```
#1.Linear search using class and object
import array as mya
class linear_search:
  def ls(self,a,n,k):
    for i in range(0,n):
      if a[i]==k:
        return ("The element is present at index "+str(i)+" in the list")
    return ("The key element you are searching for is not present in the list")
l1=[]
a=mya.array('b',l1)
lim=int(input("Enter the limit"))
for i in range(lim):
  el=int(input("Enter the element"))
  a.append(el)
  lim=lim-1
k=int(input("Enter key value"))
n=len(a)
obj=linear_search()
obj.ls(a,n,k)
```

```
Enter the limit3
Enter the element5
Enter the element2
Enter the element1
Enter key value6
'The key element you are searching for is not present in the lis
t'
```

```
#2.Binary search
class bis:
  def bs(self,l1,low,high):
    while low<=high:
      mid=(low+high)//2
      if l1[mid]<k:
        low=mid+1
      elif l1[mid]>k:
        high=mid-1
      else:
        return mid
    return -1
ob1=bis()
l1 = []
lim=int(input("Enter the limit"))
for i in range(lim):
   el=int(input("Enter the element"))
   l1.append(el)
   lim=lim-1
k=int(input("Enter a key value"))
leng=len(l1)
res=ob1.bs(l1,0,leng-1)
if res == -1:
   print("element is not present")
else:
   print("element is at position ",res)
```

```
Enter the limit20
```

```
Enter the element1
Enter the element2
Enter the element3
Enter the element4
Enter the element5
Enter the element6
Enter the element7
Enter the element8
Enter the element9
Enter the element10
Enter the element11
Enter the element12
Enter the element13
Enter the element14
Enter the element15
Enter the element16
Enter the element17
Enter the element18
Enter the element19
Enter the element20
Enter a key value21
element is not present
```

```
#3.Insertion sort.
def ins(l1):
  for i in range(1,len(l1)):
    temp=l1[i]
    j=i-1
    while j>=0 and temp<l1[j]:
      l1[j+1]=l1[j]
      j=j-1
    l1[j+1]=temp
l1=[]
lim=int(input("Enter the limit"))
for i in range(lim):
  el=int(input("Enter the element"))
  l1.append(el)
  lim=lim-1
ins(l1)
for i in range(len(l1)):
  print(l1[i])
```

```
Enter the limit5
Enter the element1
Enter the element4
Enter the element3
Enter the element5
Enter the element2
1
2
3
4
5
```

```
#4.Bubble sort
```

```
def bbs(l1):
  for i in range(0,len(l1)-1):
    for j in range(len(l1)-1):
      if (l1[j]>l1[j+1]):
        l1[j],l1[j+1]=l1[j+1],l1[j]
  return l1
l1=[]
lim=int(input("Enter the limit"))
for i in range(lim):
  el=int(input("Enter the element"))
  l1.append(el)
  lim=lim-1
print("The unsorted list is ",l1)
res=bbs(l1)
print("The sorted list ",l1)
```

```
Enter the limit5
Enter the element6
Enter the element8
Enter the element9
Enter the element7
Enter the element10
The unsorted list is  [6, 8, 9, 7, 10]
The sorted list  [6, 7, 8, 9, 10]
```

```
#5.Selection sort
def ss(l1):
  for i in range(len(l1)-1):
    #min=i
    for j in range(i+1,len(l1)):
      if l1[j]<l1[i]:
        #min=j
        (l1[i],l1[j])=(l1[j],l1[i])
    print(l1)

l1=[]
lim=int(input("Enter the limit"))
for i in range(lim):
  el=int(input("Enter the element"))
  l1.append(el)
  lim=lim-1
res=ss(l1)
```

```
Enter the limit4
Enter the element3
Enter the element4
Enter the element2
Enter the element1
[1, 4, 3, 2]
[1, 2, 4, 3]
[1, 2, 3, 4]
```

```
#maximum of n numbers
l1=[2000,50,70,90,100,200]
n=len(l1)
res=l1[0]
