

TOPIC 2: BRUTE FORCE ALGORITHM

1. List Examples

Aim: Demonstrate different types of lists.

Algorithm:

1. Create an empty list.
2. Create a list with one element.
3. Create a list with all identical elements.
4. Create a list with negative numbers and sort it.

Python code:

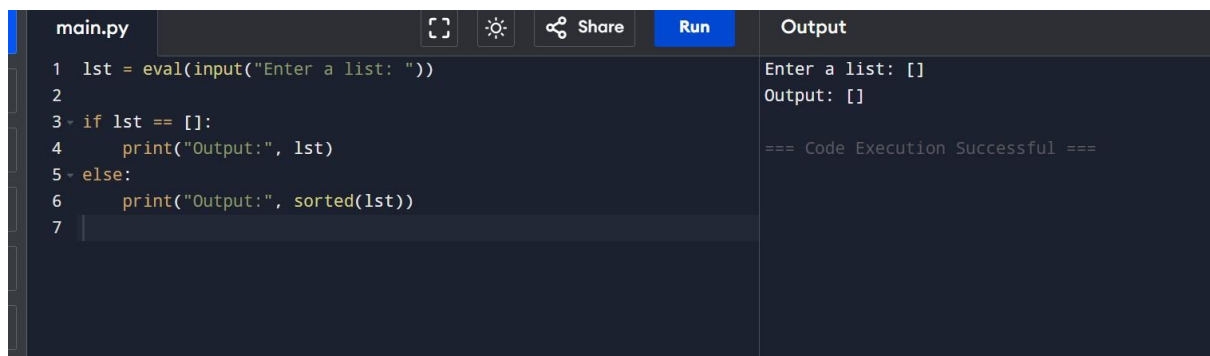
```
lst = eval(input("Enter a list: "))  
if  
lst == []:  
    print("Output:", lst) else:  
    print("Output:", sorted(lst))
```

Input:

[]

Output:

[]



The screenshot shows a code editor with a file named 'main.py'. The code is as follows:

```
1 lst = eval(input("Enter a list: "))  
2  
3 if lst == []:  
4     print("Output:", lst)  
5 else:  
6     print("Output:", sorted(lst))  
7
```

The 'Output' pane on the right shows the following text:

```
Enter a list: []  
Output: []  
  
=== Code Execution Successful ===
```

2. Selection Sort

Aim: Sort an array using Selection Sort.

Algorithm:

1. Divide the array into sorted and unsorted parts.
2. Find the minimum element in the unsorted part.
3. Swap it with the first element of the unsorted part.
4. Repeat until the array is fully sorted.

Python Code:

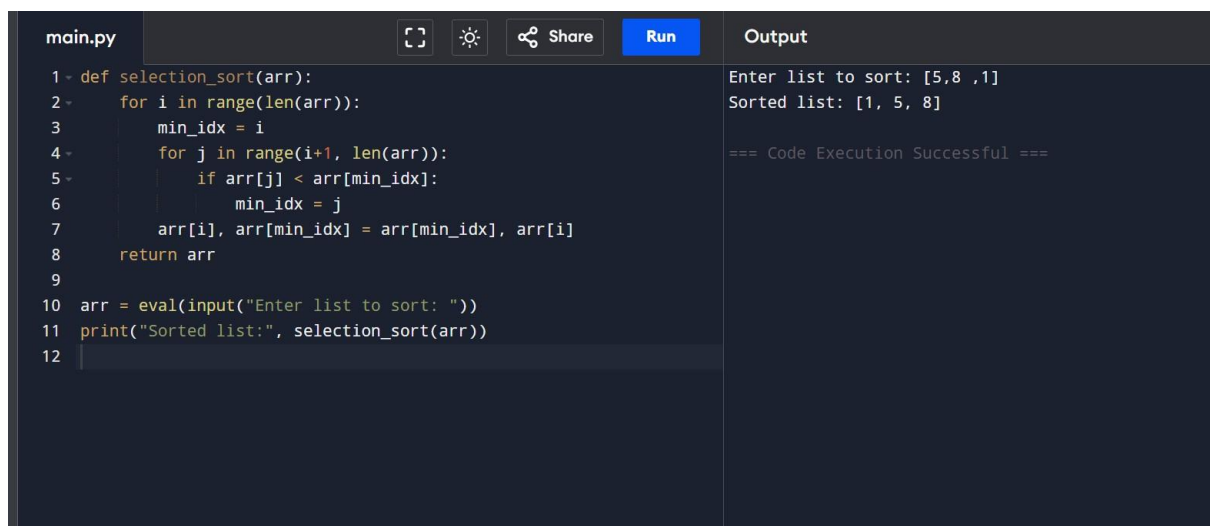
```
def selection_sort(arr):  
    for i in range(len(arr)):  
        min_idx = i  
        for j in  
            range(i+1, len(arr)):            if  
                arr[j] < arr[min_idx]:  
                    min_idx = j  
        arr[i], arr[min_idx] = arr[min_idx], arr[i]  
    return arr  
  
arr = eval(input("Enter list to sort: "))  
print("Sorted  
list:", selection_sort(arr))
```

