[pivot\_index] = x[pivot\_index],x[0] #swap

for j in range(len(x)-1):

if x[j+1] < x[0]:

x[j+1],x[i+1] = x[i+1],x[j+1]

i += 1

x[0],x[i] = x[i],x[0]

return x,i

def RSelect(x,k):

if len(x) == 1:

return x[0]

else:

xpart = partition(x,randrange(len(x)))

x = xpart[0] # partitioned array

j = xpart[1] # pivot index

if j == k:

return x[j]

elif j>k:

return RSelect(x[:j],k)

else:

k = k - j - 1

return RSelect(x[(j+1):], k)

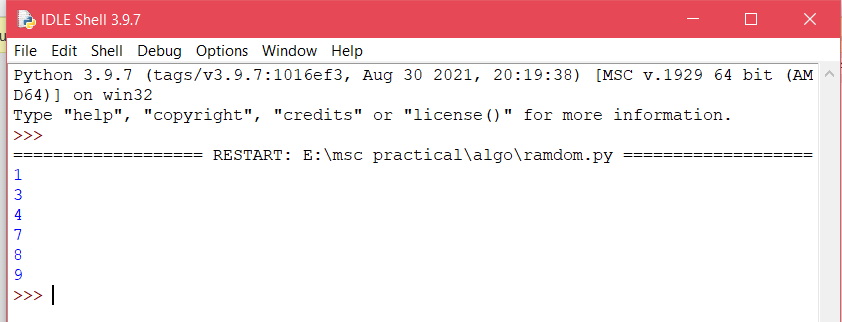
#driver code

x = [3,1,8,4,7,9]

for i in range(len(x)):

print(RSelect(x,i))

Output:



**Practical 2**

**Q.2) Write a Program for Heap Sort Algorithm**

def heapify(arr,n,i):

largest=i

l=2\*i+1

r=2\*i+2

if l<n and arr[i]<arr[l]:

largest=l

if r<n and arr[largest]<arr[r]:

largest=r

if largest !=i:

arr[i],arr[largest]=arr[largest],arr[i]#swap

heapify(arr,n,largest)

def heapSort(arr):

n=len(arr)

for i in range(n,-1,-1):

heapify(arr,n,i)

for i in range(n-1,0,-1):

arr[i],arr[0]=arr[0],arr[i]

heapify(arr,i,0)

arr=[12,11,13,5,6,7]

heapSort(arr)

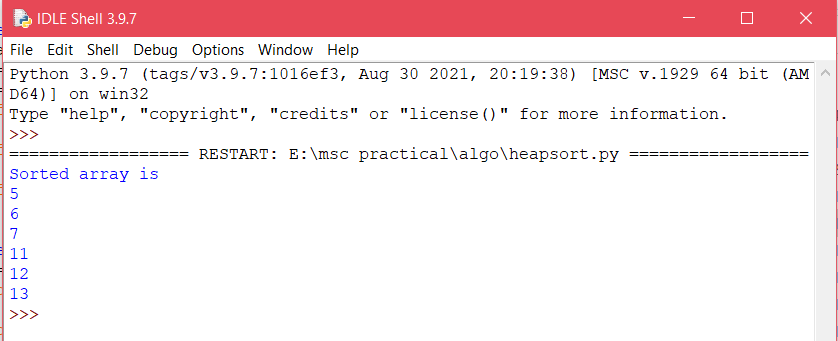
n=len(arr)

print("Sorted array is")

for i in range(n):

print("%d" %arr[i])

Output:



**Practical 3**

**Q3)Write a Program to perform Radix Sort Algorithm.**

def countingSort(arr, exp1):

n=len(arr)

output=[0]\*(n)

count=[0]\*(10)

for i in range(0,n):

index=arr[i]

count[index%10]+=1

for i in range(1,10):

count[i]+=count[i-1]

i=n-1

while i>=0:

index=arr[i]

output[count[index%10]-1]=arr[i]

count[index%10]-=1

i-=1

i=0

for i in range(0,len(arr)):

arr[i]=output[i]

def radixSort(arr):

max1=max(arr)

exp=1

while max1/exp>=1:

countingSort(arr,exp)

