

```

def compute(first_input, second_input):
    counter = 0
    if len(first_input) < len(second_input):
        length_shortest_input = len(first_input)
    else:
        length_shortest_input = len(second_input)
    for i in range(length_shortest_input):
        if first_input[i] == second_input[i]:
            counter += 1
    return counter

```

3

Figure 1: commonCharsMN

```

def compute(array: list[int], key: int) -> int:
    index1: int = 0
    index2: int = len(array) - 1
    while index1 <= index2:
        m: int = (index1 + index2) // 2
        if key < array[m]:
            index2 = m - 1
        elif key > array[m]:
            index1 = m + 1
        else:
            return m
    return -1

```

4

Figure 2: binarySearchMT

```
def compute(numbers: list[int]) -> str:  
    result: list[float] = [0.0] * len(numbers)  
    for i in range(len(numbers)):  
        if numbers[i] == 0:  
            result[i] = 0.0  
            continue  
        if numbers[i] < 0:  
            result[i] = math.sqrt(-1 * numbers[i])  
        else:  
            result[i] = math.sqrt(numbers[i])  
    return str(result)
```

2

Figure 3: squareRootMT

```
def compute(number: str) -> int:  
    if number == "0":  
        return 0  
    if number == "1":  
        return 1  
    if number[-1] == "0":  
        return 2 * compute(number[:-1])  
    if number[-1] == "1":  
        return 1 + 2 * compute(number[:-1])  
    return -1
```

2

Figure 4: binaryToDecimalMT

```

def compute(number):
    if number == 0:
        return 0
    return (number % 10) + compute(number // 10)

```

1

Figure 5: crossSumMN

```

def compute(string: str, start: int, end: int) -> int:
    result: int = 0
    keys: list[str] = string.split(" ")
    for i in range(len(keys)):
        key: int = int(keys[i])
        check: bool = (key >= start and key <= end)
        if check:
            result += 1
    return result

```

2

Figure 6: countIntegerIntervalMT

```
def compute(sentence):
    stack = []
    for i in sentence:
        if i == '(':
            stack.append(')')
        elif i == '{':
            stack.append('}')
        elif i == '[':
            stack.append(']')
        elif len(stack) == 0 or stack.pop() != i:
            return False
    return len(stack) == 0
```

2

Figure 7: validParenthesesMN

```
def compute(input):
    flag = True
    for i in range(2, (input // 2) + 1):
        if input % i == 0:
            flag = False
    return flag
```

3

Figure 8: primeMN

```
def compute(word: str, letters: list[str]) -> int:  
    letterCount: int = 0  
    for i in range(len(word)):  
        for j in range(len(letters)):  
            if word[i] == letters[j]:  
                letterCount += 1  
    return letterCount
```

2

Figure 9: countLettersMT

```
def compute(word: str) -> bool:  
    result: bool = True  
    for i in range(0, len(word) // 2):  
        j: int = len(word) - 1 - i  
        if word[i] != word[j]:  
            result = False  
            break  
    return result
```

3

Figure 10: palindromeMT

```
def compute(numbers: list[int]) -> float:  
    counter: int = 0  
    sum: int = 0  
    while counter < len(numbers):  
        sum = sum + numbers[counter]  
        counter = counter + 1  
    result: float = sum / counter  
    return result
```

1

Figure 11: arrayAverageMT

```
def compute(sentence: str) -> str:  
    result: list[int] = []  
    words: list[str] = sentence.split()  
    for i in range(len(words)):  
        if i > 0:  
            result.append(" ")  
        result.append(words[i][0].upper() + words[i][1:])  
    return "".join(result)
```

2

Figure 12: capitalizeFirstLetterMT

```

def compute(word: str, substring: str) -> bool:
    for i in range(len(word)):
        for j in range(len(substring)):
            if i + j >= len(word):
                break
            if word[i + j] != substring[j]:
                break
            else:
                if j == len(substring) - 1:
                    return True
    return False

```

~~1~~  
2

Figure 13: containsSubstringMT

```

def compute(value):
    if value == 1:
        return 1
    return compute(value - 1) * value

```

1

Figure 14: factorialMN

```
def compute(array: list[int]) -> list[int]:
    for i in range(len(array)):
        for j in range(i, 0, -1):
            if array[j-1] > array[j]:
                array[j-1], array[j] = array[j], array[j-1]
    return array
```

Figure 15: bubbleSortMT

```
def compute(number1: int, number2: int) -> int:
    result: int = number1 * number2
    for i in range(1, number1 * number2):
        if i % number1 == 0 and i % number2 == 0:
            result = i
            break
    return result
```

Figure 16: leastCommonMultipleMT

```
def compute(a: int, b: int) -> int:  
    if b == 0:  
        return 1  
    if b == 1:  
        return a  
    return a * compute(a, b - 1)
```

1

Figure 17: powerMT

```
def compute(input):  
    a = ""  
    b = ""  
    for i in range(len(input) - 1, -1, -1):  
        a = input[i] + a  
        b = b + input[i]  
    return a + b
```

2

Figure 18: forwardBackwardMN

```
def compute(array, x):
    for i in range(len(array)):
        if array[i] == x:
            return i
    return -1
```

1

Figure 19: linearSearchMN

```
def compute(a, b, c, d):
    if a > b:
        b, a = a, b
    if c > d:
        d, c = c, d
    if a > c:
        c, a = a, c
    if b > d:
        d, b = b, d
    if b > c:
        c, b = b, c
    return [a, b, c, d]
```

3

Figure 20: unrolledSortMN