DESING AND ANALYSIS OF ALGORITHMS

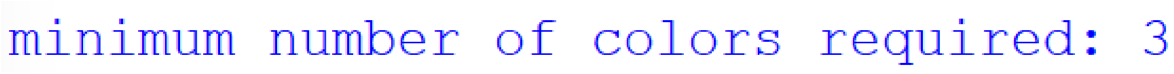
HACKATHON

1.GRAPH COLOURING

PROGRAM:

```
def graph_coloring(adj_list):
    colors = {}
    max_color = 0

    for node in adj_list:
        neighbor_colors = set(colors.get(nei, 0) for nei in adj_list[node])
        color = 1
        while color in neighbor_colors:
            color += 1
        colors[node] = color
        max_color = max(max_color, color)

    return max_color

adj_list = {
    0: [1, 2, 3],
    1: [0, 2],
    2: [1, 3, 0],
    3: [2, 0]
}

max_regions_colored = graph_coloring(adj_list)
print("minimum number of colors required:",max_regions_colored)
```

OUTPUT:



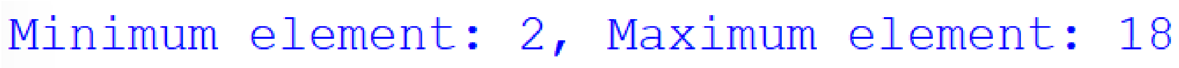
```
minimum number of colors required: 3
```

2.FIND THE MAXIMUM AND MINIMUM VALUES IN SORTED ARRAY

PROGRAM:

```
def find_min_max(arr):

 if not arr:
  return None
 minimum = maximum = arr[0]
 for element in arr:
  minimum = min(minimum, element)
  maximum = max(maximum, element)
 return minimum, maximum

arr = [2,4,6,8,10,12,14,18]
min_value, max_value = find_min_max(arr)
print(f"Minimum element: {min_value}, Maximum element: {max_value}")
```

OUTPUT:



```
Minimum element: 2, Maximum element: 18
```

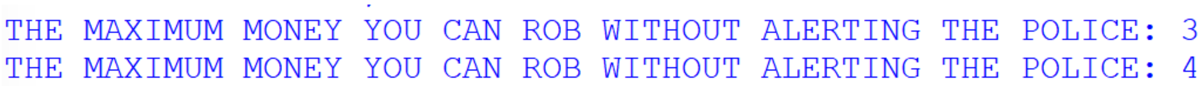
3.PROFESSIONAL ROBBER PLANNING

TO FIND THE MAXIMUM MONEY YOU CAN ROB WITHOUT ALERTING THE POLICE

PROGRAM:

```
def rob(nums):
  if not nums:
    return 0
  if len(nums) <= 2:
    return max(nums)

  def rob_helper(nums):
    dp = [0] * len(nums)
    dp[0] = nums[0]
    dp[1] = max(nums[0], nums[1])

    for i in range(2, len(nums)):
            dp[i] = max(dp[i-1], dp[i-2] + nums[i])

        return dp[-1]

    return max(rob_helper(nums[1:]), rob_helper(nums[:-1]))

print("THE MAXIMUM MONEY YOU CAN ROB WITHOUT ALERTING THE POLICE:",rob([2, 3, 2]))
print("THE MAXIMUM MONEY YOU CAN ROB WITHOUT ALERTING THE POLICE:",rob([1, 2, 3, 1]))
```

OUTPUT:



```
THE MAXIMUM MONEY YOU CAN ROB WITHOUT ALERTING THE POLICE: 3
THE MAXIMUM MONEY YOU CAN ROB WITHOUT ALERTING THE POLICE: 4
```