for i in range(1,n):
```

```
  if l1[i]>res:
    res=l1[i]
print(res)
```

```
2000
```

```
#Linear search method 2
class lse:
  def ls(self,l1,n,k):
    for i in l1:
      if i==k:
        return ("The element is present at index "+str(l1.index(i))+" in the list")
    return ("The key element you are searching for is not present in the list")
l1=[1,2,5,7,8,10,12,15,67,89,3,5,7,9,0]
n=len(l1)
k=int(input("Enter the element to find: "))
obj=lse()
obj.ls(l1,n,k)
```

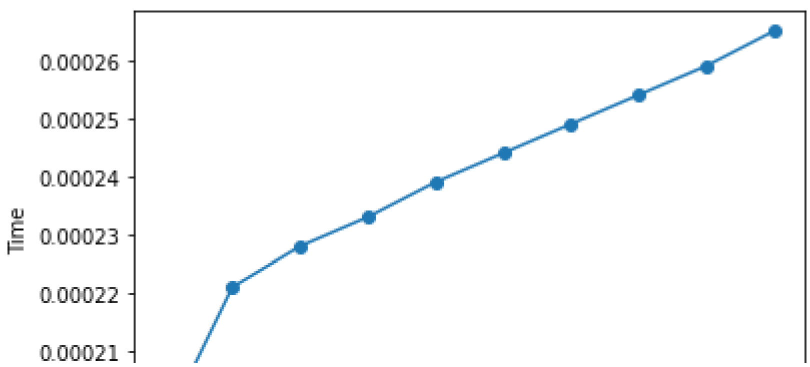
```
Enter the element to find: 10
'The element is present at index 5 in the list'
```

```
# Merge sort
import random
import time
import timeit
import matplotlib.pyplot as plt
# start = timeit.default_timer()
def merge(a1,a2):
   c=[]
   x=0
   y=0
   while(x<len(a1) and y<len(a2)):
      if(a1[x]<a2[y]):
         c.append(a1[x])
         x+=1
      else:
         c.append(a2[y])
         y+=1
   while(x<len(a1)):
      c.append(a1[x])
      print(c)
      x+=1
   while(y<len(a2)):
      c.append(a2[y])
      print(c)
      y+=1
   return c

def mergesort(array):
   if(len(array)==1):
      return array
   mid=(len(array))//2
   a1=mergesort(array[:mid])
   a2=mergesort(array[mid:])
   return merge(a1,a2)
```

```
array=[]
x_coordinate = []
y_coordinate = []
start=time.time()
for i in range(0,10):
  n=random.randint(10,5000)
  array.append(n)
  x_coordinate.append(i*100)
  y_coordinate.append(round(time.time()-start,6))
print(mergesort(array))
plt.plot(x_coordinate, y_coordinate, marker="o")
plt.xlabel("Size")
plt.ylabel("Time")
plt.show()
```

```
[3706, 4279]
[3340, 3871]
[3340, 3841, 3871]
[3340, 3706, 3841, 3871, 4279]
[636, 3424]
[2883, 3772]
[2883, 3772, 3804]
[636, 2883, 3424, 3772]
[636, 2883, 3424, 3772, 3804]
[636, 2883, 3340, 3424, 3706, 3772, 3804, 3841]
[636, 2883, 3340, 3424, 3706, 3772, 3804, 3841, 3871]
[636, 2883, 3340, 3424, 3706, 3772, 3804, 3841, 3871, 4279]
[636, 2883, 3340, 3424, 3706, 3772, 3804, 3841, 3871, 4279]
```



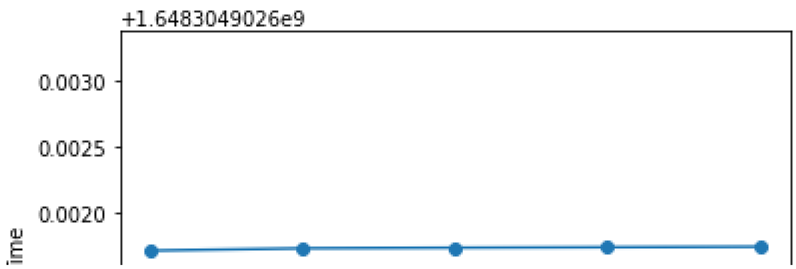

```
a=[1,2,3,4,5]
mid=len(a)//2
a1=a[:mid+1]
a2=a[mid+1:]
print(a1)
a2
```

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

```
# Quick sort
```

```
# import random
# def partition(my_arr, start, end):
#   pivot = my_arr[end]
#   i = start-1
#   for j in range(start, end):
#     if my_arr[j]<=pivot:
#       i=i+1
#       my_arr[i], my_arr[j] = my_arr[j], my_arr[i]
#     print(my_arr)
#   my_arr[i+1], my_arr[end] = my_arr[end], my_arr[i+1]
#   return i+1
# def quicksort(my_arr, start, end):
#   if start<end:
#     q = partition(my_arr, start, end)
#     quicksort(my_arr, start, q-1)
#     quicksort(my_arr, q+1, end)
# my_arr = []
# for i in range(0,5):
#   n=random.randint(100,500)
#   my_arr.append(n)
# quicksort(my_arr, 0, 4)
# print(my_arr)
import time
import matplotlib.pyplot as plt
# start1 = timeit.default_timer()
# Quick sort
import random
def partition(my_arr, start, end):
  pivot = my_arr[end]
  i = start-1
  for j in range(start, end):
    if my_arr[j]<=pivot:
      i=i+1
      my_arr[i], my_arr[j] = my_arr[j], my_arr[i]
    print(my_arr)
  my_arr[i+1], my_arr[end] = my_arr[end], my_arr[i+1]
  return i+1
def quicksort(my_arr, start, end):
  if start<end:
    q = partition(my_arr, start, end)
    quicksort(my_arr, start, q-1)
    quicksort(my_arr, q+1, end)
my_arr = []
