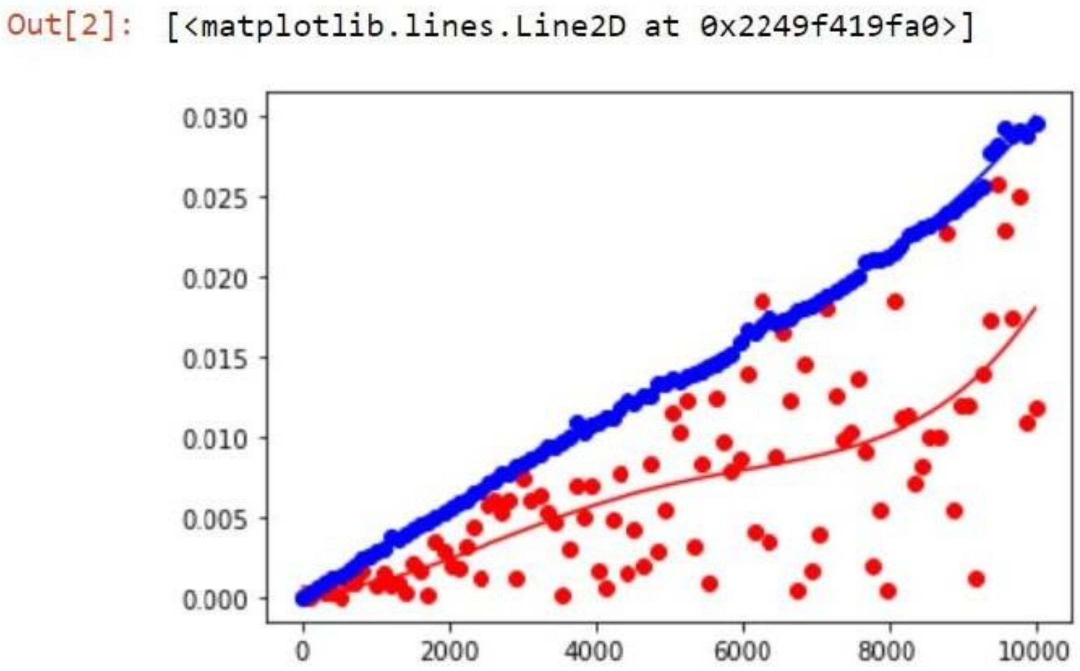
Ex. No: 1 Linear Search

Program

```
  import matplotlib.pyplot as plt
  import numpy as np
  import timeit import math
  import random
  def linear_Search(list1,n,key):
     for i in range(0,n):
        if(list1[i]==key):
        return i
     return -1
  list1=[1,3,5,4,7,9]
  key=7 n=len(list1)
  res=linear_Search(list1,n,key)
  if(res==-1):
     print("element not found")
  else:
     print("element found at index:",res)
  def contains(lst,x):
     for y in lst:
        if x==y:
        return True
     return False
ns=np.linspace(10,10_000,100,dtype=int)

     ts=[timeit.timeit('contains(lst,0)',
                 setup='lst=list(range({}));random.shuffle(lst)'.format(n),
                 globals=globals(),number=100) for n in ns]
  plt.plot(ns,ts,'or')
  degree=4
  coeffs=np.polyfit(ns,ts,degree) p=np.poly1d(coeffs)
  plt.plot(ns,[p(n) for n in ns],'-r')




  ts=[timeit.timeit('contains(lst,-1)',
                 setup='lst=list(range({}));random.shuffle(lst)'.format(n), globals=globals(),
                 number=100)
                   for n in ns]
  plt.plot(ns,ts,'ob')
  degree=4
  coeffs=np.polyfit(ns,ts,degree) p=np.poly1d(coeffs)
  plt.plot(ns,[p(n) for n in ns],'-b')
```

