**ASSIGNMENT – 1**

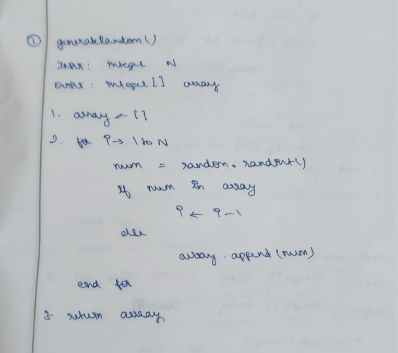
**AIM:**

To solve the given problems using Python and analyze the time complexities of the problems.

**Qn1:**

****

**Psuedo Code:**

****

**Source Code:**

import random

def generateUnique(n):

    l = []

    while len(l) < n:

        num = random.randint(1,10000)

        if num not in l:

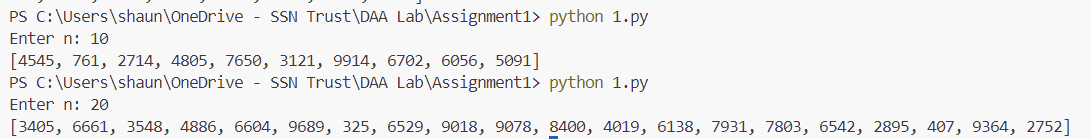
            l.append(num)

    return l

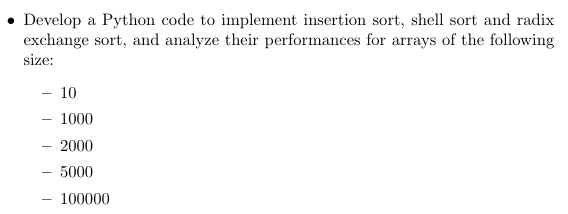
n = int(input("Enter n: "))

print(generateUnique(n))

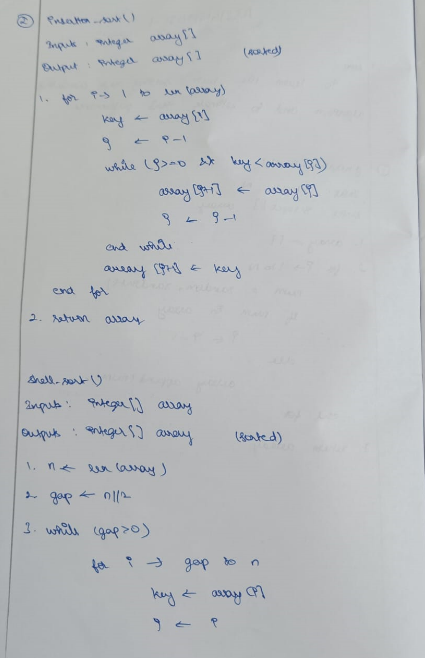
**Output:**

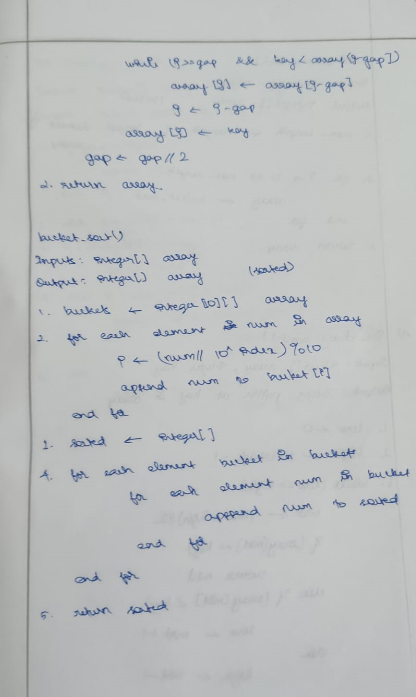
****

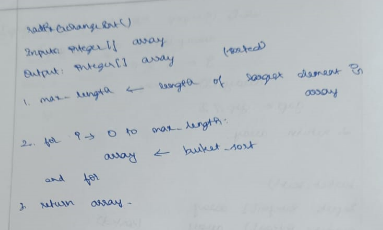
**Qn2:**

****

**Psuedo Code:**

****

****

****

**Source Code:**

import random

import time

def generateList(n):