x_coordinate = []
y_coordinate = []
start=time.time()
for i in range(0,5):
  n=random.randint(100,500)
  my_arr.append(n)
  x_coordinate.append(i*100)
  y_coordinate.append(round(time.time()-start1,8))
quicksort(my_arr, 0, 4)
print(my_arr)
print('Time: ', time.time()- start)
plt.plot(x_coordinate, y_coordinate, marker="o")
plt.xlabel("Size")
plt.ylabel("Time")
plt.show()
```

```
[138, 317, 110, 306, 392]
[138, 317, 110, 306, 392]
[138, 317, 110, 306, 392]
[138, 317, 110, 306, 392]
[138, 317, 110, 306, 392]
[138, 317, 110, 306, 392]
[138, 110, 317, 306, 392]
[138, 110, 306, 317, 392]
[110, 138, 306, 317, 392]
Time:  0.0031223297119140625
```




```
# Finding max and min using divied and conquer method.
import random
def DAC_Max(a, index, l):
    max = -1
    if (index >= l - 2):
        if (a[index] > a[index + 1]):
            return a[index]
        else:
            return a[index + 1];
    max = DAC_Max(a, index + 1, l)
    if (a[index] > max):
        return a[index]
    else:
        return max
def DAC_Min(a, index, l):
    min = 0
    if (index >= l - 2):
        if (a[index] < a[index + 1]):
            return a[index]
        else:
            return a[index + 1]
    min = DAC_Min(a, index + 1, l)

    if (a[index] < min):
        return a[index]
    else:
        return min;
a=[]
for i in range(0,10000):
  n=random.randint(0,100000)
  a.append(n)
l1=len(a)-1
```

```
max = DAC_Max(a, 0, 11)
min = DAC_Min(a, 0, 11)
print("The minimum number in a given array is : ", min);
print("The maximum number in a given array is : ", max);
```

```
The minimum number in a given array is :  14279
The maximum number in a given array is :  90663
```

```
#Kurskals algorithm
from collections import defaultdict
class Graph:

    def __init__(self, vertices):
        self.V = vertices
        self.graph = []
    def addEdge(self, u, v, w):
        self.graph.append([u, v, w])
    def find(self, parent, i):
        if parent[i] == i:
            return i
        return self.find(parent, parent[i])
    def union(self, parent, rank, x, y):
        xroot = self.find(parent, x)
        yroot = self.find(parent, y)
        if rank[xroot] < rank[yroot]:
            parent[xroot] = yroot
        elif rank[xroot] > rank[yroot]:
            parent[yroot] = xroot
        else:
            parent[yroot] = xroot
            rank[xroot] += 1
    def KruskalMST(self):

        result = []
        i = 0
        e = 0
        self.graph = sorted(self.graph,
                            key=lambda item: item[2])

        parent = []
        rank = []
        for node in range(self.V):
            parent.append(node)
            rank.append(0)
        while e < self.V - 1:
            u, v, w = self.graph[i]
            i = i + 1
            x = self.find(parent, u)
            y = self.find(parent, v)
            if x != y:
                e = e + 1
                result.append([u, v, w])
                self.union(parent, rank, x, y)
        minimumCost = 0
        print ("Edges in the constructed MST")
        for u, v, weight in result:
            minimumCost += weight
            print("%d - %d = %d" % (u, v, weight))
        print("Minimum Spanning Tree" , minimumCost)
g = Graph(5)
```

```
g.addEdge(0, 1, 5)
g.addEdge(0, 2, 7)
g.addEdge(0, 3, 8)
g.addEdge(1, 3, 9)
g.addEdge(2, 3, 2)
