Write a Program to Bubble Sort

Time Complexity : worst_case O(n2)

space complexity : O(1) Total Number of Swap per Pass: n-1

Total Number of swap : n*(n-1)/2

Algorithm Summary :

- This algorithm sort the adjacent elements
- This alorithm is not suitable for Large dataset due to high complexity
- Can be prefer when we dont have care about Time complexity and memory cost is high

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

bubblesort([10,20,4,1,4,2])
```

```
[1, 2, 4, 4, 10, 20]
```

Write a program to the selection Sort :

Defination : Selection sort is sorting algorithm which sort the element by finding the minimum element in the array then it swap the element.

- It is quite better than bubble sort because it total number of swap is 0(n) times
- It is stable Algorithm (Order of the sequence is preserve)

Time and space complexity:

- Time Complexity for worst Case : O(n2)
- Space Complexity :O(1)

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

```
Selection_sort([10,4,1,3,4])
```

```
[1, 3, 4, 4, 10]
```

Write a Program to Quick sort:

** Quick sort is an sorting algorithm that use divide and conquer approach to solve the problems .It select the piviot form the array and compare the pivot elemnet with the array and then divide the array element into three segment first :

```
      * left_array: that contain the value less than the pivot
      * Right_array: This array contain the value greater than the
pivot
      * middle_array:that contain the array equal to the pivot
```

** This Algorithm is not sitable for Small dataset ** It is not Stable Algorithm

Time and space complexity of the Quick sort:

^ Time Complexity:

```
  * Best case       : O(N log(N))
  * Average case y : O(N log(N))
  * Worst case      : O(N2)
```

^ Space Complexity :

```
 * Auxiallary Space : O(1)
 * Worst case       : O(N)
```

We can implement this algorithm using two Approaches:

- Using List comprension
- in -place Quick Sort

```
# Using List comprension

def Quick_sort(arr):
  if len(arr)<=1:
    return arr
  pivot=arr[0]

  left_array=[i for i in arr if i<pivot]
  right_array=[i for i in arr if i>pivot]
  middle_array=[i for i  in arr if i==pivot]

  return Quick_sort(left_array)+middle_array+Quick_sort(right_array)
```

```
Quick_sort([1,10,2,4,11])
```

```
[1, 2, 4, 10, 11]
```

```
# Qick Sort Using In Place


def partition(arr,low,high):
  pivot=arr[high]
  i=low-1
  for j in  range(low,high):
    if arr[j]<pivot:
      i+=1
      arr[i],arr[j]=arr[j],arr[i]

    # This place the pivot element at the last of the array
  arr[i+1],arr[high]=arr[high],arr[i+1]

    # this Function will return the partion elemnet
  return i+1

def Quick_sort(arr,low,high):
  if low<high:
    pi=partition(arr,low,high)
    Quick_sort(arr,low,pi-1)
    Quick_sort(arr,pi+1,high)


arr=eval(input("ENter array elemnet "))
low=0
high=(len(arr)-1)
Quick_sort(arr,low,high)
print (arr)
```

merge Sort :

Defination : MergeSort is an simple sorting algorithm that works on the principle of divide and concqure approach it recursively partion array until we get the individual element is sorted then we merge it such a way that we get result as sorted array.

- It is popular algorithm that is known for Efficiency and Stability.
- This array works well for the large dataset.
- It is an efficient Sorting Technique.

- It can not be used where cost of the memory is high
- It gives gaurrented performance when worst case also occure.

Time and Space COmplexity of the algorithm :

^ Time Complexity :

```
      * Best Case    : O(N log(N))
      * Average Case : O(N log(N))
      * Worst case   : O(N log(N))
```

^ Space Complexity :

```
      * Average case :   O(1)
      * Worst case   :  O(N)
```

Additionaly space is reqire for temparory space is required during the merging

^ Application of the Merge Sort :

```
    * Sorting the Large Dataset
    * External Sorting
    * Finding median of the large dataset
```

How it works :

```
  * divide : Divide the array into len(arr) times
  * concqure : trated the individual element as sorted
  * Merge : merge the array in such a way that we get sorted array