Output:

element found at index: 4



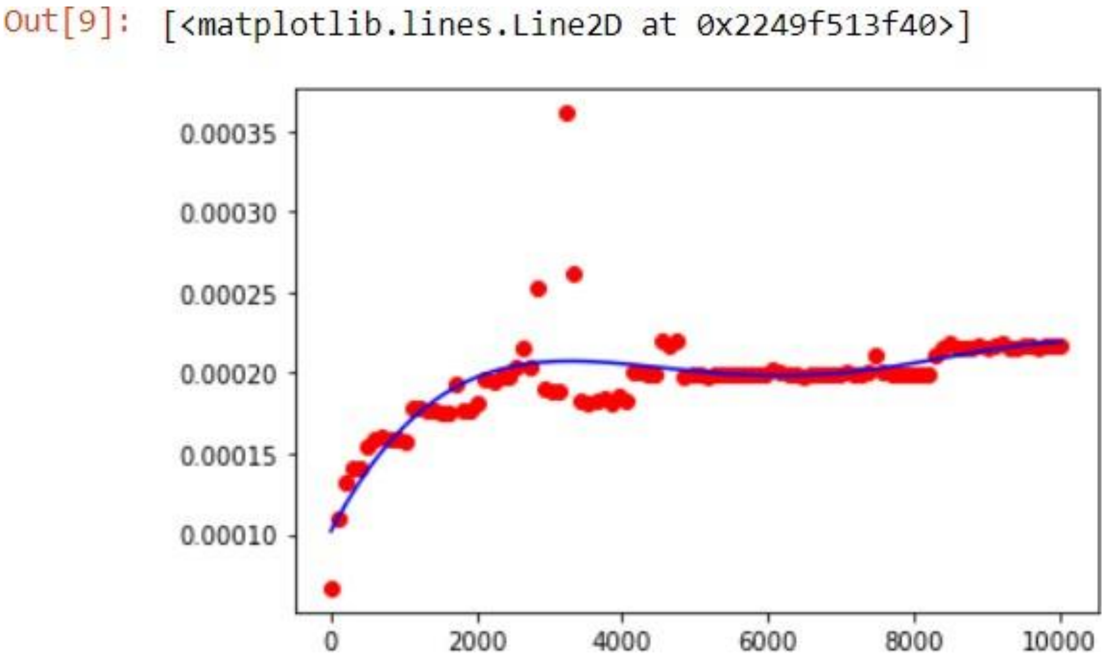
```
Out[2]: [<matplotlib.lines.Line2D at 0x2249f419fa0>]
```

Ex. No: 2 Binary Search

Program

```
import matplotlib.pyplot as plt
import numpy as np
import timeit import math
import random
def search(nums,target):
start=0
end=len(nums)-1
 while start<=end:
        mid=start+(end-start)//2
        if nums[mid]>target:
            end=mid-1
        elif nums[mid]<target:satrt=mid+1
        else:
            return mid
            return -1
if___name___=='__main___':
    nums=[2,12,15,17,27,29,45]
    target=17
    print(search(nums,target))
    def contains(lst,x):
    lo=0 hi=len(lst)-1 while lo
    <= hi:
        mid=(lo+hi)//2if x<lst[mid]:
            hi=mid-1
            elif x>lst[mid]:
            lo=mid+1
            else:
            return True
            else:
        return False
        ns=np.linspace(10,10_000,100,dtype=int)
ts=[timeit.timeit('contains(lst,0)',
                setup='lst=list(range({}));random.shuffle(lst)'.format(n), globals=globals(),
                number=100)
                 for n in ns]
plt.plot(ns,ts,'or')
degree=4
coeffs=np.polyfit(ns,ts,degree)
p=np.poly1d(coeffs)
plt.plot(ns,[p(n) for n in ns],'-b')
```

Output:



```
Out[9]: [<matplotlib.lines.Line2D at 0x2249f513f40>]
```

0.00035
0.00030
0.00025
0.00020
0.00015
0.00010
0
2000
4000
6000
8000
10000

Ex. No: 3 Pattern Matching

Program

```
# Python3 program for Naive Pattern
# Searching algorithm
def search(pat, txt):
    M = len(pat)
    N = len(txt)
    # A loop to slide pat[] one by one */
    for i in range(N - M + 1):
        j = 0
        # For current index i, check
        # for pattern match */
        while(j < M):
            if (txt[i + j] != pat[j]):
                break
            j += 1

        if (j == M):
            print("Pattern found at index ", i)
# Driver's Code
if __name__ == '__main__':
    txt = "AABAACAADAABAAABAA"
    pat = "AABA"

    # Function call
    search(pat, txt)
```

Output:

```
Pattern found at index  0
Pattern found at index  9
Pattern found at index  13
```