exp\*=10

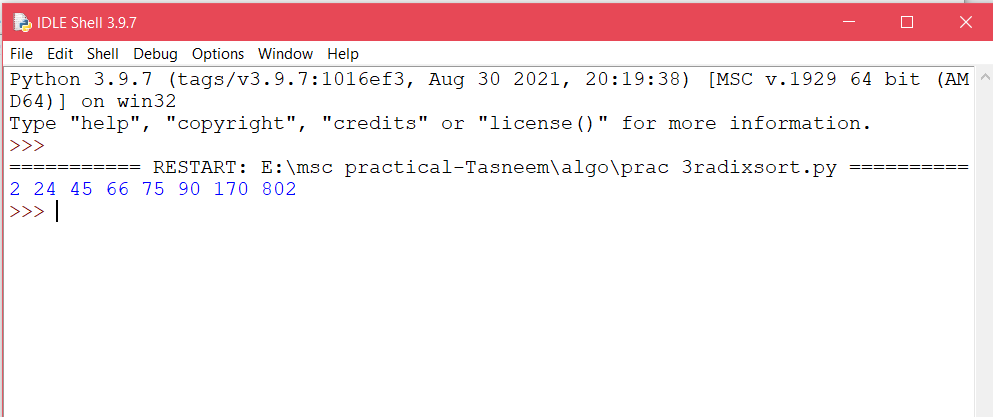
arr=[170,45,75,90,802,24,2,66]

radixSort(arr)

for i in range(len(arr)):

print(arr[i]),

Output:



**Practical 4**

**Q4) Write a Program to Perform Bucket Sort Algorithm.**

def insertionSort(b):

for i in range(1,len(b)):

up=b[i]

j=i-1

while j>=0 and b[j]>up:

b[j+1]=b[j]

j-=1

b[j+1]=up

return b

def bucketSort(x):

arr=[]

slot\_num=10

for i in range(slot\_num):

arr.append([])

for j in x:

index\_b=int(slot\_num\*j)

arr[index\_b].append(j)

for i in range(slot\_num):

arr[i]=insertionSort(arr[i])

k=0

for i in range(slot\_num):

for j in range(len(arr[i])):

x[k]=arr[i][j]

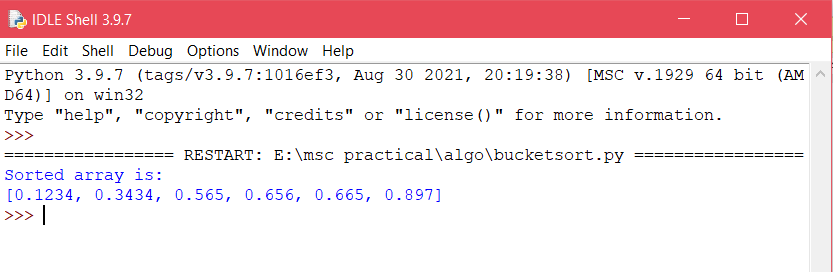
k+=1

return x

x=[0.897,0.565,0.656,0.1234,0.665,0.3434]

print("Sorted array is:")

print(bucketSort(x))

Output: 

**Practical 5**

**Q5) Write a Program to Perform Floyd-Warshall algorithm.**

V = 4

INF=99999

def floydWarshall(graph):

dist = list(map(lambda i :list( map(lambda j : j , i)) ,graph))

for k in range(V):

for i in range(V):

for j in range(V):

dist[i][j] = min(dist[i][j] ,dist[i][k]+ dist[k][j])

printSolution(dist)

def printSolution(dist):

for i in range(V):

for j in range(V):

if(dist[i][j] == INF):

print('%7s' %("INF"),)

else:

print('%7d\t' %(dist[i][j]),)

if j == V-1:

print(" ")