    l = []

    for i in range(n):

        num = random.randint(1,10000)

        l.append(num)

    return l

def insertionSort(l):

    start = time.time()

    for i in range(len(l)):

        key = l[i]

        j = i-1

        while j>=0 and key<l[j]:

            l[j+1] = l[j]

            j -= 1

        l[j+1] = key

    end = time.time()

    runtime = end - start

    return l,runtime

def shellSort(l):

    gap = len(l)//2

    start = time.time()

    while gap > 0:

        j = gap

        while j<len(l):

            i = j - gap

            while i>=0:

                if l[i+gap] > l[i]:

                    break

                else:

                    l[i+gap], l[i] = l[i], l[i+gap]

                i -=  gap

            j += 1

        gap //= 2

    end = time.time()

    runtime = end - start

    return l, runtime

def countingSort(l, base):

    output = [0] \* len(l)

    count = [0] \* 10

    for i in l:

        index = i//base

        count[index % 10] += 1

    for i in range(1, 10):

        count[i] += count[i-1]

    for i in range(len(l)-1, -1, -1):

        index = l[i]//base

        output[count[index%10]-1] = l[i]

        count[index % 10] -= 1

    for i in range(len(l)):

        l[i] = output[i]

def radixExchangeSort(l):

    start = time.time()

    max\_ = max(l)

    base = 1

    while max\_/base >= 1:

        countingSort(l, base)

        base \*= 10

    end = time.time()

    runtime = end - start

    return l, runtime

n = [10, 1000, 2000, 5000]

for i in n:

    l = generateList(i)

    print("\nInput Size: ", i)

    if i <= 100:

        print("List: ", l)

    print("\nInsertion Sort:")

    sortedInsertion, runtimeInsertion = insertionSort(l)

    if i <= 100:

        print("Sorted List: ", sortedInsertion)

    print("Runtime: ", runtimeInsertion)

    print("\nShell Sort:")

    sortedInsertion, runtimeInsertion = shellSort(l)

    if i<= 100:

        print("Sorted List: ", sortedInsertion)

    print("Runtime: ", runtimeInsertion)

    print("\nRadix Exchange Sort:")

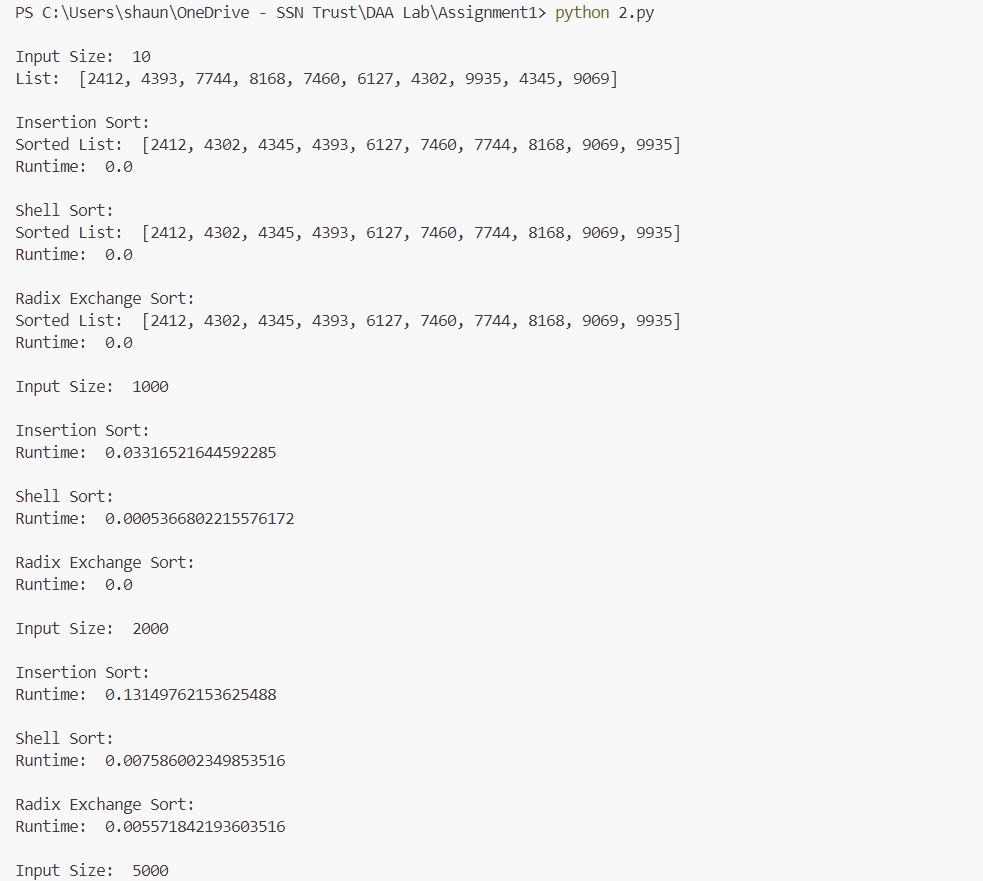
    sortedInsertion, runtimeInsertion = radixExchangeSort(l)

