1.Discuss how your selection\_sort function handles edge cases. Specifically, consider and explain the outcomes for the following cases:

* + An empty list
  + A list with one element
  + A list with all identical elements
  + A list with negative numbers

**Test Cases:**

1. **Input:** []
   * **Expected Output:** []
2. **Input:** [1]
   * **Expected Output:** [1]
3. **Input:** [7, 7, 7, 7]
   * **Expected Output:** [7, 7, 7, 7]
4. **Input:** [-5, -1, -3, -2, -4]
   * **Expected Output:** [-5, -4, -3, -2, -1]

a=[-1,-5,-3,-4,-2]

n=len(a)

temp=0

for i in range(n):

min\_index=i

for j in range(i+1,n):

if a[j]<a[min\_index]:

min\_index=j

if min!=i:

a[i],a[min\_index]=a[min\_index],a[i]

print("sorted array",a)

2. Describe the Selection Sort algorithm's process of sorting an array. Selection Sort works by dividing the array into a sorted and an unsorted region. Initially, the sorted region is empty, and the unsorted region contains all elements. The algorithm repeatedly selects the smallest element from the unsorted region and swaps it with the leftmost unsorted element, then moves the boundary of the sorted region one element to the right. Explain why Selection Sort is simple to understand and implement but is inefficient for large datasets. Provide examples to illustrate step-by-step how Selection Sort rearranges the elements into ascending order, ensuring clarity in your explanation of the algorithm's mechanics and effectiveness.

**Example Scenarios**:

1. **Sorting a Random Array**:
   * **Input**: [5, 2, 9, 1, 5, 6]
   * **Output**: [1, 2, 5, 5, 6, 9]
2. **Sorting a Reverse Sorted Array**:
   * **Input**: [10, 8, 6, 4, 2]
   * **Output**: [2, 4, 6, 8, 10]
3. **Sorting an Already Sorted Array**:
   * **Input**: [1, 2, 3, 4, 5]
   * **Output**: [1, 2, 3, 4, 5]

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

n=len(a)

temp=0

for i in range(n):

min\_index=i

for j in range(i+1,n):

if a[j]<a[min\_index]:

min\_index=j

a[i],a[min\_index]=a[min\_index],a[i]

print("sorted array",a)

3. Modify your bubble\_sort function to stop early if the list becomes sorted before all passes are completed. Explain why this optimization improves performance and how it affects the time complexity in the best case.

**Test Cases:**

* Test your optimized function with the following lists:
  1. **Input:** [64, 25, 12, 22, 11]
     + **Expected Output:** [11, 12, 22, 25, 64]
  2. **Input:** [29, 10, 14, 37, 13]
     + **Expected Output:** [10, 13, 14, 29, 37]
  3. **Input:** [3, 5, 2, 1, 4]
     + **Expected Output:** [1, 2, 3, 4, 5]
  4. **Input:** [1, 2, 3, 4, 5] (Already sorted list)
     + **Expected Output:** [1, 2, 3, 4, 5]
  5. **Input:** [5, 4, 3, 2, 1] (Reverse sorted list)
     + **Expected Output:** [1, 2, 3, 4, 5]

a=[1,7,4,2,5]

n=len(a)

c=0

for i in range(0,n):

c=0

for j in range(i+1,n):

if a[i]>a[j]:

a[i],a[j]=a[j],a[i]

c=c+1

if c==0:

break

print("sorted array",a)

4.Describe how Insertion Sort manages arrays with duplicate elements during the sorting process. Explain the algorithm's behavior when encountering duplicate values, including whether it preserves the relative order of duplicates and how it affects the overall sorting outcome. Provide specific examples with arrays containing duplicate integers, demonstrating how Insertion Sort sorts the array while ensuring duplicates are correctly positioned, and discuss any considerations or adjustments that might be necessary.

Example Scenarios:

1. Array with Duplicates:
   * Input: [3, 1, 4, 1, 5, 9, 2, 6, 5, 3]
   * Output: [1, 1, 2, 3, 3, 4, 5, 5, 6, 9]
2. All Identical Elements:
   * Input: [5, 5, 5, 5, 5]
   * Output: [5, 5, 5, 5, 5]
3. Mixed Duplicates:
   * Input: [2, 3, 1, 3, 2, 1, 1, 3]
   * Output: [1, 1, 1, 2, 2, 3, 3, 3]

def insertionsort(a):

n=len(a)

for i in range(1,n):

temp=a[i]

j=i-1;

while(j>=0 and a[j]>temp):

a[j+1]=a[j]

j=j-1

a[j+1]=temp

return a

a=list(map(int,input("enter the array :").split()))

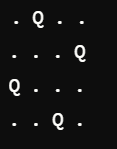
res=insertionsort(a)

print("sorted array :",res)

5.Discuss the importance of visualizing the solutions of the N-Queens Problem to understand the placement of queens better. Use a graphical representation to show how queens are placed on the board for different values of N. Explain how visual tools can help in debugging the algorithm and gaining insights into the problem's complexity. Provide examples of visual representations for N = 4, N = 5, and N = 8, showing different valid solutions.

**Example Scenarios**:

1. **Visualization for 4-Queens**:
   * **Input**: N = 4
   * **Output**:



* + **Explanation**: Each 'Q' represents a queen, and '.' represents an empty space.

def solve\_and\_print\_n\_queens(N):

board = [-1] \* N

solutions = []

def solve(row):

if row == N:

solutions.append(board[:])

return

for col in range(N):

valid = True

for prev\_row in range(row):

if board[prev\_row] == col or \

board[prev\_row] - prev\_row == col - row or \

board[prev\_row] + prev\_row == col + row:

valid = False

break

if valid:

board[row] = col

solve(row + 1)

board[row] = -1

solve(0)

print(f"Solutions for N = {N}:")

for solution in solutions:

for row in range(N):

line = ['.'] \* N

line[solution[row]] = 'Q'

print(' '.join(line))

print()

N\_values = [4]

for N in N\_values:

solve\_and\_print\_n\_queens(N)