**Lab Sheet 2**

**Sorting and Recursion**

**Name: Vinayak V Thayil**

**Roll No: AM.EN.U4CSE21161**

Implement the following sorting algorithms and answer the associated questions.

1. Bubble sort

def bubble\_sort(arr):

    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]

arr = []

n = int(input("Enter number of elements : "))

for i in range(0, n):

    ele = int(input())

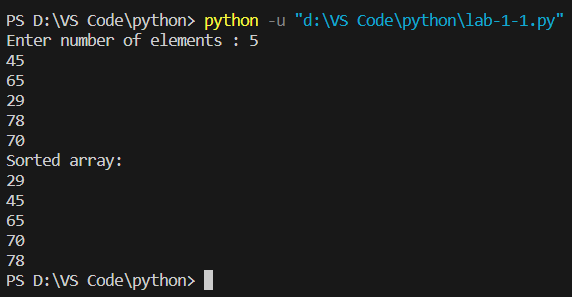
    arr.append(ele)

bubble\_sort(arr)

print ("Sorted array:")

for i in range(len(arr)):

    print ("%d" %arr[i])



* 1. What is the time complexity of a simple bubble sort algorithm? Is there any difference between the best case and the worst case?

**Time Complexity:**

**No Difference**

* 1. How can we change the best-case complexity to Ω(n)? Modify your algorithm accordingly. What is the worst-case complexity of the improved algorithm?

**Using Flag Variable**

def bubble\_sort(arr):

    n = len(arr)

    for i in range(n):

        flag = 0

        for j in range(0, n-i-1):

            if arr[j] > arr[j+1] :

                arr[j], arr[j+1] = arr[j+1], arr[j]

                flag = 1

        if flag == 0:

            break

arr = []

n = int(input("Enter number of elements : "))

for i in range(0, n):

    ele = int(input())

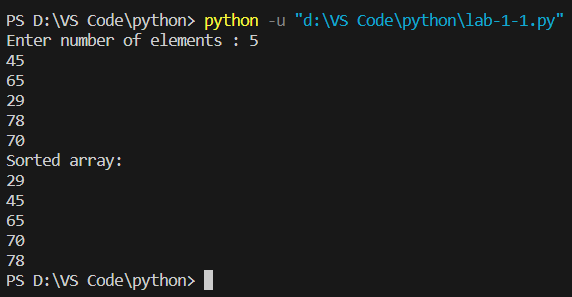
    arr.append(ele)

bubble\_sort(arr)

print ("Sorted array:")

for i in range(len(arr)):

    print ("%d" %arr[i])

****

**Best-Case Time Complexity: Ω(n)**

**Worst-Case Time Complexity:**

* 1. Give examples for best-case and worst-case inputs.

**Best Case: Sorted Array**

**Worst Case: Sorted Array in Descending Order**

1. Selection sort

def selection\_sort():

    values = input("Enter the values: ")

    values = values.split()

    values = [int(x) for x in values]

    print("The values are: ", values)

    for i in range(len(values)):

        min = i

        for j in range(i+1, len(values)):

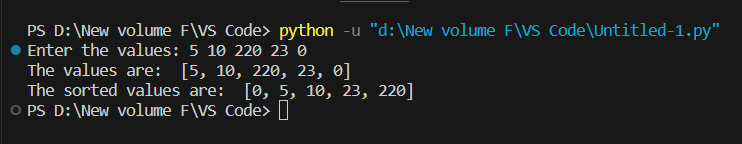
            if values[min] > values[j]:

                min = j

        values[i], values[min] = values[min], values[i]

    print("The sorted values are: ", values)

selection\_sort()



* 1. What is the time complexity of a simple selection sort algorithm? Is there any difference between the best case and the worst case?

**Time Complexity:**

**No Difference**

* 1. How can we change the best-case complexity to Ω(n)? Modify your algorithm accordingly. What is the worst-case complexity of the improved algorithm?

def selection\_sort():

    values = input("Enter the values: ")

    values = values.split()

    values = [int(x) for x in values]

    print("The values are: ", values)

    for i in range(len(values)):

        min = i

        for j in range(i+1, len(values)):

            if values[min] > values[j]:

                min = j

        if min != i:

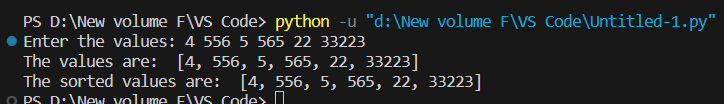
            values[i], values[min] = values[min], values[i]

        else:

            break

    print("The sorted values are: ", values)

selection\_sort()



**Worst Case Time Complexity:**

* 1. Give examples for best-case and worst-case inputs.

