DAA- HACKTHON :-

6.sequential search :-

```
def find_kth_missing(arr, k):
    missing_count = 0
    current_number = 1
    index = 0
    n = len(arr)

    while missing_count < k:
        if index < n and arr[index] == current_number:
            index += 1
        else:
            missing_count += 1
            if missing_count == k:
                return current_number
        current_number += 1

    return -1

arr = [2, 3, 4, 7, 11]
k = 5
missing_number = find_kth_missing(arr, k)
print(f'The {k}-th missing number is: {missing_number}')
```

7. Binary search:-

```
def binary_search(arr, target):
    left, right = 0, len(arr) - 1
    while left <= right:
        mid = (left + right) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
    return -1

def main():
    arr = [5,10,15,20,25,30,35,40,45]
    target = int(input("Enter the number to search for: "))

    result = binary_search(arr, target)

    if result != -1:
        print(f"Element {target} found at index {result}.")
    else:
        print(f"Element {target} not found in the array.")

if __name__ == "__main__":
    main()
```

8.combination sum:-

```
def combination_sum(candidates, target):
    results = []

    def backtrack(start, target, path):
        if target < 0:
            return
        if target == 0:
            results.append(path)
            return

        for i in range(start, len(candidates)):
            backtrack(i, target - candidates[i], path + [candidates[i]])

    candidates.sort()
    backtrack(0, target, [])

    return results

candidates = [2, 3, 6, 7]
target = 7
print(combination_sum(candidates, target))
```

9.merge sort:-

```
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
```

```
arr = [5,2,9,1,5,6]
merge_sort(arr)
print("Sorted array is:", arr)
```

10.Closest pair of points:-

```
import heapq

def k_closest_points(points, k):
    def squared_distance(point):
        return point[0] ** 2 + point[1] ** 2

    max_heap = []

    for point in points:
        dist = squared_distance(point)
        heapq.heappush(max_heap, (-dist, point))
        if len(max_heap) > k:
            heapq.heappop(max_heap)

    return [point for (_, point) in max_heap]

points = [[1,3],[-2,2],[5,8],[0,1]]
k = 2
print("The closest points are:", k_closest_points(points, k))
```

1.GRAPH COLOURING:

```
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(max_regions_colored)
```

2.MAXIMUM AND MINIMUM VALUE:

```
array1 = [2, 4, 6, 8, 10, 12, 14, 18]
array2 = [11,13,15,17,19,21,23,35,37]
min_value = min(array1)
max_value = max(array1)
min_value2 = min(array2)
max_value2 = max(array2)
```

```
print("Input Array:", array1)
print("Minimum Value:", min_value)
print("Maximum Value:", max_value)
print("Input Array:", array2)
print("Minimum Value:", min_value2)
print("Maximum Value:", max_value2)
```

3.ROBBERY:

```
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]))

# Test the function with example inputs
```

```
print(rob([2, 3, 2]))
print(rob([1, 2, 3, 1]))
```

4.DIJAKRASTRA’S:

```
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)
```

5.SELECTION SORT:

```
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

random_array = [52, 9, 1, 5, 6]
sorted_random_array = selection_sort(random_array)
print(sorted_random_array)



reverse_sorted_array = [12, 8, 6, 4, 2]
sorted_reverse_array = selection_sort(reverse_sorted_array)
print(sorted_reverse_array)

already_sorted_array = [1, 2, 3, 4, 5]
sorted_already_sorted_array = selection_sort(already_sorted_array)
print(sorted_already_sorted_array)
```