CSCI 3333 Homework: Search
(with Solutions)

1 Recurrence Relations

Problem 1. Give a recurrence relation describing the running time of the following C++ function:

```cpp
int fib(int n)
{
    if (n < 2)
        return 1;
    return fib(n-1) + fib(n-2);
}
```

Solution 1. \( T(n) = T(n - 1) + T(n - 2) + c, T(0) = T(1) = d \) where \( c, d \in \mathbb{R}^+ \).

Problem 2. Give a recurrence relation describing the running time of the following C++ function:

```cpp
int factorial(int n)
{
    if (n < 2)
        return 1;
    return n * factorial(n-1);
}
```

Solution 2. \( T(n) = T(n - 1) + c, T(1) = d \) where \( c, d \in \mathbb{R}^+ \).

2 Ternary Search

Binary search is actually just one algorithm in a family of searching algorithms called binary, ternary, quaternary, etc. search. Each \( k \)-ary search algorithm partitions the input array into \( k \) equal-sized parts, and does \( k - 1 \) comparisons to decide which of these parts the search should recurse into.

Problem 3. Describe the ternary search algorithm (can use words, pseudocode, etc. – don’t worry about precise indices, types, etc.). For convenience, call the searched-for element \( x \) and the input array \( A \).

Solution 3. If \( A \) has length at most 10, search through \( A \) for \( x \) and return if found. Compare \( x \) to the elements of \( A \) one-third and two-thirds of the way through \( A \); if \( x \) is equal to either one return true. If \( x \) is less than the one-third element, recursively search in the first third of \( A \). If \( x \) is greater than the two-thirds element, recursively search in the last third of \( A \). Otherwise, recursively search in the middle third of \( A \).
Problem 4. Implement ternary search as a C++ function with header `bool search(int x, int* A, int n)`, where `x` is the item being searched for, `A` is the array to search in, and `n` is the length of the array.

Solution 4. `bool search(int x, int* A, int n)`
{
    if (n < 10)
    {
        for (int i = 0; i < n; ++i)
            if (A[i] == x)
                return true;
        return false;
    }

    int mid1 = A[n/3];
    int mid2 = A[2*n/3];
    if (x == mid1 || x == mid2)
        return true;
    if (x < mid1)
        return search(x, A, n/3);
    if (x > mid2)
        return search(x, A + 2*n/3, n - 2*n/3);
    return search(x, A + n/3, 2*n/3 - n/3);
}

Problem 5. Analyze the running time of ternary search (give the recurrence relation, find a closed form, prove the closed form correct).

Solution 5. A recurrence relation is \( T(n) = T(n/3) + c \), \( T(1) = d \) where \( c, d \in \mathbb{R}^+ \). First, finding a closed form via repeated substitution:

\[
T(n) = T(n/3) + c \\
= (T((n/3)/3) + c) + c \\
= T(n/3^2) + 2c \\
= (T((n/3^2)/3) + c) + 2c \\
= T(n/3^3) + 3c
\]

So we guess that \( T(n) = T(n/3^k) + kc \). Let \( k = \log_3(n) \). Then \( T(n) = T(1) + \log_3(n)c = d + c \log_3(n) \).

Now prove it by induction. Base case: \( T(1) = d = d + 0 = d + c \log_3(1) \).
Inductive step: Assume \( T(n) = d + c \log_3(n) \).
Then

\[ T(3n) = T((3n)/3) + c \]
\[ = T(n) + c \]
\[ = d + c \log_3(n) + c \]
\[ = d + c + c \log_3(n) \]
\[ = d + c(1 + \log_3(n)) \]
\[ = d + c(\log_3(3) + \log_3(n)) \]
\[ = d + c(\log_3(3n)) \]

Proved! So \( T(n) = d + c \log_3(n) = \Theta(\log(n)) \) is the running time of ternary search.