* Totem Interactive *
:- Developer Assignment

Page No. 1
Date 05 01 25

Name :- Shivam Prajapati
Contact :- 8928988334 / prajapatishivam7634@gmail.com

Part 1: Hand-written

1. Algorithmic Thinking:
Write pseudo-code for a function that finds the first n prime numbers.

•) Solution

Step 1. Input Number of Primes (n)
-> Ask user to input a positive integer (n).

Step 2. Initialize Variables
-> Create an empty list "primes" to store prime number
-> Set "number" to 2, the smallest prime number

Step 3. Start While Loop
-> Repeat until the size of the "primes" list is equal to "n"

Step 4. Assume Current Number is Prime
-> Set "isPrime" to True

Step 5. Check Divisibility
-> For each divisor from 2 to the square root of "number"
-> If number % divisor == 0
-> Set isPrime to False and exit the loop

Step 6 Add Prime to List
-> If "isPrime" is True after the loop, append "number" to the "primes" list.

Step 7. Increment Number
-> increase "number" by "1".

Step 8. End While Loop
-> Continue until "n" prime numbers are found

Step 9 Output the Primes
-> Display the "primes" list to the user.

•) Code

```
def first_n_primes (n):
    primes = []
    number = 2

    while len(primes) < n:
        is_prime = True

        for divisor in range (2, math.sqrt(number) + 1):
            if number % divisor == 0:
                is_prime = False
                break

        if is_prime:
            primes.append(number)

        number += 1

    return primes
```

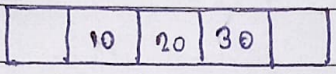
2. Data Structure

Write pseudo-code for a function that implements a basic Queue with enqueue and dequeue operations.

•) Solution

Note: ① Enqueue - Insertion of item from REAR end
② Dequeue - Deletion of item from FRONT end

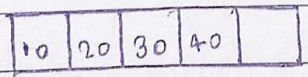
① Pseudo-code for enqueue (Algorithm)



Rear = 3, Front = 1, N = 5

Step 1: Begin
Step 2: if Rear = N-1 then print "Overflow & exit"
Step 3: Input new item
Step 4: Rear ← Rear + 1
Step 5: Queue [Rear] ← item
Step 6: Exit

② Pseudo-code for dequeue (Algorithm)



Front = 0, N = 5, Rear = 3

Step 1. Begin
Step 2. if Front = -1 then print "Underflow & exit"
Step 3. Set item ← Queue [Front]
Step 4. Front ← Front + 1
Step 5: print "item deleted"
Step 6: Exit

•) Code for enqueue and dequeue

```
#define Max_Size 5
int queue[Max_Size];
int front = -1;
int rear = -1;

// enqueue
if (rear == Max_Size - 1) {
    print ("Overflow & Exit");
} else {
    int ~~input~~ item = input ("Enter a number/element");
    if (front == -1) {
        front = rear = 0;
    } else {
        rear ++;
    }
    Queue[rear] = item;
    print ("Item inserted");
}

// dequeue
if (front == -1) {
    print ("Underflow & exit");
} else {
    item = queue[front];
    if (front == rear) {
        front = rear = -1;
    } else {
        front ++;
    } print ("Item Deleted");
}
```

3. Recursion

Write pseudo-code for finding the factorial of number using recursion

•) Solution

```
Function factorial(n):
    If n <= 1;
        Return 1
    Else:
        Return n * factorial (n-1)
```

•) Example

-> Find factorial of 4

```
factorial (4) = 4 * factorial (3);
              = 4 * 3 * factorial (2);
              = 4 * 3 * 2 * factorial (1);
              = 4 * 3 * 2 * 1;
```

4. Sorting Algorithm

Write pseudo-code for Bubble Sort Algorithm

•) Solution

```
Int Bubble_Sort ( int arr [ ] )
{
   int N = arr.length ;
   int temp ;
  for (int i = N-1 ; i >= 1 ; i--)
  {
     for (int j = 0 ; j < i ; j++ )
     { if (arr [j] > arr [j+1] )
        { // Swap
           temp = arr [j];
           arr [j] = arr [j+1];
           arr [j+1] = temp;
        }
     }
  }
   return arr ;
}
```

•) Note :- Bubble Sort always arrange the data in the descending order. And gives the output of array in the ascending order.

5. Searching Algorithms

Write pseudo-code for a function that performs binary search on sorted array.

•) Solution

Note: Binary search is apply on sorted array and if array is not sorted then we have to sort the array then apply Binary Search.

•) Pseudo-code for Binary Search

```
Procedure BinarySearch (arr, n, target):
    left = 0
    right = n-1

    While left <= right:
        mid = (left + right) // 2
        If arr[mid] == target:
            Return mid
        Else If arr[mid] < target:
            left = mid + 1
        Else:
            right = mid - 1
    End while

    Return -1  // Target not found.
```

•) arr :- A sorted array
•) n :- Size of the array
•) target :- The value to search for.