# define Partition function :



def Merge_Sort(arr):
  if len(arr)<=1:
    return arr
  mid=len(arr)//2
  left_array=arr[:mid]
  right_array=arr[mid:]

  sorted_left_array=Merge_Sort(left_array)
  sorted_right_array=Merge_Sort(right_array)

  return merge(sorted_left_array,sorted_right_array)



def merge(left_array,right_array):
```

```
  sorted_array=[]
  left_index=right_index=0
  while left_index<len(left_array) and right_index<len(right_array):
    if left_array[left_index]<right_array[right_index]:
      sorted_array.append(left_array[left_index])
      left_index +=1
    else :
      sorted_array.append(right_array[right_index])
      right_index +=1
  sorted_array.extend(left_array[left_index:])
  sorted_array.extend(right_array[right_index:])

  return sorted_array

Merge_Sort([10,2,3,4,5,0])
```

```
[0, 2, 3, 4, 5, 10]
```

Searching algorithm :

```
 * Defination of Searching Algorithm :

      Searching algorithm are use search the index of the element of
the array

1) Linear Search
2) Binary Search
```

1) Linear Search :

```
  Defination : It seach Element by iterating over the array linearly
and compare the serch key with the element and return the location of
the elemnt
```

^ Time COmplexity :

- best COmplexity:O(1)
- averge COmplexity : O(N) it perfom n/2 times operation but 1/2 is constant hence O(N)
- Worst complexity is : O(N) It compare the element at n times ^ Space COmplexity : It does not require any type of memory . It just iterate over the array.hence Space compLexity=O(1)

Advantage of Linear Search :

```
  * Simple to implement and understand
  * Works on unsorted data
  * Applicable to various data structures
```

Disadvantages of Linear Search :

```
  * In efficient for large datasets
  * Not the fastest search algorithm
```

Application of the Linear search:

```
* Single-pass search scenarios
* Finding duplicates
* Small datasets
* Unsorted data
```

```
from re import I
# Implemnetation of the Linear Search


def Linear_search(arr,key):
  for i in range(len(arr)):
    if arr[i]==key:
      return i
  print ("Search Key is not Found ")
  return -1


arr=eval(input("Enter the array element "))

search_key=int(input("Enter search key"))

print("Search key is fount at index :",Linear_search(arr,search_key))
```

```
Enter the array element [10,20,30,40,50]
Enter search key50
Search key is fount at index : 4
```

2) Binary Search Algorithm :

Defination : It is simple seaching algorithm .This algorithm is only works when we are having sorted data It finds the mid of the array and then compare the mid with the search key if search key is equal to the mid then returns the mid elseif mid is less than seach key then it update the right to the mid-1 elif seach key is greater then it update the left to the mid+1

Complexity of the Binary search:

^ Time COmplexity :
* Best case : O(1) * Average complexity : O(N log (N)) * Worst case : O(N) // Loop will be executed until the len(aar)//2 ^ Space complexity : O(1)

Advantages of Binary Search :

```
  * It is efficient than the Linear search beacause it iterate over
the array by len(arr)//2 times This is maximum case time complexity
  * Simplicity and Implementation
```

Disadvantages of the Binary Search :

```
    * It require to be sorted
    * Recursive Overhead: In the recursive implementation, there is
additional overhead due to recursive function calls, which can be a
disadvantage for very large datasets or systems with limited stack
memory
```

Application Binary search :

```
  * Finding Elements in a Sorted Array
  * Database Indexing: Databases often use binary search or its
variants to quickly locate records in sorted indexes.
  * Search Engines: Binary search can be used in search engines for
efficiently retrieving data from large, sorted datasets.
```

```python
# Binary Search Implementation


def Binary_search(arr,key,low,high):
  if low>high:
    return -1

  mid =(low+high)//2
  if arr[mid]==key:
    return mid
  elif arr[mid]>key:
    high=mid-1
    return Binary_search(arr,key,low,high)

  else:
    low=mid+1
    return Binary_search(arr,key,low,high)
  return -1


arr=eval(input("Enter sorted array element"))
key=int(input("Enter search key"))
index=Binary_search(arr,key,0,len(arr)-1)
if index !=-1:
  print(f"element is found at index {index}")
else:
    print("Enter search key is not present in array")
```

