Algorithm Analysis

Abraham J. Reines

April 2, 2024

Task 6.3.1

Given the function $f(n) = 10^{12}n^3 + 10^6n^2 + n + 1$, we analyze its growth rates in terms of Big O, Small o, and Omega compared to $g(n) = n^4$ as n approaches infinity.

Big O Analysis

The Big O notation describes an upper bound of the algorithm, meaning it grows at most at the same rate as $g(n)$. The limit of $\frac{f(n)}{g(n)}$ as n approaches infinity has been found to be 0, which indicates that $f(n)$ grows no faster than $g(n)$ and is in fact slower. Therefore, $f(n)$ is $O(n^4)$.

Big O: True, because $f(n)$ is bounded above by $g(n)$ as n approaches infinity, and since the limit of $\frac{f(n)}{g(n)}$ is 0, $f(n)$ is indeed $O(n^4)$.

Small o Analysis

The Small o notation describes a stricter upper bound, meaning it grows strictly slower than $g(n)$. The analysis shows:

Small o: True, because the limit of $\frac{f(n)}{g(n)}$ as n approaches infinity is 0, which confirms $f(n)$ grows slower than $g(n)$. Thus, $f(n)$ is $o(n^4)$.

Omega Analysis

The Omega notation describes a lower bound, meaning the algorithm grows at least as fast as $g(n)$. The results of our analysis are:

Omega: False, because the limit of $\frac{f(n)}{g(n)}$ as n approaches infinity is 0, which signifies that $f(n)$ does not grow as fast as $g(n)$. Therefore, $f(n)$ is not $\omega(n^4)$.

6.3.2

Application Case:

- 250GB hard disk with 160GB of files.
- CPU with one core, 4GHz clock speed, and 64-bit registers.
- Capable of 4×10^9 8-character comparisons per second.

Search Time Analysis

1. **Single 1KB Signature:**

   The time required to search with a single 1KB signature is computed to be approximately 5 seconds. This duration represents the worst-case scenario, providing a concrete timeframe for completing a single search cycle. It is important to note that the Boyer-Moore algorithm will often perform faster than this worst-case estimate due to its pattern-skipping feature.

2. **Database of 1 Million 1KB Signatures:**

   Scaling up the search to account for a database of 1 million 1KB signatures, the total time estimated for the search is approximately 57.87 days, assuming a worst-case scenario where each signature search is independent and linear. The actual performance of the Boyer-Moore algorithm is expected to be better due to its sublinear average-case performance.

3. **Improving Efficiency:**

   To further improve the efficiency of malware/virus detection through string matching, we can utilize parallel processing by distributing the search workload across multiple CPU cores or even different machines. Additionally, implementing more advanced string matching algorithms such as Aho-Corasick, which is designed for searching multiple patterns simultaneously, can significantly reduce the time complexity. Optimizing the algorithm to take advantage of the preprocessing of the virus signatures and employing heuristic methods to skip unlikely sections of data can also enhance performance.

Scripting

```python
"""
Author: Abraham Reines
Created: Mon Apr  1 12:55:40 PDT 2024
Modified: Tue Apr  2 15:09:13 PDT 2024
Name of file: CheckingLimits.py
"""

from sympy import symbols, limit, oo

n = symbols('n')
f = 10**12 * n**3 + 10**6 * n**2 + n + 1

def AnalyzeAsymptoticBehavior(f, g, n):
    """
    Analyze the asymptotic behavior of f(n) against g(n) using Big O, small o, and Omega notations.

    Parameters:
    f (sympy expression): The function f(n) whose asymptotic behavior is to be analyzed.
    g (sympy expression): The benchmark function g(n) for comparison.
    n (sympy symbol): The variable with respect to which the limit is computed.
    """

    # Compute the limit for all conditions once
    asymptotic_limit = limit(f/g, n, oo)

    # Determine Big O condition
    big_O = asymptotic_limit.is_finite
    big_O_result = "True" if big_O else "False"
    print(f"Big O: {big_O_result}, because the limit of f(n)/g(n) as n approaches infinity is {asymptotic_limit}, which is finite.")

    # Determine small o condition
    small_o = asymptotic_limit == 0
    small_o_result = "True" if small_o else "False"
    print(f"Small o: {small_o_result}, because the limit of f(n)/g(n) as n approaches  infinity is {asymptotic_limit}, which confirms f(n) grows slower than g(n).")

    # Determine Omega condition by checking the inverse of the limit
    omega_limit = limit(g/f, n, oo)
    omega = omega_limit == 0
    omega_result = "True" if omega else "False"
    print(f"Omega: {omega_result}, because the limit of g(n)/f(n) as n approaches infinity is {omega_limit}, which confirms f(n) does not grow as fast as g(n)." )

