Lab Sheet 1

Basic iterative programs

1. Write a program to find the smallest element of an unsorted array of size N.
   Don't use any built-in functions/ methods like min() supported by certain programming languages.
   a. How many times does your loop execute?
   b. As the elements in the array changes (the size of the array remains same), will there be any change in the number of times the loop executes? What is the minimum and maximum number of times the loop executes?
   c. What is the time complexity of your program?

```
def minArr(arr):
    minval = arr[0]
    for i in arr:
        if i < minval:
            minval = i
    return minval


arr = [5, 3, 1, 2, 7]
minval = minArr(arr)
print("Smallest element:", minval)
```

```
Smallest element: 1
```

a. The Loop executes N times, where N is the length of the input Array

b. The Loop will always execute N times, no matter the values of the Array

c. Time Complexity: O(N)

2. Write a program to find the smallest and the largest element in a sorted array.
   a. Do we need any loops in this program?
   b. What is the time complexity of your program?

```
def MinMaxArr(arr):
    return arr[0], arr[-1]


arr = [1, 2, 3, 4, 5]
Min, Max = MinMaxArr(arr)
print("Smallest element:", Min)
print("Largest element:", Max)
```

```
Smallest element: 1
Largest element: 5
```

a. No, you can access the Minimum Value at index 0 and Maximum Value at index N-1(or -1 if the language supports reverse indexing)

b. Time Complexity: O(1)

3. Write a program to search an element 'k' in an unsorted array of size N.
   a. As the elements in the array changes (the size of the array remains same), will there be any change in the number of times the loop executes? What is the minimum and maximum number of times the loop executes?
   b. What is the time complexity of your program?

```
def searchArr(k, arr):
  for i, val in enumerate(arr):
    if k==val:
      return i

arr = [20, 14, 5, 6, 8, 91, -3, 42]
k = 5
if searchArr(k, arr) is not None:
  print(f"Element {k} is found at Index {searchArr(k, arr)}")
else:
  print(f"Element {k} is NOT found")
```

```
Element 5 is found at Index 2
```

a. As the Element changes, the number of execution changes, Minimum number of execution=1, Maximum Number of execution=N

b. Time Complexity: O(N)

4. We need to search an element 'k' in a sorted array of size N.
   a. Will your program for Qn. No 3 work for this case?
   b. Is the program for Qn. No 3 the most efficient one for this? (Hint: There exists a Binary search algorithm)
   c. Write an iterative program to implement Binary search.
   d. As the elements in the array changes (the size of the array remains same), will there be any change in the number of times the loop executes? What is the minimum and maximum number of times the loop executes?
   e. What is the time complexity of your program?

```
def searchsrtdArr(k, arr):
  l, r = 0, len(arr) - 1

  while l <= r:
    m = (l + r) // 2
    if arr[m] == k:
      return m
    elif arr[m] < k:
      l = m + 1
    else:
      r = m - 1


arr = [1, 3, 5, 7, 9, 11, 13]
k=9
i = searchsrtdArr(k, arr)
if i is not None:
    print(f"Found {k} at index {i}.")
else:
    print(f"{k} not found in the array.")
```

```
Found 9 at index 4.
```

a. Yes, it will work since it is a linear search Function, but it is less efficient provided it is a sorted array

b. No, Binary search is more efficient for Sorted Arrays

c. Function searchsrtdArr(k, arr) is a version of Binary search where k is the target value that is searched in the sorted Array arr, if the target is not found it returns None

d. Min=1, Max=log2(N)

e. Time Complexity: O(log2(N))

5. Write an efficient program to find an element in an array which neither the smallest nor the largest. (Hint: you can do this without a loop.)
   a. What is the time complexity of your program?

```
def q5(arr):
```

```
  arr.sort()
  return arr[1] if len(arr) > 2 else None

arr = [5, 2, 8, 1, 9, 4]
print(f"The Middle Element of the Array {arr} is {q5(arr)}")
```

```
The Middle Element of the Array [5, 2, 8, 1, 9, 4] is 2
```

a. Time Complexity: O(N*log(N))

6. Write an efficient program to check if a given number is prime or not.
   a. What is the time complexity of the algorithm?
   b. Show that the problem can be solved in root(n) time.

```
def isPrime(x):
  for i in range(2, int(x**0.5)):
    if x%i == 0:
      return False
  return True

x = 17

if isPrime(x):
  print(f"The Number X={x} is a Prime Number")
else:
  print(f"The Number X={x} is a NOT Prime Number")
```

```
The Number X=17 is a Prime Number
```

a. Time Complexity: O(√N)

b. The above function isPrime(x) is better than the brute force method which is O(N). this function only iterates until the square root of the number, since there is no factor that exists which is greater than the square root of the number itself

7. Write an efficient program to find the GCD (also called HCF) of two given numbers.
   a. What is the time complexity of the algorithm?
   b. Find an input that requires maximum number of iterations to solve.

```
def GCD(a, b):
  for i in range(min(a, b), 0, -1):
    if a%i == b%i == 0:
      return i

print(GCD(60, 120))
```

```
60
```

a. Time Complexity: O(N), where N is the smallest of a and b

b. Choose two prime Numbers, their HCF or GCD is 1, for example GCD(17, 23)

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.