g.addEdge(2,4,10)
g.KruskalMST()
```

```
Edges in the constructed MST
2 - 3 = 2
0 - 1 = 5
0 - 2 = 7
2 - 4 = 10
Minimum Spanning Tree 24
```

```
#Prims algorithm
def primsAlgorithm(vertices):
    adjacencyMatrix = [[0 for column in range(vertices)]
                    for row in range(vertices)]
    mstMatrix = [[0 for column in range(vertices)]
                    for row in range(vertices)]
    for i in range(0,vertices):
        for j in range(0+i,vertices):
            adjacencyMatrix[i][j] = int(input('Enter the path weight between the vertices:
            adjacencyMatrix[j][i] = adjacencyMatrix[i][j]
    positiveInf = float('inf')
    selectedVertices = [False for vertex in range(vertices)]
    while(False in selectedVertices):
        minimum = positiveInf
        start = 0
        end = 0

        for i in range(0,vertices):
            if selectedVertices[i]:
                for j in range(0+i,vertices):
                    if (not selectedVertices[j] and adjacencyMatrix[i][j]>0):
                        if adjacencyMatrix[i][j] < minimum:
                            minimum = adjacencyMatrix[i][j]
                            start, end = i, j
        selectedVertices[end] = True



        mstMatrix[start][end] = minimum

        if minimum == positiveInf:
            mstMatrix[start][end] = 0

        mstMatrix[end][start] = mstMatrix[start][end]
    print(mstMatrix)
primsAlgorithm(int(input('Enter the vertices number: ')))
```

```
Enter the vertices number: 6
Enter the path weight between the vertices: (0, 0):  1
Enter the path weight between the vertices: (0, 1):  3
Enter the path weight between the vertices: (0, 2):  2
Enter the path weight between the vertices: (0, 3):  5
Enter the path weight between the vertices: (0, 4):  6
```

```
Enter the path weight between the vertices: (0, 5):  7
Enter the path weight between the vertices: (1, 1):  4
Enter the path weight between the vertices: (1, 2):  89
Enter the path weight between the vertices: (1, 3):  8
Enter the path weight between the vertices: (1, 4):  9
Enter the path weight between the vertices: (1, 5):  11
Enter the path weight between the vertices: (2, 2):  10
Enter the path weight between the vertices: (2, 3):  12
Enter the path weight between the vertices: (2, 4):  15
Enter the path weight between the vertices: (2, 5):  14
Enter the path weight between the vertices: (3, 3):  16
Enter the path weight between the vertices: (3, 4):  18
Enter the path weight between the vertices: (3, 5):  19
Enter the path weight between the vertices: (4, 4):  21
Enter the path weight between the vertices: (4, 5):  23
Enter the path weight between the vertices: (5, 5):  45
[[0, 3, 2, 5, 6, 7], [3, 0, 0, 0, 0, 0], [2, 0, 0, 0, 0, 0], [5, 0, 0, 0, 0, 0]
```

```
#job sequencing with deadlines
def jswd(arr,t):
  n=len(arr)
  for i in range(n):
    for j in range(n-1-i):
          if(arr[j][2]<arr[j+1][2]):
              arr[j],arr[j+1]=arr[j+1],arr[j]
  result = [False]*t
  job = ['-1']*t
  for i in range(len(arr)):
      for j in range(min(t-1,arr[i][1]-1),-1,-1):
        if(result[j] is False):
          result[j] = True
          job[j] = arr[i][0]
          break
  print(job)
arr = [['j1',2,60],['j2',1,100],['j3',3,20],['j4',2,40],['j5',1,20]]
print("Following is the maximum profit sequence of jobs")
jswd(arr,3)