```
Enter sorted array element[10,30,40]
Enter search key40
element is found at index 2
```

Stack Data Structure :

1) Defination : Stack is Linear Data Structure which store data in contineous memory allocation . 2) It follows Last in First Out principle (LIFO). 3) LIFO implies that the element that is inserted last, comes out first and FILO implies that the element that is inserted first, comes out last.

Key Operations on Stack Data Structures

1) Push: Adds an element to the top of the stack. 2) Pop: Removes the top element from the stack. 3_ Peek: Returns the top element without removing it. 4) IsEmpty: Checks if the stack is empty. 5) IsFull: Checks if the stack is full (in case of fixed-size arrays).

Application of Stack :

- Recursion
- Expression Evaluation and Parsing
- Depth-First Search (DFS)
- Undo/Redo Operations
- Browser History
- Function Calls
- Real World Example of stack :
  - A stack of plates: we add and remove plates from the top.

Time and Space complexity :

- Time Complexity:
  - Push: O(1)
  - Pop: O(1)
  - Peek: O(1)
- Space complexity :

  Space complexity =O(N) times

```python
class stack:
  def __init__(self,size):
    self.stack1=[]
    self.size=size
  def isfull(self):
    return len(self.stack1)==self.size

  def isempty(self):
    return len(self.stack1)==0

  # This function insert element at the last of the stack
```

```
  def push(self,element):
    if self.isfull():
      print("stack is Full Element cannot be inserted ")
    else :
      self.stack1.append(element)
      return self.stack1

  # this function remove the last element of the stack
  def pop(self):
    if self.isempty():
      print("Stack is empty")
    else:
      self.stack1.pop()
      return self.stack1

  # this function select the first elemnet of the stack
  def peek(self):
    return self.stack1[-1]


size=int(input("Enter the size of the stack"))
stack11=stack(size)
stack11.push(10)
stack11.push(20)
stack11.push(30)
stack11.push(40)


stack11.pop()
stack11.pop()

stack11.peek()
print(stack11.stack1)
```

```
Enter the size of the stack6
[10, 20]
```

Queue Data Structure :

- Defination : Queueue is Linear data structure that store data in contineous memory allocation .

- It has two end Front and rear :

  - Front : From front elemnet is removed
  - Rear : From rear Element is added to the queue

Basic Operations of Queue Data Structure

- Enqueue (Insert): Adds an element to the rear of the queue.
- Dequeue (Delete): Removes and returns the element from the front of the queue.

- Peek: Returns the element at the front of the queue without removing it.
- Empty: Checks if the queue is empty.
- Full: Checks if the queue is full.

Applications of Queue

- Task scheduling in operating systems
- Data transfer in network communication
- Simulation of real-world systems (e.g., waiting lines)
- Priority queues for event processing queues for event processing

Time and space complexity of the Queue:

- Time Complexity:
  - Enqueue (enqueue): O(1)
  - Dequeue (dequeue): O(1)
  - Peek (peek): O(1)
- Space Complexity: O(n), where n is the number of elements in the queue.

```
class Queue:
  def __init__(self,size):
    self.queue=[]
    self.front=self.rear=0
    self.size=size

    # this function check Whether queue is full or not
  def queueisFull(self):
    return len(self.queue)==self.size

    # this function check whether queue is empty or not
  def queueIsEmpty(self):
    return len(self.queue)==0

    # This function insert the elemnet in the queue
  def enqueue(self,element):
    if self.queueisFull():
      print("Print Queue is Full")

    else :
      self.queue.append(element)
      self.rear +=1
      return self.queue

      # This function remove element from the queue
  def dequeue(self):
    if self.queueIsEmpty():
      print("Queue is Empty")
    else :
      self.queue.pop(0)
```

```
      self.rear -=1
      return self.queue

# This function return the next element to remove in queue

  def peek(self):
    if self.queueIsEmpty():
      print("Queue is empty")
    return self.queue[0]


size=int(input("Enter the size of the queue "))
queue1=Queue(size)
queue1.enqueue(10)
queue1.enqueue(20)
queue1.enqueue(30)
queue1.enqueue(40)

print("Original Queue is : ", queue1.queue)

queue1.dequeue()
print("Original Queue after dequeue method  : ", queue1.queue)
print(queue1.queue)
queue1.dequeue()

print()
print("Original Queue after dequeue method  : ", queue1.queue)
print(queue1.queue)
print(queue1.peek())
```

