## Name:-Tanvi Ajit Vichare

## Roll.No-547

## Msc.CS-Part1

## Algorithm Practical

## 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] 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:

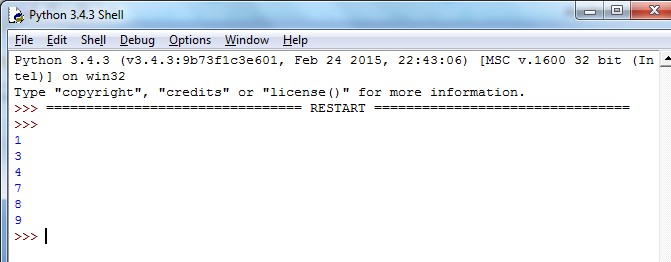
k = k - j - 1

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

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

for i in range(len(x)):

print (RSelect(x,i)) Output:



## Write a Program for Heap Sort Algorithm

Python program for implementation of heap Sort

# To heapify subtree rooted at index i. # n is size of heap

def heapify(arr, n, i):

largest = i # Initialize largest as root l = 2 \* i + 1 # left = 2\*i + 1

r = 2 \* i + 2 # right = 2\*i + 2

# See if left child of root exists and is # greater than root

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

# See if right child of root exists and is # greater than root

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

# Change root, if needed

if largest != i:

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

# Heapify the root. heapify(arr, n, largest)

# The main function to sort an array of given size def heapSort(arr):

n = len(arr)

# Build a maxheap.

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

# One by one extract elements for i in range(n-1, 0, -1):

arr[i], arr[0] = arr[0], arr[i] # swap heapify(arr, i, 0)

# Driver code to test above 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:

Sorted array is

5 6 7 11 12 13

## Write a Program to perform Radix Sort Algorithm

Python program for implementation of Radix Sort

# A function to do counting sort of arr[] according to # the digit represented by exp.

def countingSort(arr, exp1): n = len(arr)

# The output array elements that will have sorted arr output = [0] \* (n)

# initialize count array as 0 count = [0] \* (10)

# Store count of occurrences in count[] for i in range(0, n):

index = (arr[i]/exp1) count[ (index)%10 ] += 1

# Change count[i] so that count[i] now contains actual # position of this digit in output array

for i in range(1,10): count[i] += count[i-1]

# Build the output array i = n-1

while i>=0:

index = (arr[i]/exp1)

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

i -= 1

# Copying the output array to arr[],

# so that arr now contains sorted numbers i = 0

for i in range(0,len(arr)): arr[i] = output[i]

# Method to do Radix Sort def radixSort(arr):

# Find the maximum number to know number of digits max1 = max(arr)

# Do counting sort for every digit. Note that instead # of passing digit number, exp is passed. exp is 10^i # where i is current digit number

exp = 1

while max1/exp > 0: countingSort(arr,exp) exp \*= 10

# Driver code to test above

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

radixSort(arr)

for i in range(len(arr)): print(arr[i]),

Output:

# 2 24 45 66 75 90 170 802

## Write a Program to Perform Bucket Sort Algorithm

# Python3 program to sort an array # using bucket sort

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 # 10 means 10 slots, each

# slot's size is 0.1 for i in range(slot\_num):

arr.append([])

# Put array elements in different buckets for j in x:

index\_b = int(slot\_num \* j) arr[index\_b].append(j)

# Sort individual buckets for i in range(slot\_num):

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

# concatenate the result k = 0

for i in range(slot\_num):

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

k += 1

return x

# Driver Code

x = [0.897, 0.565, 0.656,

0.1234, 0.665, 0.3434]

print("Sorted Array is") print(bucketSort(x))

Output:

Sorted array is

0.1234 0.3434 0.565 0.656 0.665 0.897

## Write a Program to Perform Folyd-Warshall algorithm

# Python Program for Floyd Warshall Algorithm # Number of vertices in the graph

V = 4

# Define infinity as the large enough value. This value will be # used for vertices not connected to each other

INF = 99999

# Solves all pair shortest path via Floyd Warshall Algorithm def floydWarshall(graph):

""" dist[][] will be the output matrix that will finally

have the shortest distances between every pair of vertices """ """ initializing the solution matrix same as input graph matrix

OR we can say that the initial values of shortest distances are based on shortest paths considering no

intermediate vertices """

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

""" Add all vertices one by one to the set of intermediate vertices.

---> Before start of an iteration, we have shortest distances between all pairs of vertices such that the shortest distances consider only the vertices in the set

{0, 1, 2, .. k-1} as intermediate vertices.

----> After the end of a iteration, vertex no. k is added to the set of intermediate vertices and the

set becomes {0, 1, 2, .. k} """

for k in range(V):

# pick all vertices as source one by one for i in range(V):

# Pick all vertices as destination for the # above picked source

for j in range(V):

# If vertex k is on the shortest path from

# i to j, then update the value of dist[i][j] dist[i][j] = min(dist[i][j] ,

dist[i][k]+ dist[k][j]

)

printSolution(dist)

# A utility function to print the solution def printSolution(dist):

print "Following matrix shows the shortest distances\ between every pair of vertices"

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 ""

# Driver program to test the above program # Let us create the following weighted graph """

10

(0)------->(3)

| /|\

5 | |

| | 1

\|/ |

(1)------->(2)

3 """

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

[INF,0,3,INF],

[INF, INF, 0, 1],

[INF, INF, INF, 0]

]

# Print the solution floydWarshall(graph);

Output:

Following matrix shows the shortest distances between every pair of

vertices

0

INF

INF INF

5

0

INF INF

8

3

0

INF

9

4

1

0

## Write a Program for Counting Sort Algorithm in python

Python program for counting sort

# The main function 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 inidividul # characters and initialize count array as 0

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 contains 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 characters

for i in range(len(arr)): ans[i] = output[i]

return ans

# Driver program to test above function arr = "geeksforgeeks"

ans = countSort(arr)

print "Sorted character array is %s" %("".join(ans))

# Sorted character array is eeeefggkkorss

## 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))

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:

# [set([1, 2, 3, 8, 9, 10]), set([4, 5, 7]), set([5, 6, 7])]

1. Write a Program for found a subset with given sum

# A recursive solution for subset sum # problem

# Returns true if there is a subset

# of set[] with sun equal to given sum def isSubsetSum(set,n, sum) :

# Base Cases

if (sum == 0) : return True

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

# If last element is greater than # sum, then ignore it

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

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

# else, check if sum can be obtained # by any of the following

# (a) including the last element # (b) excluding the last element

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

# Driver program to test above function 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:

# Found a subset with given sum