Date: 24.02.24 Reg No: 3122225001127
ExNo: 1 Name: Shaun Allan H

ASSIGNMENT – 1

AIM:

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

Qn1:

- Develop a Python code that takes as input a value n, and generates a list of n unique random values. [CO1,K3]

Psuedo Code:

```
① generateRandom()
Input : integer N
Output : integer [] array

1. array ← []
2. for i -> 1 to N
        num = random.randint()
        if num in array
                i ← i-1
        else
                array.append(num)
   end for
3. return array
```

Date: 24.02.24
ExNo: 1

Reg No: 3122225001127
Name: Shaun Allan H

Source Code:

```python
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:

```
PS C:\Users\shaun\OneDrive - SSN Trust\DAA Lab\Assignment1> python 1.py
Enter n: 10
[4545, 761, 2714, 4805, 7650, 3121, 9914, 6702, 6056, 5091]
PS C:\Users\shaun\OneDrive - SSN Trust\DAA Lab\Assignment1> python 1.py
Enter n: 20
[3405, 6661, 3548, 4886, 6604, 9689, 325, 6529, 9018, 9078, 8400, 4019, 6138, 7931, 7803, 6542, 2895, 407, 9364, 2752]
```

Date: 24.02.24
ExNo: 1

Reg No: 3122225001127
Name: Shaun Allan H

Qn2:

- Develop a Python code to implement insertion sort, shell sort and radix exchange sort, and analyze their performances for arrays of the following size:
  - 10
  - 1000
  - 2000
  - 5000
  - 100000

Psuedo Code:

```
(2) Insertion_sort()
Input : integer array[]
Output : integer array[]        (sorted)

1. for i -> 1 to len(array)
       key <- array[i]
       j <- i-1
       while (j>=0 && key < array[j])
           array[j+1] <- array[j]
           j <- j-1
       end while
       array[j+1] <- key
   end for
2. return array


Shell_sort()
Input : integer[] array
Output : integer[] array        (sorted)
1. n <- len(array)
2. gap <- n//2
3. while (gap>0)
       for i -> gap to n
           key <- array[i]
           j <- i
```

```
            while (j>=gap && key < array[j-gap])
                  array[j] ← array[j-gap]
                  j ← j-gap
            array[j] ← key
      gap ← gap//2
2. return array.


bucket_sort()
Input: integer[] array
Output: integer[] array      (sorted)
1. bucket ← integer[10][] array
2. for each element num in array
         i ← (num// 10^order)%10
         append num to bucket[i]
   end for
3. sorted ← integer[]
4. for each element bucket in bucket
         for each element num in bucket
               append num to sorted
         end for
   end for
5. return sorted
```

```
radix_exchange_sort()
Input: integer[] array
Output: integer[] array      (sorted)
1. max_length ← length of larger element in
                                        array
2. for i → 0 to max_length
         array ← bucket_sort
   end for
3. return array.
```

Date: 24.02.24
ExNo: 1

Reg No: 3122225001127
Name: Shaun Allan H

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:

```
PS C:\Users\shaun\OneDrive - SSN Trust\DAA Lab\Assignment1> python 2.py

Input Size:  10
List:  [2412, 4393, 7744, 8168, 7460, 6127, 4302, 9935, 4345, 9069]

Insertion Sort:
Sorted List:  [2412, 4302, 4345, 4393, 6127, 7460, 7744, 8168, 9069, 9935]
Runtime:  0.0

Shell Sort:
Sorted List:  [2412, 4302, 4345, 4393, 6127, 7460, 7744, 8168, 9069, 9935]
Runtime:  0.0

Radix Exchange Sort:
Sorted List:  [2412, 4302, 4345, 4393, 6127, 7460, 7744, 8168, 9069, 9935]
Runtime:  0.0

Input Size:  1000

Insertion Sort:
Runtime:  0.03316521644592285

Shell Sort:
Runtime:  0.0005366802215576172

Radix Exchange Sort:
Runtime:  0.0

Input Size:  2000

Insertion Sort:
Runtime:  0.13149762153625488

Shell Sort:
Runtime:  0.007586002349853516

Radix Exchange Sort:
Runtime:  0.005571842193603516

Input Size:  5000
```

```
Input Size:  5000

Insertion Sort:
Runtime:  0.7231106758117676

Shell Sort:
Runtime:  0.01772332191467285

Radix Exchange Sort:
Runtime:  0.01053166389465332
PS C:\Users\shaun\OneDrive - SSN Trust\DAA Lab\Assignment1>
```

Date: 24.02.24
ExNo: 1

Reg No: 3122225001127
Name: Shaun Allan H

Qn3:

- Develop a Python code to implement insertion sort, and analyze its performance for an array of size 100000 when the input array is:
  - Sorted in ascending order
  - Sorted in descending order
  - Not sorted

Source Code:

```python
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

```
PS C:\Users\shaun\OneDrive - SSN Trust\DAA Lab\Assignment1> python 3.py
Runtime Performance
Sorted in ascending order:  0.0
Sorted in descending order:  4.0131614208221436
Not Sorted:  2.0340213775634766
PS C:\Users\shaun\OneDrive - SSN Trust\DAA Lab\Assignment1>
```

Date: 24.02.24 Reg No: 3122225001127
ExNo: 1 Name: Shaun Allan H

Qn4:

- Compare the performances of recursive and non-recursive algorithms for binary search using an array of size 100000. [CO1,K2]

Psuedo Code:

```
4) ite_binary_search()
Inputs: integer[] array, integer key
Output: integer position of key in array

1. low <- 0
2. high <- len(array) - 1
3. while (low <= high)
       mid <- (low + high) // 2
       if (array[mid] == key)
           return mid
       else if (array[mid] < key)
           low <- mid + 1
       else
           high <- mid - 1
   end while
4. return -1
```

rec_binary_search()

Input : Integer[] array, Integer key, Integer high

Output : Integer position of key in array

1. if (low > high)
   return -1
   end if
2. mid ← (high + low)//2
3. if (array[mid] == key)
   return mid
4. else if (array[mid] < key)
   return ~~be~~ rec_binary_search (array, key, mid+1, high)
5. else
   return rec_binary_search (array, key, ~~mi~~ low, mid-1)

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:

```
PS C:\Users\shaun\OneDrive - SSN Trust\DAA Lab\Assignment1> python 4.py
Performnce Comparison

Non-Recursive Binary Search Result:  71403
Non-Recursive Binary Search Runtime:  0.0

Recursive Binary Search Result:  71403
Recursive Binary Search Runtime:  0.0
PS C:\Users\shaun\OneDrive - SSN Trust\DAA Lab\Assignment1> python 4.py
Performnce Comparison

Non-Recursive Binary Search Result:  1059
Non-Recursive Binary Search Runtime:  0.0

Recursive Binary Search Result:  1059
Recursive Binary Search Runtime:  0.0
PS C:\Users\shaun\OneDrive - SSN Trust\DAA Lab\Assignment1> python 4.py
Performnce Comparison

Non-Recursive Binary Search Result:  13377
Non-Recursive Binary Search Runtime:  0.0

Recursive Binary Search Result:  13377
Recursive Binary Search Runtime:  0.0
PS C:\Users\shaun\OneDrive - SSN Trust\DAA Lab\Assignment1>
```

Learning Outcomes:

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