**DAA- HAKTHAN**

1. **Minimum and maximum**

Arr=[2,4,6,8,10,12,14,18]

min = arr[0]

max = arr[0]

for num in arr:

if num < min:

min = num

if num > max:

max = num

print("Minimum number:",min)

print("Maximum number:", max)

1. **single source shortest path: dijkstra algorithm**

graph = {

'A': {'B': 6, 'C': 1},

'B': {'A': 6, 'C': 2, 'D': 2},

'C': {'A': 1, 'B': 2, 'D': 1},

'D': {'B': 2, 'C': 1}

}

distances = {node: float('infinity') for node in graph}

distances['A'] = 0

unvisited = graph.keys()

while unvisited:

current\_node = min(unvisited, key=lambda node: distances[node])

unvisited.remove(current\_node)

for neighbor, weight in graph[current\_node].items():

new\_distance = distances[current\_node] + weight

if new\_distance < distances[neighbor]:

distances[neighbor] = new\_distance

distances

1. **Binary search**

arr = [2, 3, 4, 10, 40]

x = 10

low = 0

high = len(arr) - 1

mid = 0

while low <= high:

mid = (high + low) // 2

if arr[mid] < x:

low = mid + 1

elif arr[mid] > x:

high = mid - 1

else:

break

if low > high:

print("Element not found")

else:

print("Element found at index", mid)

1. **combination sum 1**

candidates = [2, 3, 6, 7]

target = 7

results = []

stack = [(0, [], target)]

while stack:

index, path, target = stack.pop()

if target == 0:

results.append(path)

for i in range(index, len(candidates)):

if candidates[i] <= target:

stack.append((i, path + [candidates[i]], target - candidates[i]))

print(results)

1. **merge sort without using function**

N = 8

a = [31, 23, 35, 27, 11, 21, 15, 28]

def merge\_sort(arr):

if len(arr) > 1:

mid = len(arr) // 2

L = arr[:mid]

R = arr[mid:]

merge\_sort(L)

merge\_sort(R)

i = j = k = 0

while i < len(L) and j < len(R):

if L[i] < R[j]:

arr[k] = L[i]

i += 1

else:

arr[k] = R[j]

j += 1

k += 1

while i < len(L):

arr[k] = L[i]

i += 1

k += 1

while j < len(R):

arr[k] = R[j]

j += 1

k += 1

merge\_sort(a)

print(a)

1. **sequential search without usinh function.**

Arr = [2, 3, 4, 7, 11]

k = 5

output = -1

for i in range(len(Arr)):

if Arr[i] > k:

output = i \* 2

break

print(output)

1. **selection sort algorithm without using function.**

input = [5, 2, 9, 1, 5, 6]

for i in range(len(input)):

min = i

for j in range(i+1, len(input)):

if input[min] > input[j]:

min = j

input[i], input[min] = input[min], input\_list[i]

print(input)

1. **closest pairs of points(divide and conquer method) without using function**

points = [[1,3],[-2,2],[5,8],[0,1]]

k = 2

points.sort(key=lambda x: x[0]\*\*2 + x[1]\*\*2)

output = points[:k]

print(output)

**Single source shortest path:dijkstra algorithm**

Example:

N=5

Graph=[[0,10,3,infinity,infinity],[infinity,0,1,2,infinity],[infinity,4,0,8,2],[infinity,infinity,infinity,0,7],[infinity,infinity,infinity,9,0]]

Source=0

Output:[0,7,3,9,5]

N = 5

Graph = [[0, 10, 3, float('inf'), float('inf')],

[float('inf'), 0, 1, 2, float('inf')],

[float('inf'), 4, 0, 8, 2],

[float('inf'), float('inf'), float('inf'), 0, 7],

[float('inf'), float('inf'), float('inf'), 9, 0]]

Source = 0

def dijkstra(graph, src):

dist = [float('inf')] \* len(graph)

dist[src] = 0

visited = [False] \* len(graph)

for \_ in range(len(graph)):

u = min\_distance(dist, visited)

visited[u] = True

for v in range(len(graph)):

if not visited[v] and graph[u][v] != float('inf') and dist[u] + graph[u][v] < dist[v]:

dist[v] = dist[u] + graph[u][v]

return dist

def min\_distance(dist, visited):

min\_dist = float('inf')

min\_index = -1

for v in range(len(dist)):

if dist[v] < min\_dist and not visited[v]:

min\_dist = dist[v]

min\_index = v

return min\_index

output = dijkstra(Graph, Source)

print(output)