Input:

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

Output:

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



```
main.py  [Run] [Share] [Settings] [Full Screen]  
1- def selection_sort(arr):  
2-     for i in range(len(arr)):  
3-         min_idx = i  
4-         for j in range(i+1, len(arr)):  
5-             if arr[j] < arr[min_idx]:  
6-                 min_idx = j  
7-         arr[i], arr[min_idx] = arr[min_idx], arr[i]  
8-     return arr  
9-  
10 arr = eval(input("Enter list to sort: "))  
11 print("Sorted list:", selection_sort(arr))  
12
```

Output

```
Enter list to sort: [5,8 ,1]  
Sorted list: [1, 5, 8]  
  
=== Code Execution Successful ===
```

3. Optimized Bubble Sort

Aim: Stop Bubble Sort early if the list is already sorted.

Algorithm:

1. Compare adjacent elements and swap if needed.
2. If no swaps occur in a pass, the list is sorted.

Python Code:

```
def bubble_sort(arr):    n = len(arr)
    for i in range(n):        swapped = False
        for j in range(0, n-i-1):            if arr[j] >
arr[j+1]:                arr[j], arr[j+1] =
arr[j+1], arr[j]                swapped = True
    if not swapped:
        break
    return arr
```

```
arr = eval(input("Enter list to sort: "))
print("Sorted list:", bubble_sort(arr))
```

Input:

[64, 25, 12, 22, 11]

[29, 10, 14, 37, 13]

[3, 5, 2, 1, 4]

[1, 2, 3, 4, 5]

[5, 4, 3, 2, 1]

Output:

[11, 12, 22, 25, 64]

[10, 13, 14, 29, 37]

[1, 2, 3, 4, 5]

[1, 2, 3, 4, 5]

[1, 2, 3, 4, 5]

main.py	Output
<pre>1- def bubble_sort(arr): 2- n = len(arr) 3- for i in range(n): 4- swapped = False 5- for j in range(0, n-i-1): 6- if arr[j] > arr[j+1]: 7- arr[j], arr[j+1] = arr[j+1], arr[j] 8- swapped = True 9- if not swapped: 10- break 11- return arr 12- 13- arr = eval(input("Enter list to sort: ")) 14- print("Sorted list:", bubble_sort(arr)) 15-</pre>	<pre>Enter list to sort: [3, 8,6,2,9] Sorted list: [2, 3, 6, 8, 9] === Code Execution Successful ===</pre>

4. Insertion Sort with Duplicates Aim:

Sort arrays including duplicates.

Algorithm:

1. Take one element at a time and insert it in its correct position in the sorted part.
2. Relative order of duplicates is preserved.

Python Code:

```
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 = eval(input("Enter list to sort: ")) print("Sorted
```

```
list:", insertion_sort(arr))
```

Input:

[3, 1, 4, 1, 5, 9, 2, 6, 5, 3]

[5, 5, 5, 5, 5] [2, 3,

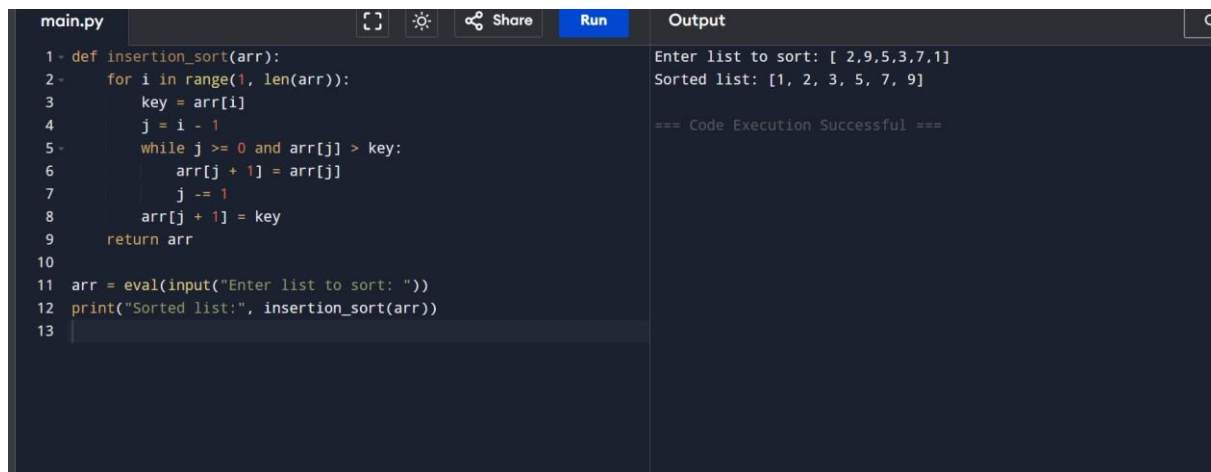
1, 3, 2, 1, 1, 3]

