LAB-7:

1. Selection Sort

```
def selectionSort(array, size):
        for ind in range(size):
            min_index = ind
            for j in range(ind + 1, size):
                if array[j] < array[min_index]:
                    min_index = j
            (array[ind], array[min_index]) = (array[min_index], array[ind])
arr = [-2, 45, 0, 11, -9,88,-97,-202,747]
size = len(arr)
selectionSort(arr, size)
print('The array after sorting in Ascending Order by selection sort is:')
print(arr)
```

OUTPUT:

```
>>>
= RESTART: C:/Users/bored/AppData/Local/Programs/Python/Python312/selection sort
.py
The array after sorting in Ascending Order by selection sort is:
[-202, -97, -9, -2, 0, 11, 45, 88, 747]
>>>
```

2. Bubble Sort

```
def bubbleSort(arr):
    n = len(arr)
    for i in range(n-1):
        swapped = False
        for j in range(0, n-i-1):
            if arr[j] > arr[j + 1]:
                swapped = True
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
        if not swapped:
            return
arr = [64, 34, 25, 12, 22, 11, 90]
bubbleSort(arr)
print("Sorted array is:")
for i in range(len(arr)):
    print("% d" % arr[i], end=" ")
```

OUTPUT:

```
>>>
= RESTART: C:/Users/bored/AppData/Local/Programs/Python/Python312/bubble sort.py
Sorted array is:
 11  12  22  25  34  64  90
>>>
```

3. Insertion sort

```
def insertionSort(arr):
    n = len(arr)
    if n <= 1:
        return
    for i in range(1, n):
        key = arr[i]
        j = i-1
        while j >= 0 and key < arr[j]:
            arr[j+1] = arr[j]
            j -= 1
        arr[j+1] = key
arr = [12, 11, 13, 5, 6]
insertionSort(arr)
print(arr)
```

OUTPUT:

```
>>>
= RESTART: C:/Users/bored/AppData/Local/Programs/Python/Python312/insertion sort
.py
[5, 6, 11, 12, 13]
>>>
```

4. Sequential Search

```
def Sequential_Search(dlist, item):
    pos = 0
    found = False
    while pos < len(dlist) and not found:
        if dlist[pos] == item:
            found = True
        else:
            pos = pos + 1
    return found, pos
print(Sequential_Search([11,23,58,31,56,77,43,12,65,19],31))
```

OUTPUT:

```
>>>
= RESTART: C:/Users/bored/AppData/Local/Programs/Python/Python312/sequential sea
rch.py
(True, 3)
>>>
```

5.Brute-Force String Matching

```
def brute_force_string_match(str1, str2):
    for i in range(len(str1)):
        for j in range(len(str2)):
```

```
            if str1[i] == str2[j]:
                return True
    return False
str1 = "hello"
str2 = "world"
result = brute_force_string_match(str1, str2)
print(result)
```

OUTPUT:

```
>>>
= RESTART: C:/Users/bored/AppData/Local/Programs/Python/Python312/brute force st
ring matching.py
True
>>>
```

6.Closest-Pair

```
import math
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
def compareX(a, b):
    p1 = a
    p2 = b
    return (p1.x - p2.x)
def compareY(a, b):
    p1 = a
    p2 = b
    return (p1.y - p2.y)
def dist(p1, p2):
    return math.sqrt((p1.x - p2.x)*(p1.x - p2.x) + (p1.y - p2.y)*(p1.y - p2.y))
def bruteForce(P, n):
    min_dist = float("inf")
    for i in range(n):
        for j in range(i+1, n):
```

```
            if dist(P[i], P[j]) < min_dist:
                min_dist = dist(P[i], P[j])
    return min_dist
def min(x, y):
    return x if x < y else y
def stripClosest(strip, size, d):
    min_dist = d
    strip = sorted(strip, key=lambda point: point.y)
    for i in range(size):
        for j in range(i+1, size):
            if (strip[j].y - strip[i].y) >= min_dist:
                break
            if dist(strip[i], strip[j]) < min_dist:
                min_dist = dist(strip[i], strip[j])
    return min_dist
def closestUtil(P, n):
    if n <= 3:
        return bruteForce(P, n)
    mid = n//2
    midPoint = P[mid]
    dl = closestUtil(P, mid)
    dr = closestUtil(P[mid:], n - mid)
    d = min(dl, dr)
    strip = []
    for i in range(n):
        if abs(P[i].x - midPoint.x) < d:
            strip.append(P[i])
    return min(d, stripClosest(strip, len(strip), d))
def closest(P, n):
    P = sorted(P, key=lambda point: point.x)
    return closestUtil(P, n)
```

```
if __name__ == "__main__":
    P = [Point(x=2, y=3), Point(x=12, y=30),
         Point(x=40, y=50), Point(x=5, y=1), Point(x=12, y=10), Point(x=3, y=4)]
    n = len(P)
    print("The smallest distance is", closest(P, n))
```

OUTPUT:

```
>>>
    = RESTART: C:/Users/bored/AppData/Local/Programs/Python/Python312/closest pair.p
    y
    The smallest distance is 1.4142135623730951
>>>
```

7.Convex-Hull Problems

```
from functools import cmp_to_key
mid = [0, 0]
def quad(p):
    if p[0] >= 0 and p[1] >= 0:
        return 1
    if p[0] <= 0 and p[1] >= 0:
        return 2
    if p[0] <= 0 and p[1] <= 0:
        return 3
    return 4
def orientation(a, b, c):
    res = (b[1]-a[1]) * (c[0]-b[0]) - (c[1]-b[1]) * (b[0]-a[0])
    if res == 0:
        return 0
    if res > 0:
        return 1
    return -1
def compare(p1, q1):
    p = [p1[0]-mid[0], p1[1]-mid[1]]
    q = [q1[0]-mid[0], q1[1]-mid[1]]
    one = quad(p)
```