```

```
Following is the maximum profit sequence of jobs
['j2', 'j1', 'j3']
```

```
# All pairs shortest path Floyds 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"),end=" ")
            else:
                print ("%7d\t" % (dist[i][j]),end=' ')
            if j == V-1:
                print ()
graph = [[0, 9, -4,INF],
         [6, 0, INF, 2],
         [INF, 5, 0,   1],
         [INF, INF, 1, 0]
         ]
floydWarshall(graph)
```

```
      0        1                -4                -3
      6        0                 2                 2
     11        5                 0                 1
     12        6                 1                 0
```

```
#Dijkstras Algorithm
import sys
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 \tDistance from Source")
        for node in range(self.V):
            print(node, "\t", dist[node])
    def minDistance(self, dist, sptSet):
        min = sys.maxsize
        for u in range(self.V):
            if dist[u] < min and sptSet[u] == False:
                min = dist[u]
                min_index = u

        return min_index
    def dijkstra(self, src):

        dist = [sys.maxsize] * self.V
        dist[src] = 0
        sptSet = [False] * self.V

        for cout in range(self.V):
            x = self.minDistance(dist, sptSet)
            sptSet[x] = True
            for y in range(self.V):
                if self.graph[x][y] > 0 and sptSet[y] == False and \
                dist[y] > dist[x] + self.graph[x][y]:
                        dist[y] = dist[x] + self.graph[x][y]

        self.printSolution(dist)
g = Graph(9)
g.graph = [[0, 4, 0, 0, 0, 0, 0, 10, 0],
        [4, 0, 8, 0, 0, 0, 0, 15, 0],
        [0, 8, 0, 7, 0, 4, 0, 0, 2],
        [0, 0, 7, 0, 8, 14, 0, 0, 0],
        [0, 0, 0, 9, 0, 11, 0, 0, 0],
        [0, 0, 4, 13, 10, 0, 2, 0, 0],
        [0, 0, 0, 0, 0, 2, 0, 1, 6],
```

```
        [8, 12, 0, 0, 0, 0, 1, 0, 7],
        [0, 0, 2, 0, 0, 0, 6, 8 , 0]
        ];

g.dijkstra(0);
```

```
      Vertex  Distance from Source
      0        0
      1        4
      2        12
      3        19
      4        23
      5        13
      6        11
      7        10
      8        14
```

```
# (SUM OF SUBSETS) Find a subset of a given set S = {s1,s2,.....,sn} of n positive integer
def printAllSubsetsRec(arr, n, v, sum) :
    if (sum == 0) :
        for value in v :
            print(value, end=" ")
        print()
        return
    #print("No sub sets found")
    if (n == 0):
        return
    printAllSubsetsRec(arr, n - 1, v, sum)
    v1 = [] + v
    v1.append(arr[n - 1])
    printAllSubsetsRec(arr, n - 1, v1, sum - arr[n - 1])
def printAllSubsets(arr, n, sum):
    v = []
    printAllSubsetsRec(arr, n, v, sum)
arr = [1,2,5,6,8]
sum = 9
n = len(arr)
printAllSubsets(arr, n, sum)
```

```
      6 2 1
      8 1
```

```
#BFS
g = {
   0 : [2,3],
   1 : [1, 4,3],
   2 : [0,1,3],
   3 : [1,2,4],
   4 : [4,3]
}

vN = []
q = []

def bfs(visitedList: list, g: dict, node):
    visitedList.append(node)
    q.append(node)
```

```
    while q:
        m = q.pop(0)
        print (m, end = "\t")
        for adjacent in g[m]:
            if adjacent not in visitedList:
                visitedList.append(adjacent)
                q.append(adjacent)

bfs(vN, g, 0)
```

```
0       2       3       1       4
```