```
Enter the size of the queue 7
Original Queue is :  [10, 20, 30, 40]
Original Queue after dequeue method  :  [20, 30, 40]
[20, 30, 40]

Original Queue after dequeue method  :  [30, 40]
[30, 40]
30
```

Linked List Data Structure :

- defination of Linked List:
  - Linked List is a linear data structure which looks like a series of nodes, where each node has two parts: data and next pointer.
  - Unlike Arrays, Linked List elements are not stored at a contiguous location.
  - In the linked list there is a head pointer, which points to the first element of the linked list, and if the list is empty then it simply points to null or nothing.

Basic Terminologies of Linked List:

- Head: The Head of a linked list is a pointer to the first node or reference of the first node of linked list. This pointer marks the beginning of the linked list.
- Node: Linked List consists of a series of nodes where each node has two parts: data and next pointer.
- Data: Data is the part of node which stores the information in the linked list.
- Next pointer: Next pointer is the part of the node which points to the next node of the linked list.

Importance of the LinkedList:

- Dynamic Data structure: The size of memory can be allocated or de-allocated at run time based on the operation insertion or deletion.
- Ease of Insertion/Deletion: The insertion and deletion of elements are simpler than arrays since no elements need to be shifted after insertion and deletion, Just the address needed to be updated.
- Efficient Memory Utilization: As we know Linked List is a dynamic data structure the size increases or decreases as per the requirement so this avoids the wastage of memory.
- Implementation: Various advanced data structures can be implemented using a linked list like a stack, queue, graph, hash maps, etc.

Operations on Singly Linked List:

- Insertion: The insertion operation can be performed in three ways. They are as follows…
  - Inserting At the Beginning of the list
  - Inserting At End of the list
  - Inserting At Specific location in the list
- Deletion: The deletion operation can be performed in three ways. They are as follows…
- Deleting from the Beginning of the list
- Deleting from the End of the list
- Deleting a Specific Node
- Search: It is a process of determining and retrieving a specific node either from the front, the end or anywhere in the list.
- Display: This process displays the elements of a Single-linked list.

Advantages of Linked Lists:

- Dynamic size:
- Efficient Insertion and Deletion : O(N) time complexity
- Memory Efficiency

Disadvantages of Linked List:

```
* Slow Access Time
* Pointers
* Extra memory required
```

Application of LinkedList :

- The list of songs in the music player are linked to the previous and next songs.
- In a web browser, previous and next web page URLs are linked through the previous and next buttons.
- In image viewer, the previous and next images are linked with the help of the previous and next buttons.

```python
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None

    def InsertElementAtBegining(self, data):
        new_Node = Node(data)
        new_Node.next = self.head
        self.head = new_Node

    def InsertElementAtEnd(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
            return
        current = self.head
        while current.next:
            current = current.next
        current.next = new_node

    def TraverseLinkedList(self):
        current = self.head
        while current:
            print(current.data, end="->")
            current = current.next
        print("None")

    def FindtheMidOffLinkedList(self):
        slow = self.head
        fast = self.head
        while fast and fast.next:
            slow = slow.next
            fast = fast.next.next
```

```
        return slow

    def FindMiddleValueOfLinkedList_using_list(self):
        current = self.head
        arr = []
        while current:
            arr.append(current)
            current = current.next
        if not arr:
            return None
        return arr[(len(arr) - 1) // 2]

    def InsertElementAtMiddle(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
            return
        middle_node=FindMiddleValueOfLinkedList_using_list()
        new_node.next = middle_node.next
        middle_node.next = new_node

    def InsertElementAtspecifiedPosition(self, data, position):
        new_node = Node(data)
        if position < 0:
            return "Position Invalid"
        if position == 0:
            self.InsertElementAtBegining(data)
            return
        current = self.head
        count = 0
        while current is not None and count < position - 1:
            current = current.next
            count += 1
        if current is None:
            return "Invalid Position"
        new_node.next = current.next
        current.next = new_node

    def RemoveElementFrom_begining(self):
        if self.head is None:
            return
        self.head = self.head.next

    def removeElementfrom_last(self):
        if self.head is None:
            return
        if self.head.next is None:
            self.head = None
            return
        current = self.head
```