```

```
# g(n) is defined as n**4 for comparison
AnalyzeAsymptoticBehavior(f, n**4, n)
```

Listing 1: Content from CheckingLimits.py

```
"""
Author: Abraham Reines
Created: Mon Apr  1 13:16:18 PDT 2024
Modified:
Name of file: BackofEnvelope.py
"""

import os

def calculate_search_time(file_size_bytes, signature_size_bytes,
    ↪ comparisons_per_second):
    """
    Calculate the time required to search a file with a given signature size using
    ↪ the Boyer-Moore algorithm.

    Parameters:
    file_size_bytes (int): Size of the file in bytes.
    signature_size_bytes (int): Size of the virus signature in bytes.
    comparisons_per_second (int): Number of 8-character comparisons the CPU can
    ↪ perform per second.

    Returns:
    float: Time required to search the file in seconds.
    """
    # Assuming the worst-case scenario, we have to scan the entire file.
    # Boyer-Moore algorithm is sublinear on average, but we are considering the worst
    ↪  case here.
    # The CPU can perform comparisons on 8 characters at a time, so we multiply
    ↪ comparisons by 8.
    total_comparisons = file_size_bytes - signature_size_bytes + 1

    # Time in seconds to perform the total comparisons
    # Each comparison is of 8 bytes, hence we multiply comparisons_per_second by 8.
    time_seconds = total_comparisons / (comparisons_per_second * 8)

    return time_seconds

def convert_seconds_to_readable_format(seconds):
    """
    Convert seconds to a more readable format (days or years).

    Parameters:
    seconds (float): Time in seconds.

    Returns:
    str: Time in days or years, with an appropriate description.
    """
    # Constants for conversion
    seconds_in_a_day = 86400
    days_in_a_year = 365.25

    if seconds < seconds_in_a_day:
        return f"{seconds} seconds"
    elif seconds < seconds_in_a_day * days_in_a_year:
        days = seconds / seconds_in_a_day
        return f"{days:.2f} days"
    else:
        years = seconds / (seconds_in_a_day * days_in_a_year)
        return f"{years:.2f} years"

# Constants
FILE_SIZE_BYTES = 160 * (10**9)  # 160GB
```

```
SIGNATURE_SIZE_BYTES = 1 * (2**10)  # 1KB
COMPARISONS_PER_SECOND = 4 * (10**9)  # This is the number of 8-character comparisons

# Calculation for a single 1KB signature
time_for_single_signature = calculate_search_time(FILE_SIZE_BYTES,
    ↪ SIGNATURE_SIZE_BYTES, COMPARISONS_PER_SECOND)
readable_time_for_single_signature = convert_seconds_to_readable_format(
    ↪ time_for_single_signature)

# Calculation for a database of 1 million signatures
# We will multiply the time for a single signature by 1 million.
# This is a crude estimation as the Boyer-Moore algorithm does not scale linearly.
time_for_million_signatures = time_for_single_signature * 1_000_000
readable_time_for_million_signatures = convert_seconds_to_readable_format(
    ↪ time_for_million_signatures)

# Output results
print(f"Time for searching with a single 1KB signature: {
    ↪ readable_time_for_single_signature}")
print(f"Time for searching with a database of 1 million 1KB signatures: {
    ↪ readable_time_for_million_signatures}")
```

Listing 2: Content from BackofEnvelope.py

References

Academic Integrity Pledge

"This work complies with the JMU honor code. I did not give or receive unauthorized help on this assignment."