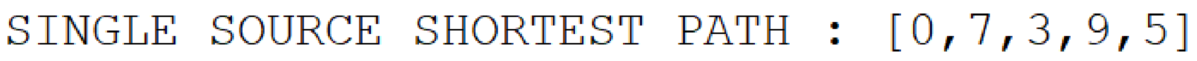
4.SINGLE SOURCE SHORTEST PATHS : DIJKSTRA’S ALGORITHM

PROGRAM:

```
import sys

def dijkstra(graph, source):
    n = len(graph)
    dist = [sys.maxsize] * n
    dist[source] = 0
    visited = [False] * n

    for _ in range(n):
        u = min_distance(dist, visited)
        visited[u] = True

        for v in range(n):
            if not visited[v] and graph[u][v] != sys.maxsize and dist[u] + graph[u][v] < dist[v]:
                dist[v] = dist[u] + graph[u][v]

    return dist
```

```
def min_distance(dist, visited):
    min_dist = sys.maxsize
    min_index = -1

    for v in range(len(dist)):
        if not visited[v] and dist[v] < min_dist:
            min_dist = dist[v]
            min_index = v

    return min_index

graph = [
    [0, 10, 3, sys.maxsize, sys.maxsize],
    [sys.maxsize, 0, 1, 2, sys.maxsize],
    [sys.maxsize, 4, 0, 8, 21],
    [sys.maxsize, sys.maxsize, sys.maxsize, 0, 6],
    [sys.maxsize, sys.maxsize, sys.maxsize, sys.maxsize, 0]
]
source = 0
result = dijkstra(graph, source)
print(result)

graph = [
    [0, 5, sys.maxsize, 10],
    [sys.maxsize, 0, 3, sys.maxsize],
    [sys.maxsize, sys.maxsize, 0, 1],
    [sys.maxsize, sys.maxsize, sys.maxsize, 0]
]
source = 3
result = dijkstra(graph, source)
print(result)
```

OUTPUT:



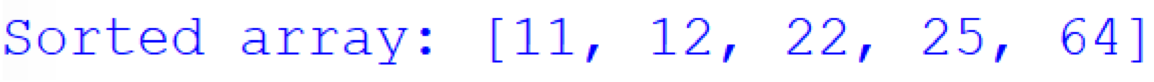
```
SINGLE SOURCE SHORTEST PATH : [0,7,3,9,5]
```

5.SELECTION SORT ALGORITHM

PROGRAM:

```
def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        min_idx = i
        for j in range(i+1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
    return arr


arr = [64, 25, 12, 22, 11]
sorted_arr = selection_sort(arr)
print("Sorted array:", sorted_arr)
```

OUTPUT:



```
Sorted array: [11, 12, 22, 25, 64]
```

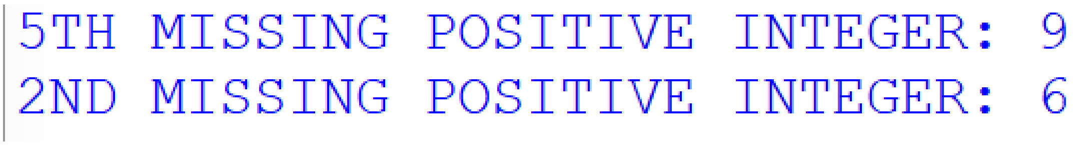
6.SEQUENTIAL SEARCH

PROGRAM:

```
def findKthPositive(arr, k):
    missing = []
    i = 1
    while len(missing) < k:
        if i not in arr:
            missing.append(i)
        i += 1
    return missing[-1]



arr1 = [2, 3, 4, 7, 11]
```

```
k1 = 5
output1 = findKthPositive(arr1, k1)
print("5TH MISSING POSITIVE INTEGER:",output1)

arr2 = [1, 2, 3, 4, 14, 15]
k2 = 2
output2 = findKthPositive(arr2, k2)
print("2ND MISSING POSITIVE INTEGER:",output2)
```

OUTPUT:



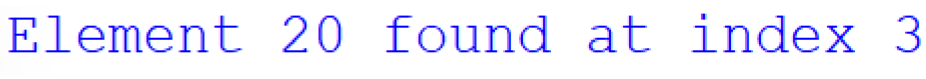
```
5TH MISSING POSITIVE INTEGER: 9
2ND MISSING POSITIVE INTEGER: 6
```

7.BINARY SEARCH

PROGRAM:

```
def binary_search(arr, element):
  low = 0
  high = len(arr) - 1
  while low <= high:
    mid = (low + high) // 2
    if arr[mid] == element:
      return mid
    elif arr[mid] < element:
      low = mid + 1
    else:
      high = mid - 1
  return -1
arr = [5,10,15,20,25,30,35,40,45]
element_to_find = 20
index = binary_search(arr, element_to_find)
if index != -1:
  print(f"Element {element_to_find} found at index {index}")
else:
  print(f"Element {element_to_find} not found in the array")
```