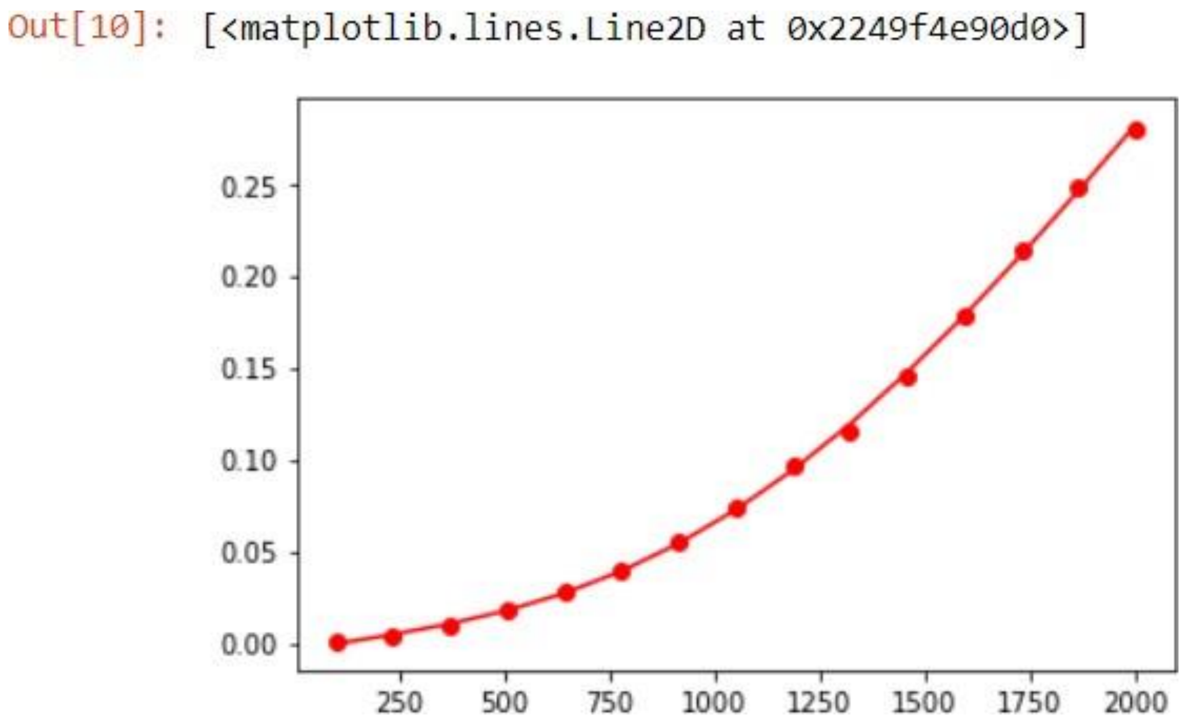
Ex. No: 4 Sorting the elements using Insertion and Heap

Program

```
import matplotlib.pyplot as plt
import numpy as np
import timeit import math
import random
def insertionSort(arr):
    for i in range(1,len(arr)):key=arr[i]
        j=i-1
        while j>=0 and key<arr[j]:arr[j+1]=arr[j]
            j-=1
        arr[j+1]=key
arr=[12,11,13,5,6]
insertionSort(arr)
for i in range(len(arr)):
     print("%d"%arr[i])
def insertion_sort(lst):
                for i in range(1,len(lst)):
                   for j in range(i,0,-1):
                        if lst[j-1]>lst[j]:
                lst[j-1],lst[j]=lst[j],lst[j-1]
                else:
                break ns=np.linspace(100,2000,15,dtype=int)
ts=[timeit.timeit('insertion_sort(lst)',
                setup='lst=list(range({}));random.shuffle(lst)'.format(n),
                globals=globals(),
                number=1)
                 for n in ns]
plt.plot(ns,ts,'or')
degree=4
coeffs=np.polyfit(ns,ts,degree)
p=np.poly1d(coeffs)
plt.plot(ns,[p(n) for n in ns],'-r')
```

Output:

```
 5
 6
11
12
13
```



```
Out[10]: [<matplotlib.lines.Line2D at 0x2249f4e90d0>]
```

Heap Sort

```
# 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[largest] <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//2 - 1, -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's code
if __name__ == '__main__':
    arr = [12, 11, 13, 5, 6, 7]
    # Function call
    heapSort(arr)
    N = len(arr)
    print("Sorted array is")
    for i in range(N):
        print("%d" % arr[i], end=" ")
```

