PROGRAM:

```
def find_max_min(arr):
   n=len(arr)
   if n==1:
      return arr[0],arr[0]
   elif n==2:
         return (arr[0],arr[1]) if arr[0]<arr[1] else(arr[1],arr[0])

   else:
      mid=n//2
      left_min,left_max=find_max_min(arr[:mid])
      right_min,right_max=find_max_min(arr[mid:])
      return min(left_min,right_min),max(left_max,right_max)
arr=[3,5,1,78,56,94,23,45]
mini,maxi=find_max_min(arr)
print("The max no is:",maxi)
print("The min no is:",mini)
```

PROGRAM:

```
import random
from timeit import default_timer as timer
import matplotlib.pyplot as plt

def mergeSort(array):
    if len(array) > 1:
        r = len(array)//2
        L = array[:r]
        M = array[r:]
        mergeSort(L)
        mergeSort(M)
        i = j = k = 0
        while i < len(L) and j < len(M):
            if L[i] < M[j]:
                array[k] = L[i]
                i += 1
            else:
                array[k] = M[j]
                j += 1
            k += 1
        while i < len(L):
            array[k] = L[i]
            i += 1
            k += 1
        while j < len(M):
            array[k] = M[j]
            j += 1
            k += 1
x=[]
y=[]
for i in range(3):

 # Generate a list of random integers
    n=int(input("\nenter the value of n:"))
    x.append(n)
    arr = [random.randint(0, 1000) for _ in range(n)]
    print("\nthe array elements are",arr)
    start_time = timer()
    ind=mergeSort(arr)
    end_time = timer()
    print("array elements are ", arr)
    elapsed_time = end_time - start_time
    y.append(elapsed_time)
    print("time taken=", elapsed_time)

# Plot the results
plt.plot(x,y)
plt.title('Time Taken for merge sort')
plt.xlabel('n')
plt.ylabel('Time (seconds)')
plt.show()
```

PROGRAM:

```
import random
from timeit import default_timer as timer
import matplotlib.pyplot as plt

def partition(array, low, high):
  pivot = array[high]
  i = low - 1
  for j in range(low, high):
    if array[j] <= pivot:
      i = i + 1
      (array[i], array[j]) = (array[j], array[i])
  (array[i + 1], array[high]) = (array[high], array[i + 1])
  return i + 1

def quickSort(array, low, high):
  if low < high:
    pi = partition(array, low, high)
    quickSort(array, low, pi - 1)
    quickSort(array, pi + 1, high)
x=[]
y=[]
for i in range(3):

  # Generate a list of random integers
    n=int(input("\nenter the value of n:"))
    x.append(n)
    arr = [random.randint(0, 1000) for _ in range(n)]
    print("\nthe array elements are",arr)
    start_time = timer()
    ind=quickSort(arr,low=0,high=n-1)
    end_time = timer()
    print("array elements are ", arr)
    elapsed_time = end_time - start_time
    y.append(elapsed_time)
    print("time taken=", elapsed_time)

# Plot the results
plt.plot(x,y)
plt.title('Time Taken for quick sort')
plt.xlabel('n')
plt.ylabel('Time (seconds)')
plt.show()
```

PROGRAM:

```
global N
N = 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, N, 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]
      ]
  if solveNQUtil(board, 0) == False:
    print ("Solution does not exist")
    return False
  printSolution(board)
  return True
solveNQ()
```

PROGRAM:

```
import numpy as np
def nearest_neighbor(matrix):
    n = matrix.shape[0]
    current_city = np.random.randint(n)
    visited_cities = [current_city]
    total_distance = 0
    while len(visited_cities) < n:
        nearest_city = np.argmin([matrix[current_city][i] for i in range(n) if i not in visited_cities])
        visited_cities.append(nearest_city)
        total_distance += matrix[current_city][nearest_city]
        current_city = nearest_city
    total_distance += matrix[visited_cities[-1]][visited_cities[0]]
    visited_cities.append(visited_cities[0])
    return visited_cities, total_distance
def calculate_distance_matrix(points):
    n = len(points)
    dist_matrix = np.zeros((n, n))
    for i in range(n):
        for j in range(n):
            dist_matrix[i][j] = np.sqrt((points[i][0] - points[j][0]) ** 2 + (points[i][1] - points[j][1]) ** 2)
    return dist_matrix
points = [(0, 0), (1, 2), (3, 1), (2, 3)]
matrix = calculate_distance_matrix(points)
route, approx_distance = nearest_neighbor(matrix)
from itertools import permutations
perm = permutations(range(len(points)))
optimal_distance = float('inf')
for p in perm:
    distance = 0
    for i in range(len(p) - 1):
        distance += matrix[p[i]][p[i + 1]]
    distance += matrix[p[-1]][p[0]]
```

```
    if distance < optimal_distance:
        optimal_distance = distance
print("Points:", points)
print("Approximation Route:", route)
print("Approximation Distance:", approx_distance)
print("Optimal Distance:", optimal_distance)
print("Error Approximation:", (approx_distance - optimal_distance) / optimal_distance)
```

PROGRAM:

```
import random
def kthSmallest(arr, l, r, k):
    if (k > 0 and k <= r - l + 1):
        pos = randomPartition(arr, l, r)
        if (pos - l == k - 1):
            return arr[pos]
        if (pos - l > k - 1):
            return kthSmallest(arr, l, pos - 1, k)
        return kthSmallest(arr, pos + 1, r,
                            k - pos + l - 1)
    return 999999999999
def swap(arr, a, b):
    temp = arr[a]
    arr[a] = arr[b]
    arr[b] = temp
def partition(arr, l, r):
    x = arr[r]
    i = l
    for j in range(l, r):
        if (arr[j] <= x):
            swap(arr, i, j)
            i += 1
    swap(arr, i, r)
    return i
def randomPartition(arr, l, r):
    n = r - l + 1
    pivot = int(random.random() * n)
    swap(arr, l + pivot, r)
    return partition(arr, l, r)
if __name__ == '__main__':
    arr = [12, 3, 5, 7, 4, 19, 26]
    n = len(arr)
    k = 3
    print("K'th smallest element is",kthSmallest(arr, 0, n - 1, k))
```