1.selection sort

```
arr = [64, 25, 12, 22, 11]
n = len(arr)
for i in range(n):
  min_index = i
  for j in range(i+1, n):
      if arr[j] < arr[min_index]:
          min_index = j
   arr[i], arr[min_index] = arr[min_index], arr[i]
print("Sorted array is:", arr)
```

2.Bubble sort

```
arr = [64, 34, 25, 12, 22, 11, 90]
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]
print("Sorted array is:", arr)
```

3.Insertion sort

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

4.Sequential search

```
def sequential_search(arr, target):
    for index, value in enumerate(arr):
        if value == target:
            return index  # Return the index if found
    return -1  # Return -1 if not found
arr = [3, 5, 2, 8, 1]
target = 8
print(sequential_search(arr, target))
```

5.String matching

```
def string_matching(input_string, target):
    return "Found" if target in input_string else "Not Found"
input_string = "ABABABCABABABABCABABABC"
target = "ABABC"
result = string_matching(input_string, target)
print(result)
```

6.Closest pair

```
import math
def euclidean_distance(p1, p2):
    return math.sqrt((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2)
def closest_pair_brute_force(points):
    min_distance = float('inf')
    closest_pair = None
    for i in range(len(points)):
        for j in range(i + 1, len(points)):
            dist = euclidean_distance(points[i], points[j])
            if dist < min_distance:
                min_distance = dist
                closest_pair = (points[i], points[j])
    return closest_pair, min_distance
```

```
points = [(1, 2), (4, 5), (7, 8), (3, 1)]
closest, distance = closest_pair_brute_force(points)
print(f"Closest pair: {closest} Minimum distance: {distance}")
```

7.Exhaustive search

```
def find_max_min(nums):
    return (max(nums), min(nums)) if nums else (None, None)
nums = [3, 34, 4, 12, 5, 2]
max_value, min_value = find_max_min(nums)
print(f"Maximum value: {max_value}, Minimum value: {min_value}")
```

8.Convex Hull

```
def orientation(p, q, r):
    return (q[1] - p[1]) * (r[0] - q[0]) - (q[0] - p[0]) * (r[1] - q[1])
def convex_hull(points):
    hull = []
    n = len(points)
    for i in range(n):
        for j in range(n):
            if i == j:
                continue
            valid = True
            for k in range(n):
                if k == i or k == j:
```

```
                continue
            if orientation(points[i], points[j], points[k]) > 0:
                valid = False
                break
        if valid:
            if points[i] not in hull:
                hull.append(points[i])
            if points[j] not in hull:
                hull.append(points[j])
    return hull
points = [(1, 1), (4, 6), (8, 1), (0, 0), (3, 3)]
hull = convex_hull(points)
print(f"Convex Hull: {hull}")
```