

Difficulty Rating : 1-5
easy hard

62038

3
also (because
first one)

```
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
```

Figure 1: countLettersMT

2

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

Figure 2: primeMN

```

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

```

Figure 3: binarySearchMN

```

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 4: bubbleSortMT

1

```
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
```

Figure 5: arrayAverageMT

5

```
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
```

if-statements are
seemingly
unconnected

Figure 6: containsSubstringMT

```

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

```

Figure 7: palindromeMT

```

def compute(a: int, b: int, c: int, d: int) -> list[int]:
    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]

```

Figure 8: unrolledSortMT

1

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

Figure 9: factorialMN

2

```
def compute(numbers):  
    result = [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)
```

looks weird in the
beginning, but pretty
easy once you work
on it in detail

Figure 10: squareRootMN

3

```
def compute(sentence):
    result = []
    words = 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)
```

Figure 11: capitalizeFirstLetterMN

3

```
def compute(sentence: str) -> bool:
    stack: list[str] = []
    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
```

Figure 12: validParenthesesMT

```
def compute(number: int) -> int:
    if number == 0:
        return 0
    return (number % 10) + compute(number // 10)
```

Figure 13: crossSumMT

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

Figure 14: leastCommonMultipleMN

2

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

Figure 15: forwardBackwardMT

2

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

long variable
names "clutter"
the code making
it harder to
read.

Figure 16: commonCharsMT

2

```
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
```

Figure 17: countIntegerIntervalMT

1

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

Figure 18: linearSearchMT

1

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

Figure 19: powerMN

2

```
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
```

Figure 20: binaryToDecimalMT