```
        while current.next.next:
            current = current.next
        current.next = None

    def RemoveElementfromLinkedlist_by_specified_pos(self, position):
        if self.head is None:
            return
        if position == 0:
            self.head = self.head.next
            return
        current = self.head
        count = 0
        while current is not None and count < position - 1:
            current = current.next
            count += 1
        if current is None or current.next is None:
            return "Invalid Position"
        current.next = current.next.next

# Test the LinkedList
linkedList1 = LinkedList()
linkedList1.InsertElementAtBegining(1)
linkedList1.InsertElementAtBegining(2)
linkedList1.InsertElementAtBegining(3)
linkedList1.InsertElementAtBegining(4)
linkedList1.InsertElementAtBegining(5)
linkedList1.InsertElementAtBegining(6)

print("Linked List Elements are:")
linkedList1.TraverseLinkedList()

linkedList1.InsertElementAtEnd(7)
linkedList1.InsertElementAtEnd(8)

print("Linked List Elements after adding at the end:")
linkedList1.TraverseLinkedList()

# Insert node at the middle
linkedList1.InsertElementAtMiddle(9)
print("Linked List Elements after adding at the middle:")
linkedList1.TraverseLinkedList()

# Insert node at a specified position
linkedList1.InsertElementAtspecifiedPosition(10, 2)
print("Linked List Elements after adding at specified position (index
2):")
linkedList1.TraverseLinkedList()

# Remove element from the beginning
linkedList1.RemoveElementFrom_begining()
```

```
print("Linked List Elements after removing from the beginning:")
linkedList1.TraverseLinkedList()

# Remove element from the end
linkedList1.removeElementfrom_last()
print("Linked List Elements after removing from the end:")
linkedList1.TraverseLinkedList()

# Remove element from a specified position
linkedList1.RemoveElementfromLinkedlist_by_specified_pos(2)
print("Linked List Elements after removing from specified position
(index 2):")
linkedList1.TraverseLinkedList()
```

```
Linked List Elements are:
6->5->4->3->2->1->None
Linked List Elements after adding at the end:
6->5->4->3->2->1->7->8->None
Linked List Elements after adding at the middle:
6->5->4->3->9->2->1->7->8->None
Linked List Elements after adding at specified position (index 2):
6->5->10->4->3->9->2->1->7->8->None
Linked List Elements after removing from the beginning:
5->10->4->3->9->2->1->7->8->None
Linked List Elements after removing from the end:
5->10->4->3->9->2->1->7->None
Linked List Elements after removing from specified position (index 2):
5->10->3->9->2->1->7->None
```

Basic Program :

1) Write a programe to find the square root of a number

```
# use exponential operator
def squart1(num):
    return num ** 0.5

# or

import math
def squart2(num):
  return math.sqrt(num)

print(squart1(4))

print(squart2(16))

print(squart1(-16))
```

```
2.0
4.0
(-2.4492935982947064e-16-4j)
```

2) How to find the cube root of the number

```
def Cuberoot(num):
  return num **(1/3)

Cuberoot(27)

3.0
```

###3) Count the number of occurances of each character in a given String

```
def count_char_occurences(str1):
  char_count={}
  for i in str1:
    if i in char_count:
      char_count[i] +=1
    else :
      char_count[i]=1
  return char_count
str1=input("Enter the String")
char_count=count_char_occurences(str1)
print("printing count of the character in sorted order of character")

for i in sorted(char_count.keys()):
  print(i," : ",char_count[i])

Enter the Stringhello soted dictionary
printing count of the character in sorted order of character
   :  2
a  :  1
c  :  1
d  :  2
e  :  2
h  :  1
i  :  2
l  :  2
n  :  1
o  :  3
r  :  1
s  :  1
t  :  2
y  :  1
```

4) sort an Array Containing as 1s and 2s

```
def SortOnesAndTwos(arr):
  one_array=[]
  two_array=[]
  for i in arr:
    if i==1:
      one_array.append(i)
    else :
      two_array.append(i)
  return (one_array+two_array)

arr=[1,2,1,2,2,1]
print("Original array :",arr)

print("Sorted array: ",SortOnesAndTwos(arr))