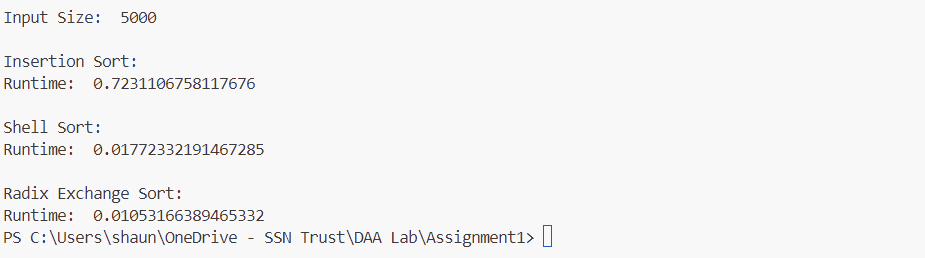
    if i <= 100:

        print("Sorted List: ", sortedInsertion)

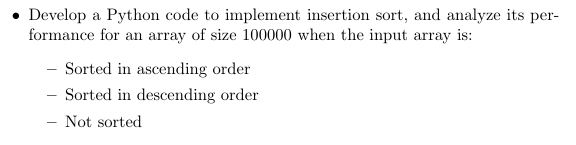
    print("Runtime: ", runtimeInsertion)

**Output:**

****

****

**Qn3:**

****

**Source Code:**

import random

import time

def generateList(n):

    l = []

    for i in range(n):

        num = random.randint(1,10000)

        l.append(num)

    return l

def insertionSort(l):

    start = time.time()

    for i in range(len(l)):

        key = l[i]

        j = i-1

        while j>=0 and key<l[j]:

            l[j+1] = l[j]

            j -= 1

        l[j+1] = key

    end = time.time()

    runtime = end - start

    return l,runtime

l = generateList(10000)

sorted\_l = sorted(l)

sorted\_l\_rev = sorted(l, reverse=True)

print("Runtime Performance")

ascList, ascRuntime = insertionSort(sorted\_l)

print("Sorted in ascending order: ", ascRuntime)

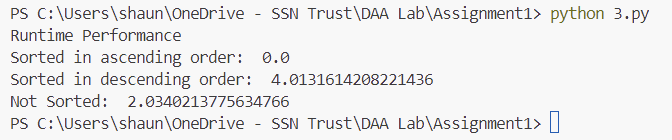
descList, descRuntime = insertionSort(sorted\_l\_rev)

print("Sorted in descending order: ", descRuntime)

notSortList, notSortRuntime = insertionSort(l)

print("Not Sorted: ", notSortRuntime)