```
#DFS
class Gra:
    def __init__(self, edges, n):
        self.adjList = [[] for i in range(n)]
        for (src, dest) in edges:
            self.adjList[src].append(dest)
def DFS(graph, v, discovered, arrival, isSC, t):
    if not isSC:
        return 0, isSC, t
    t = t + 1
    arrival[v] = t
    discovered[v] = True
    arr = arrival[v]
    for w in graph.adjList[v]:
        if not discovered[w]:
            _arr, isSC, t = DFS(graph, w, discovered, arrival, isSC, t)
            arr = min(arr, _arr)
        else:
            arr = min(arr, arrival[w])
    if v and arr == arrival[v]:
        isSC = False
    return arr, isSC, t
def isc(graph, n):
    discovered = [False] * n
    arrival = [0] * n
    isSC = True
    t = -1
    isSC = DFS(graph, 0, discovered, arrival, isSC, t)[1]
    for i in range(n):
        if not discovered[i]:
            isSC = False
    return isSC
if __name__ == '__main__':
    edges = [(0, 1), (0,4), (1, 3), (1,2), (1,4), (4, 0), (4, 1), (4, 3)]
    n = 5
    graph = Gra(edges, n)
    if isc(graph, n):
        print('The graph is connected')
    else:
        print('The graph is not connected')
```

```
The graph is not connected
```

```
def isSafe(graph, color):
    for i in range(5):
        for j in range(i + 1, 4):
```

```
            if (graph[i][j] and color[j] == color[i]):
                return False
    return True
def gc(graph, m, i, color):
    if (i == 5):
        if (isSafe(graph, color)):
            ps(color)
            return True
        return False
    for j in range(1, m + 1):
        color[i] = j
        if (gc(graph, m, i + 1, color)):
            return True
        color[i] = 0
    return False
def ps(color):
    print("Solution Exists:" " Following are the assigned colors ")
    for i in range(5):
        print(color[i],end=" ")
if __name__ == '__main__':
    graph = [
        [ 0, 1, 1, 1 ],
        [ 1, 0, 1, 0 ],
        [ 1, 1, 0, 1 ],
        [ 1, 0, 1, 0 ],
        [ 1, 0, 1, 1 ]
    ]
    m = 3
    color = [0 for i in range(5)]
    if (not gc(graph, m, 0, color)):
        print ("Solution does not exist")
```

```
Solution Exists: Following are the assigned colors
1 2 3 2 1
```

```
from sympy import symbols, Eq, solve
x, y, z = symbols("x y z")
eq_1 = Eq((2*x - 4*y + 6*z), 10)
eq_2 = Eq((4*x + 2*y + 6*z), 30)
eq_3 = Eq((4*x + 2*y - 10*z), 50)
print("Equation 1:", eq_1)
print("Equation 2:", eq_2)
print("Equation 3:", eq_3)
sol = solve((eq_1, eq_2, eq_3), (x, y, z))
print("Solution:", sol)
```

```
Equation 1: Eq(2*x - 4*y + 6*z, 10)
Equation 2: Eq(4*x + 2*y + 6*z, 30)
Equation 3: Eq(4*x + 2*y - 10*z, 50)
Solution: {x: 37/4, y: 1/4, z: -5/4}
```

```
from sys import maxsize
from itertools import permutations
V = 4
def tsp(graph, s):
    vertex = []
    for i in range(V):
```

```
        if i != s:
            vertex.append(i)

    mp= maxsize
    np=permutations(vertex)
    for i in np:

        current = 0
        k = s
        for j in i:
            current += graph[k][j]
            k = j
        current += graph[k][s]

        mp = min(mp, current)

    return mp



if __name__ == "__main__":
    graph = [[0, 10, 15, 20],
             [10, 0, 9, 10],
             [6, 13, 0, 12],
             [8, 8, 9, 0]]
    s = 0
    print("Best solution is",tsp(graph, s))
```

```
Best solution is 35
```