# Disadvantage of this technique is time and space complexity of this code is :O(N)
```

```
Original array : [1, 2, 1, 2, 2, 1]
Sorted array:  [1, 1, 1, 2, 2, 2]
```

5) Maximum sum in sub array (Kadanes Algorithm)

```
def maximumSumInsubarray(arr):
  max_sum_global=arr[0]
  max_sum_local=arr[0]
  for num in range(1,len(arr)):
   max_sum_local +=arr[num]
   if max_sum_local>max_sum_global:
    max_sum_global=max_sum_local
   if max_sum_local<0:
    max_sum_local=0
  return max_sum_global



maximumSumInsubarray([1,2,3,-1,-2])
```

```
6
```

```
# Another apprach to solve this is

def Max_Sumof(arr):
  max_sumGlobal=max_sum_local=arr[0]
  for num in arr[1:]:
    max_sum_local=max(num,(max_sum_local+num))
    max_sumGlobal=max(max_sumGlobal,max_sum_local)
  return max_sumGlobal
```

```
Max_Sumof([1,2,3,-1,-2])
```

```
6
```

6) Find the number in the array that has occure once and every other elemnet is occure twice

```
def findthelemnet(arr):
  for num in arr:
    if arr.count(num)==1:
      return num

list1=[1,2,3,4,1,2,3,4,5]
findthelemnet(list1)
```

```
5
```

7.Loop in linked list || Find the Middle of the list

```
class Node:
  def __init__(self,data):
    self.data=data
    self.next=None
class LinkedList:
  def __init__(self):
    self.head=None
  def InsertNodeAtBeggining(self,data):
    new_node=Node(data)
    new_node.next=self.head
    self.head=new_node
  def InsertNodeAtEnd(self,Data):
    new_node=Node(Data)
    if self.head is None:
      self.head=new_node
      return
    current=self.head
    while current.next:
      current=current.next
    current.next=new_node
    return
  def TraverseLinkedList(self):
    current=self.head
    while current is not None:
      print(current.data,end="->")
      current=current.next
    print()

  def FindMiddleNode(self):
    slow=fast=self.head
```

```python
    while fast and fast.next:
      slow=slow.next
      fast=fast.next.next
    return slow.data
  def reversedLlinkedlist(self):
    stack=[]
    current=self.head
    while current:
      stack.append(current.data)
      current=current.next
    for i in range(len(stack),-1,-1):
      print(i,"->",end=" ")
  def reversedLInkedlistWithout_stack(self):
    prev =None
    current=self.head
    while current:
      next_node=current.next
      current.next=prev
      prev=current
      current=next_node
    self.head=prev
    return



linkedlist1=LinkedList()

print("Inserting eleent into the data")

linkedlist1.InsertNodeAtBeggining(10)
linkedlist1.InsertNodeAtBeggining(20)
linkedlist1.InsertNodeAtBeggining(30)
linkedlist1.InsertNodeAtBeggining(40)
linkedlist1.InsertNodeAtBeggining(50)

print("Linked List Traversing :")

linkedlist1.TraverseLinkedList()

linkedlist1.InsertNodeAtEnd(60)

print("Linked list after the Inserting node at the end :")
linkedlist1.InsertNodeAtEnd(70)


linkedlist1.TraverseLinkedList()

print("Middle Node of the linkedlist: " ,linkedlist1.FindMiddleNode())


print("Reverse Linked List is : ")
```

```
# linkedlist1.reversedLlinkedlist()

linkedlist1.reversedLInkedlistWithout_stack()
linkedlist1.TraverseLinkedList()
```

```
Inserting eleent into the data
Linked List Traversing :
50->40->30->20->10->
Linked list after the Inserting node at the end :
50->40->30->20->10->60->70->
Middle Node of the linkedlist:  20
Reverse Linked List is :
70->60->10->20->30->40->50->
```

8) How to find the next greater element in array

```
def nextGreaterElement(arr):
  result=[-1]*len(arr)
  for i in range(0,len(arr)):
    for j in range(i+1,len(arr)):
      if arr[i]<arr[j]:
        result[i]=arr[j]
        break
  return result


arr=[1,2,3,4,5]
result=nextGreaterElement(arr)
print("Next greater elemnet of the array is : ",result)
```