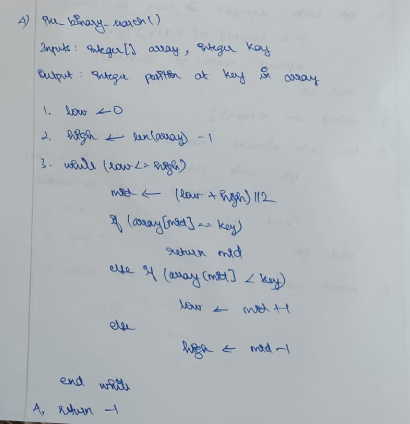
**Output**

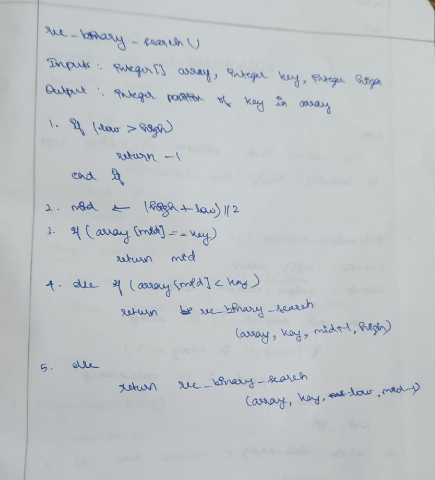
****

**Qn4:**

****

**Psuedo Code:**

****

****

**Source Code:**

import random

import time

def generateList(n):

    l = []

    for i in range(n):

        num = random.randint(1,10000)

        l.append(num)

    return l

def binarySearch(arr, key):

    l = 0

    r = len(arr) - 1

    start = time.time()

    while l<=r:

        m = (l+r)//2

        if key < arr[m]:

            r = m-1

        elif key > arr[m]:

            l = m+1

        else:

            runtime = time.time() - start

            return m, runtime

def binarySearchRecursive(arr, key, l, r, runtime):

    m = (l+r)//2

    start = time.time()

    if key < arr[m]:

        return binarySearchRecursive(arr, key, l, m-1, runtime+time.time()-start)

    elif key > arr[m]:

        return binarySearchRecursive(arr, key, m+1, r, runtime+time.time()-start)

    else:

        return m, runtime

l = generateList(100000)

l.sort()

key = random.choice(l)

print("Performnce Comparison")

nonRecIndex, nonRecRuntime = binarySearch(l, key)

print("\nNon-Recursive Binary Search Result: ", nonRecIndex)

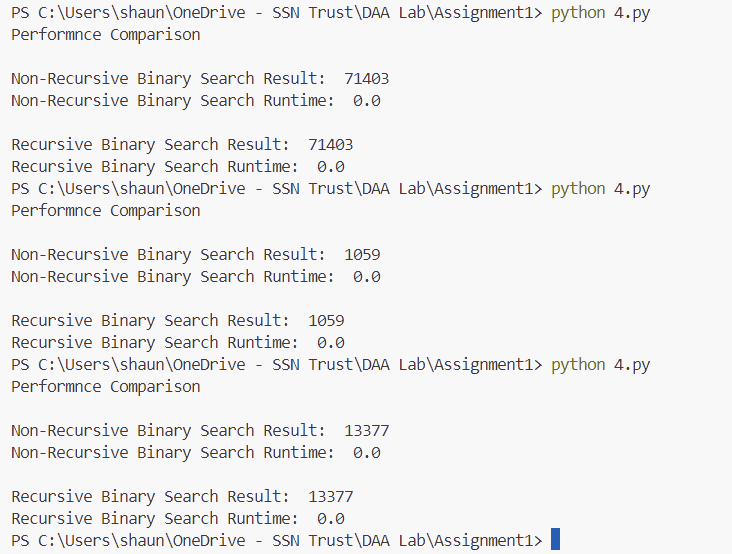
print("Non-Recursive Binary Search Runtime: ", nonRecRuntime)

recIndex, recRuntime = binarySearchRecursive(l, key, 0, len(l)-1, 0)

print("\nRecursive Binary Search Result: ", recIndex)

print("Recursive Binary Search Runtime: ", recRuntime)

**Output:**

****

**Learning Outcomes:**

* I learnt to analyse the time complexities of various algorithms
* I learnt how to implement various sorting and searching algorithms in Python