Output:

```
Sorted array is
5 6 7 11 12 13
```

Ex. No: 5 Graph Traversal using Breadth First Search

Program

```
from  collections import  defaultdict
class Graph:
    # Constructor
    def __init__(self):
        # default dictionary to store graph
        self.graph = defaultdict(list)
    # function to add an edge to graph
    def addEdge(self,u,v):
        self.graph[u].append(v)
    # Function to print a BFS of graph
    def BFS(self, s):
        # Mark all the vertices as not visited
        visited = [False] * (len(self.graph))
        # Create a queue for BFS
        queue = []
# Mark the source node as
        # visited and enqueue it
        queue.append(s)
        visited[s] = True
        while queue:
            # Dequeue a vertex from
            # queue and print it
            s = queue.pop(0)
            print (s, end = " ")

            for i in self.graph[s]:

                if visited[i] == False:

                    queue.append(i)

                    visited[i] = True

g = Graph()

g.addEdge(0, 1)

g.addEdge(0, 2)

g.addEdge(1, 2)

g.addEdge(2, 0)

g.addEdge(2, 3)

g.addEdge(3, 3)

print ("Following is Breadth First Traversal" (starting from vertex 2)")

g.BFS(2)
```

Output:

```
Following is Breadth First Traversal(starting from vertex 2)
2 0 3 1
```

Ex. No: 6 Graph Traversal using Depth First Search

Program

```
  from collections import defaultdict
  class Graph:
     def___init__(self):
        self.graph=defaultdict(list)
     def addEdge(self,u,v): self.graph[u].append(v)
     def DFSUtil(self,v,visited):
              visited[v]=True print (v)
              for i in self.graph[v]: if
                 visited[i]==False:
                    self.DFSUtil(i,visited)def
     DFS(self):
                       V=len(self.graph)
                       visited=[False]*(V)for i in
                       range(V):
                          if visited[i]==False:
                             self.DFSUtil(i,visited)
  g=Graph() g.addEdge(0,1)
  g.addEdge(0,2)g.addEdge(1
  ,2)g.addEdge(2,0)
  g.addEdge(2,3)
  g.addEdge(3,3)
print("Following is Depth First Traversal")
g.DFS()
```

Output:

```
  Following is Depth First Traversal

   0    1        2        3
```

Ex. No: 7 Dijkstra’s Algorithm

Program

class Graph():

```
class Graph():
    def___init___(self, vertices):
    self.V = vertices
        self.graph = [[0 for column in range(vertices)]
        for row in range(vertices)]
    def printSolution(self, dist):
        print("Vertex \t Distance from Source")
        for node in range(self.V):
            print(node, "\t\t", dist[node])
            def minDistance(self, dist, sptSet):
        min = 1e7
        for v in range(self.V):
            if dist[v] < min and sptSet[v] == False:
            min = dist[v]
                min_index = v
        return min_index def
    dijkstra(self, src):
        dist = [1e7] * self.Vdist[src] = 0
        sptSet = [False] * self.V for cout in
        range(self.V):
            u = self.minDistance(dist, sptSet)
            sptSet[u] = True
            for v in range(self.V):
                if (self.graph[u][v] > 0 and
                sptSet[v] == False and
                dist[v] > dist[u] + self.graph[u][v]):
                dist[v] = dist[u] + self.graph[u][v]
        self.printSolution(dist)
        g = Graph(9)
g.graph = [[0, 4, 0, 0, 0, 0, 0, 8, 0],
        [4, 0, 8, 0, 0, 0, 0, 11, 0],
        [0, 8, 0, 7, 0, 4, 0, 0, 2],
        [0, 0, 7, 0, 9, 14, 0, 0, 0],
        [0, 0, 0, 9, 0, 10, 0, 0, 0],
        [0, 0, 4, 14, 10, 0, 2, 0, 0],
        [0, 0, 0, 0, 0, 2, 0, 1, 6],
        [8, 11, 0, 0, 0, 0, 1, 0, 7],
        [0, 0, 2, 0, 0, 0, 6, 7, 0] ]
g.dijkstra(0)
```

Output:

```
Vertex    Distance from Source
0                   0
1                   4
2                   12
3                   19
4                   21
5                   11
6                   9
7                   8
8                   14
```