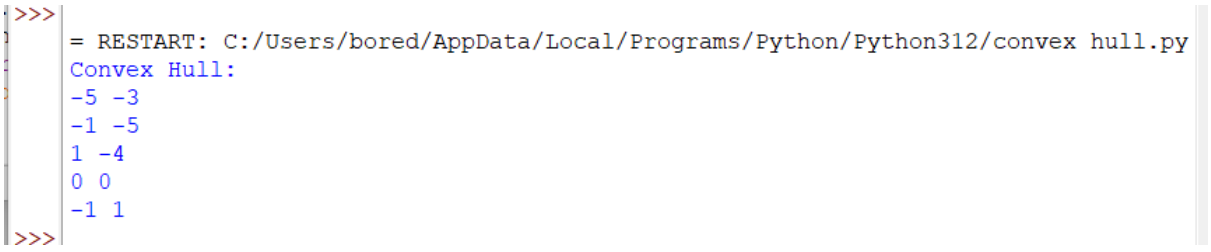
```
    two = quad(q)
    if one != two:
        if one < two:
            return -1
        return 1
    if p[1]*q[0] < q[1]*p[0]:
        return -1
    return 1
def merger(a, b):
    n1, n2 = len(a), len(b)
    ia, ib = 0, 0
    for i in range(1, n1):
        if a[i][0] > a[ia][0]:
            ia = i
    for i in range(1, n2):
        if b[i][0] < b[ib][0]:
            ib = i
    inda, indb = ia, ib
    done = 0
    while not done:
        done = 1
        while orientation(b[indb], a[inda], a[(inda+1) % n1]) >= 0:
            inda = (inda + 1) % n1
        while orientation(a[inda], b[indb], b[(n2+indb-1) % n2]) <= 0:
            indb = (indb - 1) % n2
            done = 0
    uppera, upperb = inda, indb
    inda, indb = ia, ib
    done = 0
    g = 0
    while not done:
```

```
        done = 1
        while orientation(a[inda], b[indb], b[(indb+1) % n2]) >= 0:
            indb = (indb + 1) % n2
        while orientation(b[indb], a[inda], a[(n1+inda-1) % n1]) <= 0:
            inda = (inda - 1) % n1
            done = 0
    ret = []
    lowera, lowerb = inda, indb
    ind = uppera
    ret.append(a[uppera])
    while ind != lowera:
        ind = (ind+1) % n1
        ret.append(a[ind])
    ind = lowerb
    ret.append(b[lowerb])
    while ind != upperb:
        ind = (ind+1) % n2
        ret.append(b[ind])
    return ret
def bruteHull(a):
    global mid
    s = set()
    for i in range(len(a)):
        for j in range(i+1, len(a)):
            x1, x2 = a[i][0], a[j][0]
            y1, y2 = a[i][1], a[j][1]
            a1, b1, c1 = y1-y2, x2-x1, x1*y2-y1*x2
            pos, neg = 0, 0
            for k in range(len(a)):
                if (k == i) or (k == j) or (a1*a[k][0]+b1*a[k][1]+c1 <= 0):
                    neg += 1
```

```
            if (k == i) or (k == j) or (a1*a[k][0]+b1*a[k][1]+c1 >= 0):
                pos += 1
        if pos == len(a) or neg == len(a):
            s.add(tuple(a[i]))
            s.add(tuple(a[j]))
    ret = []
    for x in s:
        ret.append(list(x))
    mid = [0, 0]
    n = len(ret)
    for i in range(n):
        mid[0] += ret[i][0]
        mid[1] += ret[i][1]
        ret[i][0] *= n
        ret[i][1] *= n
    ret = sorted(ret, key=cmp_to_key(compare))
    for i in range(n):
        ret[i] = [ret[i][0]/n, ret[i][1]/n]
    return ret
def divide(a):
    if len(a) <= 5:
        return bruteHull(a)
    left, right = [], []
    start = int(len(a)/2)
    for i in range(start):
        left.append(a[i])
    for i in range(start, len(a)):
        right.append(a[i])
    left_hull = divide(left)
    right_hull = divide(right)
    return merger(left_hull, right_hull)
```

```
if __name__ == '__main__':
    a = []
    a.append([0, 0])
    a.append([1, -4])
    a.append([-1, -5])
    a.append([-5, -3])
    a.append([-3, -1])
    a.append([-1, -3])
    a.append([-2, -2])
    a.append([-1, -1])
    a.append([-2, -1])
    a.append([-1, 1])
    n = len(a)
    a.sort()
    ans = divide(a)
    print('Convex Hull:')
    for x in ans:
        print(int(x[0]), int(x[1]))
```

OUTPUT:



8.Exhaustive Search

```
def maxPackedSets(items, sets):
    maxSets = 0
    for set in sets:
        numSets = 0
        for item in items:
            if item in set:
                numSets += 1
```

```
            items = [i for i in items if i != item]
        maxSets = max(maxSets, numSets)
    return maxSets
items = [1, 2, 3, 4, 5, 6]
sets = [
    [1, 2, 3],
    [4, 5],
    [5, 6],
    [1, 4]
]
maxSets = maxPackedSets(items, sets)
print(f"Maximum number of sets that can be packed: {maxSets}")
```

OUTPUT:

```
>>>
    = RESTART: C:/Users/bored/AppData/Local/Programs/Python/Python312/exhaustive sea
    rch.py
    Maximum number of sets that can be packed: 3
>>>
```