OUTPUT:



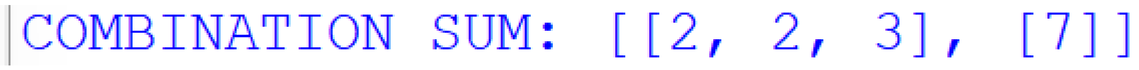
```
Element 20 found at index 3
```

8.COMBINATION SUM

PROGRAM:

```
def combinationSum(candidates, target):
    res = []
    def dfs(path, target, start):
        if target == 0:
            res.append(path)
            return
        for i in range(start, len(candidates)):
            if candidates[i] > target:
                continue
            dfs(path + [candidates[i]], target - candidates[i], i)
    dfs([], target, 0)
    return res
candidates = [2, 3, 6, 7]
target = 7
print("COMBINATION SUM:",combinationSum(candidates, target))
```

OUTPUT:



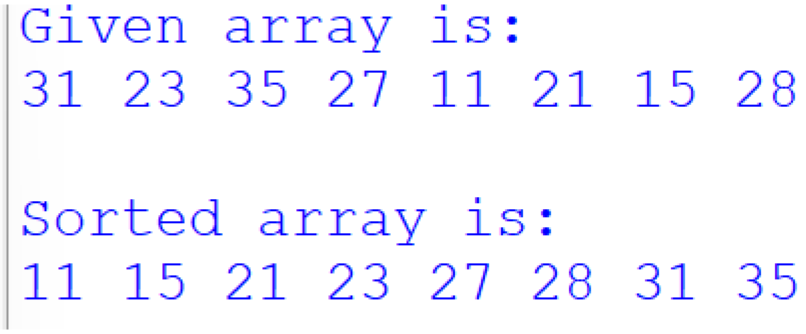
```
COMBINATION SUM: [[2, 2, 3], [7]]
```

9.MERGE SORT

PROGRAM:

```
def merge(arr, l, m, r):
    n1 = m - l + 1
    n2 = r - m
    L = [0] * n1
    R = [0] * n2

    for i in range(n1):
        L[i] = arr[l + i]
    for j in range(n2):
        R[j] = arr[m + 1 + j]

    i = 0
    j = 0
    k = l

    while i < n1 and j < n2:
        if L[i] <= R[j]:
            arr[k] = L[i]
            i += 1
        else:
            arr[k] = R[j]
            j += 1
        k += 1

    while i < n1:
        arr[k] = L[i]
        i += 1
        k += 1

    while j < n2:
        arr[k] = R[j]
        j += 1
        k += 1

def mergeSort(arr, l, r):
    if l < r:
        m = l + (r - l) // 2
        mergeSort(arr, l, m)
        mergeSort(arr, m + 1, r)
        merge(arr, l, m, r)


arr = [31,23,35,27,11,21,15,28]
n = len(arr)


print("Given array is:")
for i in range(n):
    print("%d" % arr[i], end=" ")


mergeSort(arr, 0, n - 1)


print("\n\nSorted array is:")
for i in range(n):
    print("%d" % arr[i], end=" ")
```

OUTPUT:



```
Given array is:
31 23 35 27 11 21 15 28

Sorted array is:
11 15 21 23 27 28 31 35
```

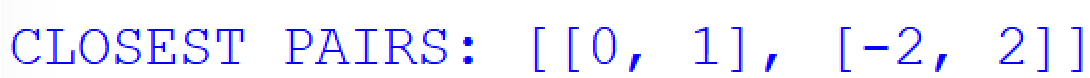
10.CLOSEST PAIR OF POINTS (DIVIDE AND CONQUER)

PROGRAM:

```
import math


def distance(point):
    return math.sqrt(point[0]**2 + point[1]**2)


def kClosest(points, k):
    points.sort(key=distance)
    return points[:k]
```

```
input_points = [[1, 3], [-2, 2], [5, 8], [0, 1]]

k = 2

output_points = kClosest(input_points, k)

print("\n")

print(output_points)
```

OUTPUT:



```
CLOSEST PAIRS: [[0, 1], [-2, 2]]
```