Ex. No: 8 Prim’s Algorithm

Program

```
class Graph:
    def___init___(self, num_of_nodes): self.m_num_of_nodes
        = num_of_nodes
        self.m_graph = [[0 for column in range(num_of_nodes)]
        for row in range(num_of_nodes)]
    def add_edge(self, node1, node2, weight):
        self.m_graph[node1][node2] = weight
        self.m_graph[node2][node1] = weight
    def prims_mst(self): postitive_inf =
        float('inf')
        selected_nodes = [False for node in range(self.m_num_of_nodes)]
        result = [[0 for column in range(self.m_num_of_nodes)]
                    for row in range(self.m_num_of_nodes)]
                    indx = 0
        for i in range(self.m_num_of_nodes):
            print(self.m_graph[i])
        print(selected_nodes)
        while(False in selected_nodes):
            minimum = postitive_inf
            start = 0
            end = 0
            for i in range(self.m_num_of_nodes):
            if selected_nodes[i]:
                    for j in range(self.m_num_of_nodes):
                        if(not selected_nodes[j] and self.m_graph[i][j]>0):
                        if self.m_graph[i][j] < minimum:
                                minimum = self.m_graph[i][j]
                                start, end = i, j
            selected_nodes[end] = True
            result[start][end] = minimum
            if minimum == postitive_inf:
                result[start][end] = 0
            print("(%d.) %d - %d: %d" % (indx, start, end, result[start][end]))
            indx += 1
            result[end][start] = result[start][end]
        for i in range(len(result)):
            for j in range(0+i, len(result)):
            if result[i][j] != 0:
                    print("%d - %d: %d" % (i, j, result[i][j]))example_graph
= Graph(9)
example_graph.add_edge(0, 1, 4)
example_graph.add_edge(0, 2, 7)
example_graph.add_edge(1, 2, 11)
example_graph.add_edge(1, 3, 9)
example_graph.add_edge(1, 5, 20)
example_graph.add_edge(2, 5, 1)
example_graph.add_edge(3, 6, 6)
example_graph.add_edge(3, 4, 2)
example_graph.add_edge(4, 6, 10)
example_graph.add_edge(4, 8, 15)
example_graph.add_edge(4, 7, 5)
example_graph.add_edge(4, 5, 1)
example_graph.add_edge(5, 7, 3)
example_graph.add_edge(6, 8, 5)
example_graph.add_edge(7, 8, 12)
example_graph.prims_mst()
```

Output:

```
 [0, 4, 7, 0, 0, 0, 0, 0, 0]
 [4, 0, 11, 9, 0, 20, 0, 0, 0]
 [7, 11, 0, 0, 0, 1, 0, 0, 0]
 [0, 9, 0, 0, 2, 0, 6, 0, 0]
 [0, 0, 0, 2, 0, 1, 10, 5, 15]
 [0, 20, 1, 0, 1, 0, 0, 3, 0]
 [0, 0, 0, 6, 10, 0, 0, 0, 5]
 [0, 0, 0, 0, 5, 3, 0, 0, 12]
 [0, 0, 0, 0, 15, 0, 5, 12, 0]
[False, False, False, False, False, False, False, False, False]

 (0.) 0 - 0: 0
 (1.) 0 - 1: 4
 (2.) 0 - 2: 7
 (3.) 2 - 5: 1
 (4.) 5 - 4: 1
 (5.) 4 - 3: 2
 (6.) 5 - 7: 3
 (7.) 3 - 6: 6
 (8.) 6 - 8: 5

  0 - 1: 4
 0 - 2: 7
 2 - 5: 1
 3 - 4: 2
 3 - 6: 6
 4 - 5: 1
 5 - 7: 3
 6 - 8: 5
```

Ex. No: 9 Floyd’s Algorithm

Program

```
nV = 4
INF = 999
def floyd_warshall(G):
    distance = list(map(lambda i: list(map(lambda j: j, i)), G))
    for k in range(nV):
        for i in range(nV):
        for j in range(nV):
                distance[i][j] = min(distance[i][j], distance[i][k] + distance[k][j])
    print_solution(distance)
def print_solution(distance):
for i in range(nV):
        for j in range(nV):
        if(distance[i][j] == INF):
                print("INF", end=" ")else:
                print(distance[i][j], end=" ")
                print(" ")
                    G = [[0, 3, INF, 5],
                            [2, 0, INF, 4],
         [INF, 1, 0, INF],
         [INF, INF, 2, 0]]
floyd_warshall(G)
```