graph = [[0,5,INF,10],

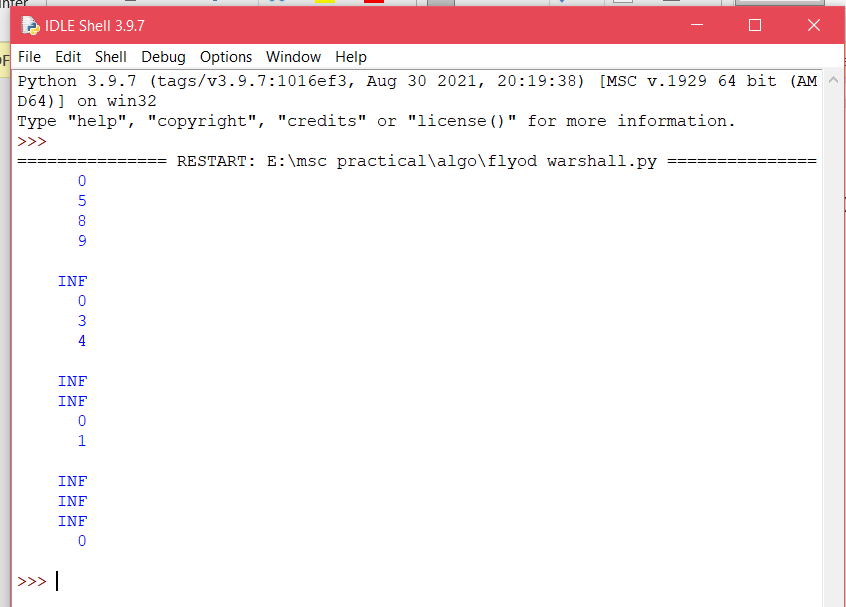
[INF,0,3,INF],

[INF,INF,0,1],

[INF,INF,INF,0]

]

floydWarshall(graph);

Output: 

**Practical 6**

**Q6) Write a Program for Counting Sort Algorithm in python.**

#the main fuction that sort the given string arr[]in

#alphabetical order

def countSort(arr):

#the output character array that will have sorted arr

output=[0 for i in range(256)]

#create a count array to store count of individual

count=[0 for i in range(256)]

#for storing the resulting answer since the

#string is immutable

ans=["" for \_ in arr]

#store count of each character

for i in arr:

count[ord(i)]+=1

#change count[i] so that count[i] now conatains actual

#position of this character in output array

for i in range(256):

count[i] += count[i-1]

#build the output character array

for i in range(len(arr)):

output[count[ord(arr[i])]-1]= arr[i]

count[ord(arr[i])]-=1

#copy the output array to arr,so that arr now

#contains sorted character

for i in range(len(arr)):

ans[i]=output[i]

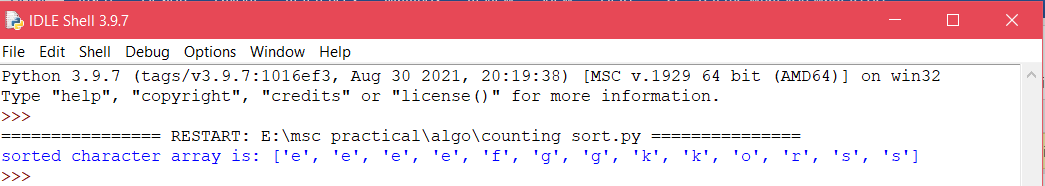
return ans

#deriver program to test above function

arr="geeksforgeeks"

ans=countSort(arr)

print('sorted character array is:',ans)

Output: 

**Practical 7**

**Q7) Write a program for Set Covering Problem.**

def set\_cover(universe,subsets):

"""Find a family of subsets that covers the universal set"""

elements=set(e for s in subsets for e in s)

#check the subsets cover the universe

if elements !=universe:

return None

covered=set()

cover=[]

#greedily add the subsets with the most uncovered points

while covered !=elements:

subset=max(subsets,key=lambda s: len(s-covered))

cover.append(subset)

covered |=subset

return cover

def main():

universe=set(range(1,11))

print(universe)

subsets=[set([1,2,3,8,9,10]),

set([1,2,3,4,5]),

set([4,5,7]),

set([5,6,7]),

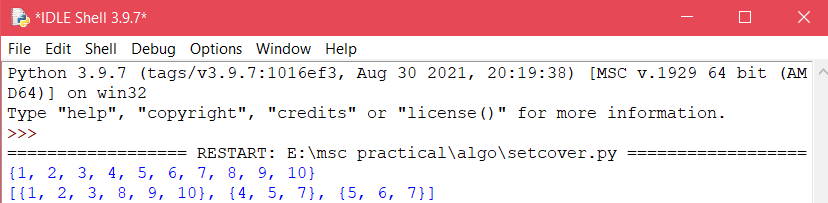
set([6,7,8,9,10])]

cover=set\_cover(universe,subsets)

print(cover)

if \_\_name\_\_=='\_\_main\_\_':

main()

Output: 

**Practical 8**

**Q8) Write a Program for found a subset with given sum.**

def isSubsetSum(set,n,sum):

if(sum==0):

return True

if(n==0 and sum!=0):

return False

if(set[n-1]>sum):

return isSubsetSum(set,n-1,sum);

return isSubsetSum(set,n-1,sum) or isSubsetSum(set,n-1,sum-set[n-1])

set=[3,34,4,12,5,2]

sum=9

n=len(set)

if(isSubsetSum(set,n,sum)==True):

print("Found a subset with given sum")

else:

print("No subset with given sum")

Output:

