```
def find_missing_number(nums):
    """
    Finds the missing number in an array of size n, containing numbers 1 to n+1.

    Args:
        nums: A list of integers.

    Returns:
        The missing integer.
    """
    n = len(nums)
    expected_sum = (n + 1) * (n + 2) // 2
    actual_sum = sum(nums)
    return expected_sum - actual_sum

def check_balanced_parentheses(s):
    """
    Checks if a string of parentheses ((), {}, []) is balanced.

    Args:
        s: A string of parentheses.

    Returns:
        True if balanced, otherwise False.
    """
    stack = []
    mapping = {')': '(', '}': '{', ']': '['}
    for char in s:
        if char in mapping:
            top_element = stack.pop() if stack else '#'
            if mapping[char] != top_element:
                return False
        else:
            stack.append(char)
    return not stack

def longest_word_in_sentence(sentence):
    """
    Finds the longest word in a given sentence.

    Args:
        sentence: A string (sentence).

    Returns:
        The longest word.
    """
    words = sentence.split()
    longest_word = ""
    for word in words:
        if len(word) > len(longest_word):
            longest_word = word
    return longest_word

def count_words_in_sentence(sentence):
    """
    Counts the number of words in a sentence.

    Args:
        sentence: A string (sentence).

    Returns:
        Integer representing the word count.
    """
    words = sentence.split()
    return len(words)

def check_pythagorean_triplet(a, b, c):
    """
    Determines if three numbers form a Pythagorean triplet.

    Args:
        a: First integer.
        b: Second integer.
        c: Third integer.

    Returns:
        True if they form a Pythagorean triplet, otherwise False.
    """
    sides = sorted([a, b, c])
    return sides[0]**2 + sides[1]**2 == sides[2]**2

# Example Usage and Outputs:
print("Find Missing Number:")
print(find_missing_number([1, 2, 4, 5]))  # Output: 3
print(find_missing_number([1, 3, 4, 5, 6])) # output: 2

print("\nCheck Balanced Parentheses:")
print(check_balanced_parentheses("(){}[]"))  # Output: True
print(check_balanced_parentheses("({[)]}"))  # Output: False
print(check_balanced_parentheses("((()))")) # output: true
print(check_balanced_parentheses("([)]")) # output: false

print("\nLongest Word in a Sentence:")
print(longest_word_in_sentence("This is a sample sentence"))  # Output: sentence
print(longest_word_in_sentence("The quick brown fox jumps over the lazy dog")) # output: quick


print("\nCount Words in a Sentence:")
print(count_words_in_sentence("This is a sample sentence"))  # Output: 5
print(count_words_in_sentence("one two three four")) # output: 4


print("\nCheck Pythagorean Triplet:")
print(check_pythagorean_triplet(3, 4, 5))  # Output: True
print(check_pythagorean_triplet(1, 2, 3))  # Output: False
print(check_pythagorean_triplet(5,12,13)) # output: true


def bubble_sort(nums):
    """
    Implements the bubble sort algorithm.


    Args:
        nums: A list of integers.


    Returns:
        Sorted list in ascending order.
    """
    n = len(nums)
    for i in range(n):
        for j in range(0, n - i - 1):
            if nums[j] > nums[j + 1]:
                nums[j], nums[j + 1] = nums[j + 1], nums[j]
    return nums


def binary_search(nums, target):
    """
    Implements the binary search algorithm.

    Args:
        nums: A sorted list of integers.
        target: The target integer.

    Returns:
        The index of the target or -1 if not found.
    """
    left, right = 0, len(nums) - 1
    while left <= right:
        mid = (left + right) // 2
        if nums[mid] == target:
            return mid
        elif nums[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
    return -1

def find_subarray_with_given_sum(nums, s):
    """
    Finds a contiguous subarray whose sum equals a given value S.

    Args:
        nums: A list of integers.
        s: The target sum.

    Returns:
        The indices of the subarray or -1 if no such subarray exists.
    """
    n = len(nums)
    for i in range(n):
        current_sum = nums[i]
        if current_sum == s:
            return [i, i]
        for j in range(i + 1, n):
            current_sum += nums[j]
            if current_sum == s:
                return [i, j]
            elif current_sum > s:
                break
    return -1

# Example Usage and Outputs:
print("Bubble Sort:")
print(bubble_sort([64, 34, 25, 12, 22, 11, 90]))  # Output: [11, 12, 22, 25, 34, 64, 90]
print(bubble_sort([5, 1, 4, 2, 8])) # output: [1, 2, 4, 5, 8]

print("\nBinary Search:")
print(binary_search([2, 5, 8, 12, 16, 23, 38, 56, 72, 91], 23))  # Output: 5
print(binary_search([2, 5, 8, 12, 16, 23, 38, 56, 72, 91], 50))  # Output: -1
print(binary_search([1,2,3,4,5], 4)) # output: 3

print("\nFind Subarray with Given Sum:")
print(find_subarray_with_given_sum([1, 4, 20, 3, 10, 5], 33))  # Output: [2, 4]
print(find_subarray_with_given_sum([1, 4, 0, 0, 3, 10, 5], 7)) # output: [1,4]
print(find_subarray_with_given_sum([1, 4, 20, 3, 10, 5], 3)) # output: [3, 3]
print(find_subarray_with_given_sum([1, 4, 20, 3, 10, 5], 50)) # output: -1


import re
```

```python
from collections import Counter

def analyze_logs(log_file_path):
    """
    Parses a log file and extracts insights.

    Args:
        log_file_path: Path to the log file.

    Returns:
        A dictionary containing insights like frequent IPs, response codes, and URLs.
    """
    ip_pattern = re.compile(r'\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}')
    response_code_pattern = re.compile(r'\s(\d{3})\s')
    url_pattern = re.compile(r'GET\s(\S+)\sHTTP')

    ip_addresses = []
    response_codes = []
    urls = []

    try:
        with open(log_file_path, 'r') as file:
            for line in file:
                ip_match = ip_pattern.search(line)
                if ip_match:
                    ip_addresses.append(ip_match.group(0))

                response_match = response_code_pattern.search(line)
                if response_match:
                    response_codes.append(response_match.group(1))
                url_match = url_pattern.search(line)
                if url_match:
                    urls.append(url_match.group(1))

        frequent_ips = Counter(ip_addresses).most_common(5)
        frequent_response_codes = Counter(response_codes).most_common(5)
        frequent_urls = Counter(urls).most_common(5)

        return {
            'frequent_ips': frequent_ips,
            'frequent_response_codes': frequent_response_codes,
            'frequent_urls': frequent_urls,
        }

    except FileNotFoundError:
        return {'error': 'File not found.'}
    except Exception as e:
        return {'error': str(e)}

# Example Usage and Output (assuming a sample log file 'sample.log' exists):

# Create a sample log file for testing.
sample_log_content = """
192.168.1.1 - - [28/Sep/2023:10:00:00 +0000] "GET /index.html HTTP/1.1" 200 1234
192.168.1.2 - - [28/Sep/2023:10:01:00 +0000] "GET /about.html HTTP/1.1" 404 5678
192.168.1.1 - - [28/Sep/2023:10:02:00 +0000] "GET /index.html HTTP/1.1" 200 9012
10.0.0.1 - - [28/Sep/2023:10:03:00 +0000] "GET /contact.html HTTP/1.1" 200 3456
192.168.1.3 - - [28/Sep/2023:10:04:00 +0000] "GET /about.html HTTP/1.1" 404 7890
192.168.1.1 - - [28/Sep/2023:10:05:00 +0000] "GET /index.html HTTP/1.1" 200 1234
10.0.0.1 - - [28/Sep/2023:10:06:00 +0000] "GET /products.html HTTP/1.1" 200 5678
192.168.1.4 - - [28/Sep/2023:10:07:00 +0000] "GET /index.html HTTP/1.1" 200 9012
```

```
192.168.1.5 - - [28/Sep/2023:10:08:00 +0000] "GET /error.html HTTP/1.1" 500 3456
192.168.1.1 - - [28/Sep/2023:10:09:00 +0000] "GET /index.html HTTP/1.1" 200 7890
"""

with open('sample.log', 'w') as f:
    f.write(sample_log_content)

results = analyze_logs('sample.log')
print(results)
```