**The best-case input for selection sort is an already sorted array.**

**For example: `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`.**

**The worst-case input for selection sort is an array sorted in descending order.**

**For example: `[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]`.**

1. 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]

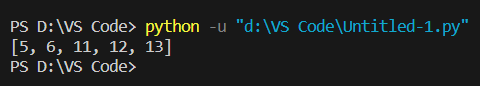
            j -= 1

        arr[j+1] = key

arr = [12, 11, 13, 5, 6]

insertionSort(arr)

print(arr)



* 1. What is the time complexity of a simple selection sort algorithm? Is there any difference between the best case and the worst case?

**Time Complexity:**

* 1. How can we change the best-case complexity to Ω(n)? Modify your algorithm accordingly. What is the worst-case complexity of the improved algorithm?

def insertion\_sort(arr):

    n = len(arr)

    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

        return arr

**Worst Case Complexity:**

* 1. Give examples for best-case and worst-case inputs.

**Best Case 🡪 Sorted Array**

**Worst Case 🡪 Array in descending order**

1. Merge sort

def merge\_sort(arr):

    if len(arr) <= 1:

        return arr

    mid = len(arr) // 2

    left\_half = arr[:mid]

    right\_half = arr[mid:]

    return merge(merge\_sort(left\_half), merge\_sort(right\_half))

def merge(left, right):

    merged = []

    left\_index = 0

    right\_index = 0

    while left\_index < len(left) and right\_index < len(right):

        if left[left\_index] <= right[right\_index]:

            merged.append(left[left\_index])

            left\_index += 1

        else:

            merged.append(right[right\_index])

            right\_index += 1

    merged += left[left\_index:]

    merged += right[right\_index:]

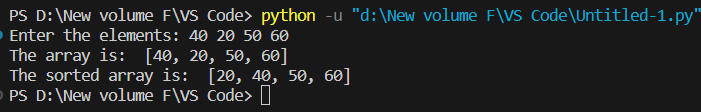
    return merged

arr = list(map(int, input("Enter the elements: ").split()))

print("The array is: ", arr)

sorted\_arr = merge\_sort(arr)

print("The sorted array is: ", sorted\_arr)



1. Quick sort

#Create a program based on Quick sort.

def Quick\_sort():

    print("Enter the Elememts:")

    list = [int(x) for x in input().split()]

    print("The list of numbers is:",list)

    def partition(list,low,high):

        i = (low-1)

        pivot = list[high]

        for j in range(low,high):

            if list[j] <= pivot:

                i = i+1

                list[i],list[j] = list[j],list[i]

        list[i+1],list[high] = list[high],list[i+1]

        return (i+1)

    def quickSort(list,low,high):

        if low < high:

            pi = partition(list,low,high)

            quickSort(list,low,pi-1)

            quickSort(list,pi+1,high)

    n = len(list)

    quickSort(list,0,n-1)

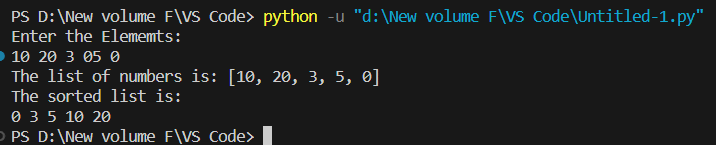
    print("The sorted list is:")

    for i in range(n):

        print(list[i],end=" ")

    print()

Quick\_sort()



1. Sort a set of strings using Radix sort

def radix\_sort(strings):

    max\_length = len(max(strings, key=len))

    strings = [s.ljust(max\_length) for s in strings]

    for i in range(max\_length - 1, -1, -1):

        strings = counting\_sort(strings, i)

    return [s.rstrip() for s in strings]

def counting\_sort(strings, position):

    count = [0] \* 256

    for s in strings:

        count[ord(s[position])] += 1

    for i in range(1, 256):

        count[i] += count[i - 1]

    output = [''] \* len(strings)

    for i in range(len(strings) - 1, -1, -1):

        index = count[ord(strings[i][position])] - 1

        output[index] = strings[i]

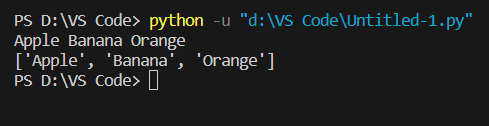
        count[ord(strings[i][position])] -= 1

    return output

input\_strings = input().split()

sorted\_strings = radix\_sort(input\_strings)

print(sorted\_strings



Write Recursive algorithms for the following problems. Implement your algorithm, write the recurrence relation, solve it, and find the asymptotic time complexity

1. Print the sum of the first N natural numbers.
2. Print the product of the first N natural numbers.
3. Print the Nth Fibonacci number.
4. Calculate xy.
5. Print the first N natural numbers.
6. Print the first N natural numbers in reverse order.
7. Find the GCD(HCF) of two numbers.
8. Print the elements of an array.
9. Print the elements of an array in reverse order.
10. Reverse a given number.
11. Check if an array is sorted or not.
12. Write a recursive algorithm to find the median of median in O(n) time.
13. Write a recursive algorithm to find the kth largest element.