LAB DAY-7(12-06-2024)

1. Selection Sort

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

arr = [64, 25, 12, 22, 11]
print("Selection Sort:", selection (arr))
```

OUTPUT:

```
Selection Sort: [11, 12, 22, 25, 64]
```

2. Bubble Sort

```
def bubble (arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr

arr = [64, 34, 25, 12, 22, 11, 90]
print("Bubble Sort:", bubble(arr))
```

OUTPUT:

```
Bubble Sort: [11, 12, 22, 25, 34, 64, 90]
```

3. Insertion Sort

```
def insertion(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
    return arr

arr = [12, 11, 13, 5, 6]
print("Insertion Sort:", insertion (arr))
```

OUTPUT:

```
Insertion Sort: [5, 6, 11, 12, 13]
```

4. Sequential Search

```
def sequential (arr, x):
    for i in range(len(arr)):
        if arr[i] == x:
            return i
    return -1

arr = [2, 3, 4, 10, 40]
x = 10
print("Sequential Search:", sequential(arr, x))
```

```
OUTPUT:
Sequential Search: 3
```

5. Brute-Force String Matching

```
def stringmatch(text, pattern):
    n = len(text)
    m = len(pattern)
    for i in range(n - m + 1):
        j = 0
        while j < m and text[i + j] == pattern[j]:
            j += 1
        if j == m:
            return i
    return -1

text = "ABAAABCDBBABCDDEBCABC"
pattern = "ABC"
print("Brute-Force String Matching:", stringmatch(text, pattern))
```

```
OUTPUT:
Brute-Force String Matching: 4
```

6. Closest-Pair (Naive approach)

```
import math

def distance(p1, p2):

    return math.sqrt((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2)

def closestpair(points):
    mindist = float('inf')
    pair = None
    n = len(points)
    for i in range(n):
        for j in range(i + 1, n):
            dist = distance(points[i], points[j])
            if dist < mindist:
                mindist = dist
                pair = (points[i], points[j])
    return pair, mindist

points = [(2, 3), (12, 30), (40, 50), (5, 1), (12, 10), (3, 4)]
print("Closest-Pair:", closestpair(points))
```

```
OUTPUT:
Closest-Pair: (((2, 3), (3, 4)), 1.4142135623730951)
```

7. Convex-Hull Problem

```
def orientation(p, q, r):
    val = (q[1] - p[1]) * (r[0] - q[0]) - (q[0] - p[0]) * (r[1] - q[1])
    if val == 0:
        return 0
    elif val > 0:
        return 1
    else:
        return 2

def convexhull(points):
    n = len(points)
    if n < 3:
        return points

    points = sorted(points)
    hull = []

    for point in points:
        while len(hull) > 1 and orientation(hull[-2], hull[-1], point) != 2:
            hull.pop()
        hull.append(point)

    lowerhullsize = len(hull)

    for point in reversed(points):
        while len(hull) > lowerhullsize and orientation(hull[-2], hull[-1], point) != 2:
            hull.pop()
        hull.append(point)

    hull.pop()
    return hull

points = [(0, 3), (2, 3), (1, 1), (2, 1), (3, 0), (0, 0), (3, 3)]
print("Convex Hull:", convexhull(points))
```

```
OUTPUT:
Convex Hull: [(0, 0), (3, 0), (3, 3), (0, 3)]
```

8. Exhaustive Search

```
def exhaustive (arr, target):
    n = len(arr)
    for i in range(n):
        if arr[i] == target:
            return i
    return -1

arr = [1, 2, 3, 4, 5]
target = 3
print("Exhaustive Search:", exhaustive(arr, target))
```

```
OUTPUT: Exhaustive Search: 2
```