Output:

```
0  3  7  5
2  0  6  4
3  1  0  5
5  3  2  0
```

Ex. No: 10 Warshall’s Algorithm

Program

```
from collections import defaultdict
class Graph:
    def___init__(self, vertices):
    self.V = vertices
    def printSolution(self, reach):
        print("Following matrix transitive closure of the given graph ")
        for i in range(self.V):
            for j in range(self.V):
            if (i == j):
                    print("%7d\t" % (1),end=" ")
                    else:
                    print("%7d\t" %(reach[i][j]),end=" ")
                    print()
    def transitiveClosure(self,graph):
    reach =[i[:] for i in graph]
        for k in range(self.V):
        for i in range(self.V):
                for j in range(self.V):
                    reach[i][j] = reach[i][j] or (reach[i][k] and reach[k][j])
        self.printSolution(reach)
g= Graph(4)
graph = [[1, 1, 0, 1],
        [0, 1, 1, 0],
        [0, 0, 1, 1],
        [0, 0, 0, 1]]
g.transitiveClosure(graph)
```

Output:

```
Following matrix transitive closure of the given graph
      1         1        1         1
      0         1        1         1
      0         0        1         1
      0         0        0         1
```

Ex. No: 11 State Space Search

Program

```
 global NN = 4

def PrintSolution(board):

  for i in range(N):
      for j in range(N):
      print(board[i][j],end=")
      print()
  def isSafe(board,row,col):
  for i in range(col):
      if board[row][i] == 1:
      return False
    for i,j in zip(range(row,-1,-1),range(col,-1,-1)):
    if board[i][j] == 1:
        return False
    for i,j in zip(range(row,-1,-1),range(col,-1,-1)):
    if board[i][j] == 1:
        return False
    return True
  def SolveNQUtil(board,col):
  if col >= N:
      return True
      for i in range(N):
      if isSafe(board,i,col):board[i][col] =
        1
        if SolveNQUtil(board,col+1) == True:
        return True
        board[i][col] = 0
        return False
  def SolveNQ():
  board= [[0,0,0,0],
          [0,0,0,0],
          [0,0,0,0],
  [0,0,0,0] ]
    ifSolveNQUtil(board,0)==False:    print("SOlution
      does not exist")
    return False
    PrintSolution(board)
    return True
  SolveNQ()
```

Output:

```
0010
1000
0001
0100
True
```

Ex. No: 12 Travelling Salesman Problem

Program

```
from sys import maxsize
from itertools  import permutations
V = 4
def TravellingSalesmanProblem(graph, s):
vertex = []
    for i in range(V):
    if i != s:
            vertex.append(i)min_path =
    maxsize
    next_permutation=permutations(vertex)
    for i in next_permutation:
        current_pathweight = 0k = s
        for j in i:
            current_pathweight += graph[k][j]k = j
        current_pathweight += graph[k][s]
        min_path = min(min_path, current_pathweight)
        return min_path
if___name___== "__main__":
    graph = [[0,10,15,20], [10,0,35,25],
            [15,35,0,30], [20,25,30,0]]
    s = 0 print(TravellingSalesmanProblem(graph, s))
```