```
Next greater elemnet of the array is :  [2, 3, 4, 5, -1]
```

9) reverse word in the string

```
def reverse_words(sentence):
  word=sentence.split()
  reversed_word=word[::-1]
  word=" ".join(reversed_word)
  return word


reverse_words("Hello welcome to the page ")
```

{"type":"string"}

10) print the fibbonasis series of the n numbers

```
def Fibbonasis_series(n):
  fibbonasis=[0,1]
  if n<=0:
    return 0
  elif  n==1 :
    return 1

  else :
    return Fibbonasis_series(n-1)+Fibbonasis_series(n-2)


n=int(input("Enter Number where you wanrt to print the Fibbonasis
Series :"))

fiibio=[Fibbonasis_series(i) for i in range(n+1) ]

print(fiibio)
```

```
Enter Number where you wanrt to print the Fibbonasis Series :10
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```

###11) Write a program to find number is Armstrong or not

```
def isArmstrrong(num):
  original_num=num
  sum=0
  count=len(str(num))
  while num>0:
    digit=num%10
    sum+=digit**count
    num=num//10

  return original_num==sum



num=int(input("Enter the number"))
if isArmstrrong(num):
  print(f"{num} is Armstrong Number")
else :
  print(f"{num} is not Armstrong Number")
```

```
Enter the number153
153 is Armstrong Number
```

12) swap two variables without using third Variable

```
def Swapvariable(num1,num2):

  # This operator perform Bitwise
  num1=num1^num2
  num2=num1^num2
  num1=num1^num2
  return num1,num2

num1=int(input("Enter the first number"))
num2=int(input("Enter the second number"))



print(f"number before swapping num1:{num1} ,num2:{num2}")

num1,num2=Swapvariable(num1,num2)
print(f"Number after Swaping num1:{num1},num2:{num2}")
```

```
Enter the first number12
Enter the second number51
number before swapping num1:12 ,num2:51
Number after Swaping num1:51,num2:12
```

13. Write a program ton print the factorial of the number

```
def Factorial(num):
  if num==0 or num==1:
    return 1
  else:
      # call the factorial function agian and again baese condition is
not occur
    return num*Factorial(num-1)

num=int(input("Enter the number"))
print(f"Factorial of given NUmber:{num} is :",Factorial(num))
```

```
Enter the number5
Factorial of given NUmber:5is : 120
```

13) Write a program to reverse an array

```
def reversearray(arr):
  origial_array=arr
  start=0
  end=len(arr)-1
  while start<end:
    arr[start],arr[end]=arr[end],arr[start]
    start+=1
```

```
    end-=1
  return arr

reverse=[1,2,3,4,5]
print("Enter array is :",reverse)
print("Reverse Array is :",reversearray(reverse))
```

```
Enter array is : [1, 2, 3, 4, 5]
Reverse Array is : [5, 4, 3, 2, 1]
```

###14) Write a proograme to find whether number is prime or not

```
def isPrime(num):
  if num<=1 or num==0:
    return False
  else :
    for i in range(2,num//2):
      if num%i==0:
        return False

  return True

num=int(input("Enter any number"))
if isPrime(num):
  print(f"Number {num} is Prime Number")
else:
  print(f"Number {num} is not Prime number")
```

```
Enter any number104729
Number 104729 is Prime Number
```

15) write a programe to find the square root of the number

```
def findsquarerrot(num):
  if num<0:
    return "Invalid Number"
  else :
    return (num**0.5)
print("Printing the square root of the number by using Exponential 
operator ")
num=int(input("Enter the NUmber"))
print(f"Square root of the number {num} is :",findsquarerrot(num))


# By using Build in Library

import math
def FindSQuareRootByLibrary(num):
  return math.sqrt(num)
```

```
Enter the NUmber4
Square root of the number 4 is : 2.0
```

```python
# How to find next greater element

def nextgreaterelement(arr):
  result=[-1]*len(arr)
  for i in range(len(arr)):
    for j in range (i+1,len(arr)):
      if result[i]<arr[j]:
        result[i]=arr[j]
        break
  return result

arr=[1,2,3,4,5]
nextgreaterelement(arr)
```

```
[2, 3, 4, 5, -1]
```