Output:

[1, 1, 2, 3, 3, 4, 5, 5, 6, 9]

[5, 5, 5, 5, 5]

[1, 1, 1, 2, 2, 3, 3, 3]



```
main.py [ ] [ ] [ ] Share Run Output
1 def insertion_sort(arr):
2     for i in range(1, len(arr)):
3         key = arr[i]
4         j = i - 1
5         while j >= 0 and arr[j] > key:
6             arr[j + 1] = arr[j]
7             j -= 1
8         arr[j + 1] = key
9     return arr
10
11 arr = eval(input("Enter list to sort: "))
12 print("Sorted list:", insertion_sort(arr))
13
```

Enter list to sort: [2,9,5,3,7,1]
Sorted list: [1, 2, 3, 5, 7, 9]
=== Code Execution Successful ===

5. Kth Missing Positive

Aim: Find the kth missing positive number.

Algorithm:

1. Start from 1 and check each number.
2. Count missing numbers until k is reached.

Python Code:

```
def findKthPositive(arr, k):
    missing = []
    current = 1
    while len(missing) < k:
        if current not in arr:
            missing.append(current)
        current += 1
    return missing[-1]
```

```
arr = eval(input("Enter sorted list: ")) k = int(input("Enter k:
")) print("Kth Missing Positive Number:", findKthPositive(arr,
k))
```

Input:

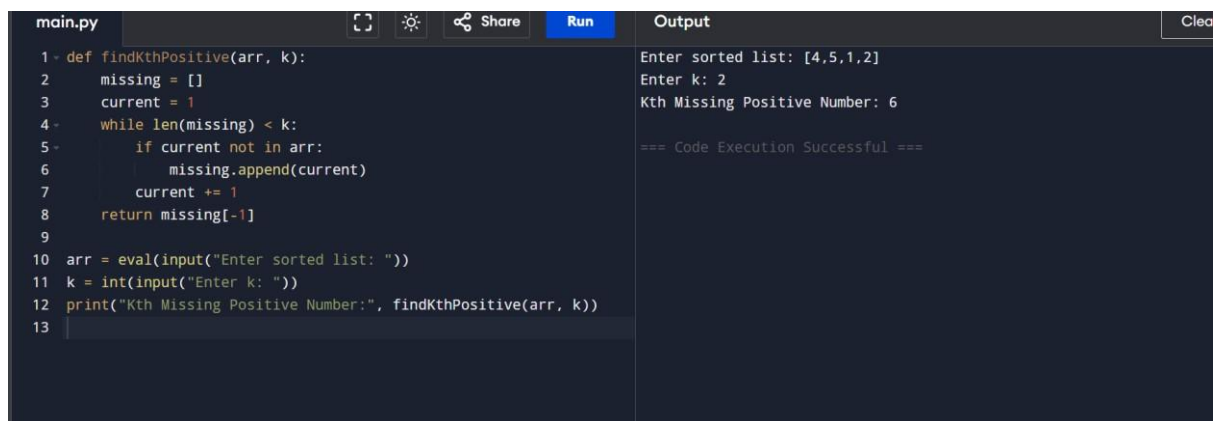
[2,3,4,7,11], k=5

[1,2,3,4], k=2

Output:

9

6



```
main.py  [ ] [ ] [ ] Share Run Output Clear
1- def findKthPositive(arr, k):
2-     missing = []
3-     current = 1
4-     while len(missing) < k:
5-         if current not in arr:
6-             missing.append(current)
7-             current += 1
8-     return missing[-1]
9
10 arr = eval(input("Enter sorted list: "))
11 k = int(input("Enter k: "))
12 print("Kth Missing Positive Number:", findKthPositive(arr, k))
13

Enter sorted list: [4,5,1,2]
Enter k: 2
Kth Missing Positive Number: 6

=== Code Execution Successful ===
```