Output:

```
80
```

Ex. No: 13 Merge Sort

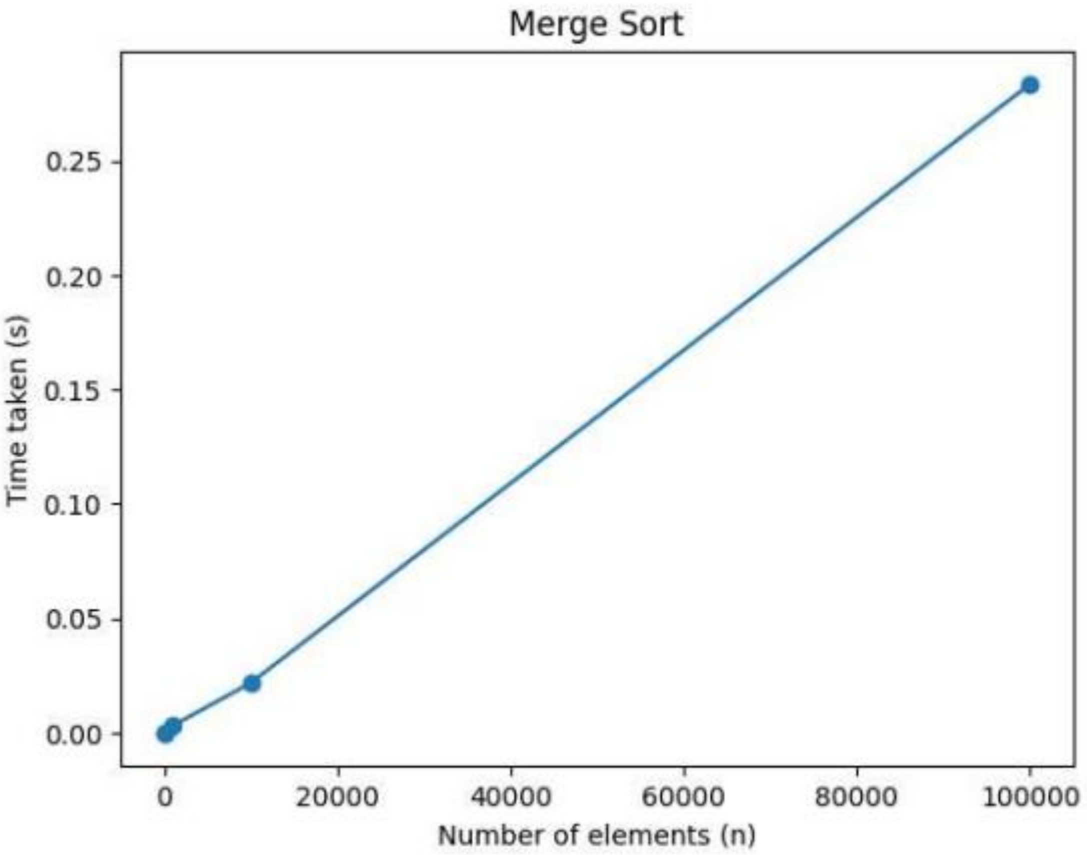
Program

```
import random
import time
 import matplotlib.pyplot as plt
def merge_sort(arr):
 if len(arr) > 1: mid = len(arr) // 2
 left_half = arr[:mid]
right_half = arr[mid:]
 merge_sort(left_half)
merge_sort(right_half)
 i = j = k = 0
 while i < len(left_half) and j < len(right_half):
 if left_half[i] < right_half[j]:
 arr[k] = left_half[i]
 i += 1
 else:
 arr[k] = right_half[j]
j += 1
 k += 1
 while i < len(left_half):
 arr[k] = left_half[i]
i += 1
k += 1
while j < len(right_half):
arr[k] = right_half[j]
```

```
 j += 1
 k += 1
def test_merge_sort(n):
arr = [random.randint(1, 100)
 for _ in range(n)]
start_time = time.time()
 merge_sort(arr)
 end_time = time.time()
 return end_time - start_time
if __name__ == '__main__':
 ns = [10, 100, 1000, 10000, 100000]
 times = []
 for n in ns:
t = test_merge_sort(n)
 times.append(t)
print(f"Merge sort took {t:.6f} seconds to sort {n} elements.")
 plt.plot(ns, times, 'o-')
 plt.xlabel('Number of elements (n)')
 plt.ylabel('Time taken (s)')
plt.title('Merge Sort')
 plt.show()
```

Output:

```
Merge sort took 0.000020 seconds to sort 10 elements.
Merge sort took 0.000249 seconds to sort 100 elements.
 Merge sort took 0.003046 seconds to sort 1000 elements.
Merge sort took 0.021679 seconds to sort 10000 elements.
Merge sort took 0.283631 seconds to sort 100000 elements.
```



Ex. No: 14 Quick Sort

Program

```
import random
 import time
 def quicksort(arr):
if len(arr) <= 1:
return arr
 pivot = arr[0]
left = []
right = []
for i in range(1, len(arr)):
 if arr[i] < pivot:
 left.append(arr[i])
else:
 right.append(arr[i])
return quicksort(left) + [pivot] + quicksort(right)
def measure_time(n, num_repeats):
 times = []
 for _ in range(num_repeats):
arr = [random.randint(0, 1000000)
for _ in range(n)]
start_time = time.time()
quicksort(arr)
end_time = time.time()
 times.append(end_time - start_time)
 return sum(times) / len(times)
 if __name__ == '__main__':

num_repeats = 10

max_n = 10000

 step_size = 100

ns = range(0, max_n + step_size, step_size)

times = []

 for n in ns:

if n == 0:

times.append(0)

else:

times.append(measure_time(n, num_repeats))

print(times)
```

