Algorithm Plan

```
def is_palindrome(string):
    reversed_string = "".join(reversed(string))
    return string == reversed_string
def palindrome_max_sub_string(string_dirty):
    string = [c for c in string_dirty.lower() if c.isalnum()]
    string = "".join(string)
    maximum_palindrome_len = 0
    palindrome_word = ""
    for idx in range(len(string)):
        for idy in range(1, len(string) + 1):
            if is_palindrome(string[idx:idy]):
                if len(string[idx:idy]) > maximum_palindrome_len:
                    max_palindrome_len = len(string[idx:idy])
                    palindrome_word = string[idx:idy]
    return palindrome_word, maximum_palindrome_len
```

```
def move_zeroes_to_end(string):
    zeroes = [c for c in string if c == "0"]
    non_zeroes = [c for c in string if c != "0"]
    return "".join(non_zeroes + zeroes)
```

```
def fibonacci(n):
    if n == 0:
        return 1
    if n == 1:
        return 1
    return fibonacci(n - 2) + fibonacci(n - 1)
for n in range(10):
    print(fibonacci(n))
```

```
def fibonacci_without_recursion(n):
    a = 0
    b = 1
    output = []
    for _ in range(n):
        output.append(b) # Update fibonacci sequence
        a, b = b, a + b      # b is the current fibonacci number
    return output
```

```
def fibonacci_without_recursion_yield(n):
  a, b = 0, 1
  for _ in range(n):
    yield b
    a, b = b, a + b
```

```
def maximum_minimum(nums):
    return max(nums), min(nums)
```

```
def sub_array_highest_sum(arr):
    highest_sum = float('-inf')
    output = []
    for idx in range(len(arr)):
        for idy in range(idx + 1, len(arr) + 1):
            current_sum = sum(arr[idx:idy])
            if current_sum > highest_sum:
                output = arr[idx:idy]
                highest_sum = current_sum
    return f" The subarray {output} has the largest sum {highest_sum}"
time O(n^2),  space O(1)
```

```
import sys   System Argument
if len(sys.argv) > 1:
    name = sys.argv[1]
    print(f"Hello, {name}!")
else:
    print("Please provide a name as an argument.")
```

```
# Definition for singly-linked list.
class ListNode:
   def __init__(self, val=0, next=None):
      self.val = val
      self.next = next
class Solution:
    def mergeTwoLists(self, list1: [ListNode], list2: [ListNode]) ->
[ListNode]:
        dummy = ListNode(0)
        output = dummy
        current1 = list1
        current2 = list2
        while current1 is not None and current2 is not None:  # Not .next
            if current1.val < current2.val:
                output.next = ListNode(current1.val)
                current1 = current1.next
            else:
                output.next = ListNode(current2.val)
                current2 = current2.next
            output = output.next
        output.next = current1 if current1 else current2 # Append rmd
        return dummy.next
```

```
    # Inserting: We need to stop before the last node to attach the new
node.
    def insert_a_node_at_the_end(self, head, num):
        new_node = ListNode(num)
        if head is None:
            head = new_node
            return head
        current = head
        while current.next is not None:
            current = current.next
        current.next = new_node

    # Printing: We need to process every node, so we include the last
node in the loop.
    def print_a_list(self, head):
        if head is None:
            print("List is empty")
            return
        current = head
        while current is not None:
            print(current.val, end="->")
            current = current.next # Note: current points to a current
List Node, while current.next points to the next one
        print("None")
```

```
def twoSum(self, nums, target: int):
    output = []
    for idx in range(0, len(nums)-1):
        for idy in range(idx+1, len(nums)):
            if (nums[idx] + nums[idy]) == target:
                output.append(idx)
                output.append(idy)
    return output
```

```
from collections import Counter
def can_construct(ransomNote, magazine):
    char_count = Counter(magazine)  # Create a Counter for magazine
    for char in ransomNote:  # Iterate chars in the ransomNote
        if char_count[char] > 0:  # Check if char in magazine
            char_count[char] -= 1  # Decrement the count of the character
        else:
            return False  # If the character is not present or its count is
zero, return False
    return True  # If all characters are successfully matched, return True
```