Output:

```
[0, 0.00013625621795654297, 0.0006334543228149414, 0.0005178928375244414,
0.0009247779846191407, 0.000916147232055664, 0.0010011672973632812,]
```

Ex. No: 15 Randomized Algorithms for kth Smallest Element

Program

```
import random

# Function to partition the array around a pivot

 def partition(arr, low, high):

i = low - 1

pivot = arr[high]

 for j in range(low, high):

 if arr[j] <= pivot:

 i += 1

arr[i], arr[j] = arr[j], arr[i]

arr[i+1], arr[high] = arr[high], arr[i+1]

return i+1

# Function to find the kth smallest number using randomized algorithm

def randomized_select(arr, low, high, k):

if low == high:

 return arr[low]

 pivot_index = random.randint(low, high)

arr[pivot_index], arr[high] = arr[high], arr[pivot_index]

index = partition(arr, low, high)

 if k == index:

return arr[k]

elif k < index:

 return randomized_select(arr, low, index-1, k)
 else:
return randomized_select(arr, index+1, high, k)
# Testing the function
 arr = [9, 4, 2, 7, 3, 6]
 k = 3
n = len(arr)
 result = randomized_select(arr, 0, n-1, k-1)
 print(f"The {k}th smallest number is: {result}")
```

Output:

```
The 3th smallest number is: 4
```

Ex. No.:16 Matrix Multiplication

Program

```
#include<stdio.h>

#include<conio.h>

void main()

{

int a[3][3],b[3][3],c[3][3];

int r1,c1,r2,c2,i,j,k;

clrscr();

printf("\n Enter the no. of rows in matrix A:");

scanf("%d",&r1);

printf("\n Enter the no. of columns in matrix A:");

scanf("%d",&c1);

printf("\n Enter the no. of rows in matrix B:");

scanf("%d",&r2);

printf("\n Enter the no. of columns in matrix B:");

scanf("%d",&c2);

if(r1!=c2)

{

printf("\n Matrix multiplication is not possible...");

}

else

{

printf("Enter the elements in matrix A:\n");

for(i=0;i<r1;i++)

{
```

```
for(j=0;j<c1;j++)
{
scanf("%d",&a[i][j]);
}
}
printf("Enter the elements in matrix B:\n");
for(i=0;i<r2;i++)
{
for(j=0;j<c2;j++)
{
scanf("%d",&b[i][j]);
}
}
for(i=0;i<r1;i++)
{
for(j=0;j<c2;j++)
{
c[i][j]=0;
for(k=0;k<r1;k++)
{
c[i][j]=c[i][j]+a[i][k]*b[k][j];
}
}
}
printf("\n Product of matrices:(A*B)=\n");
for(i=0;i<r1;i++)
{
for(j=0;j<c2;j++)
```

```
{
printf("%d",c[i][j]);
}
printf("\n");
}
}
getch();
}
```

Output:

```
Enter the no. of rows in matrix A: 2
Enter the no. of columns in matrix A: 2
Enter the no. of rows in matrix B: 2
Enter the no. of columns in matrix B: 2
Enter the elements in matrix A:
2  2
2  2
Enter the elements in matrix B:
3  3
3  3
Product of matrices:(A*B)=
12  12
12  12
```

Ex. No.:17 Finding Maximum and Minimum element In the given array

Program

```
#include<stdio.h>

#include<stdio.h>

int max,min;

int a[100];

void maxmin(int i,int j)

{

int max1,min1,mid;

if(i==j)

{

max=min=a[i];

}

else

{

if(i==j-1)

{

if(a[i]<a[j])

{

max=a[j];

min=a[i];

}

else

{
```

```
max=a[i];
min=a[j];
}
}
else
{
mid=(i+j)/2;
maxmin(i,mid);
max1=max;
min1=min;
maxmin(mid+1,j);
if(max<max1)
max=max1;
if(min>min1)
min=min1;
}
}
}
int i,num;
printf("\n Enter the total number of numbers:");
scanf("%d",&num);
printf("Enter the number:\n");
for(i=1;i<=num;i++)
scanf("%d",&a[i]);
max=a[0];
min=a[0];
maxmin(1,num);
printf("Minimum element in the array:%d\n",min);
printf("Maximum element in the array:%d\n",max);

return 0;

}
```

Output:

```
Enter the total number of numbers: 5

Enter the number:

29

21

64

27

20

Minimum element in the array: 20

Minimum element in the array: 64
```