Problem 1

(This problem was postponed from problem set 1 and has been slightly modified.) Consider the following problem. You are given an array \( A \) of distinct positive integers and a target sum \( M \). Determine whether there is a subset \( S \) of \( A \) that adds up to \( M \) exactly.

For example, if \( A = [8, 11, 2, 6, 19, 3] \) and \( M = 17 \) then a solution is \( S = \{6, 8, 3\} \). If \( M = 7 \), there is no solution.

We can reason as follows: Suppose that \( S \) is a subset of \( A \) that adds up to \( M \). Then there are two possibilities:

- \( A[N] \) is an element of \( S \). Then \( S \setminus \{A[N]\} \) (that is, the set difference of \( S \) minus \( \{A[N]\} \)) is a subset of \( A[1 \ldots N - 1] \) that adds up to \( M - A[N] \). For instance, with the above value of \( A \) and with \( M = 17 \), the set \( \{6, 8\} \) adds up to \( 17 - 3 = 14 \).

- \( A[N] \) is not an element of \( S \). Then \( S \) is a subset of \( A[1 \ldots N - 1] \) and adds up to \( M \). For instance, with the same value of \( A \) and with \( M = 10 \), the solution \( S = \{8, 2\} \) is a subset of \( A[1 \ldots 5] \).

We can therefore formulate the following recursive algorithm:

```c
/* ExactSum(A, M, N) returns a subset of A[1 ... N] that adds up to exactly M if any such set exist; else it returns FALSE */
ExactSum(A, M, N) {
    if (M = 0) return emptyset;
    else if (M < 0) return FALSE;
    else if (M > 0 and N = 0) return FALSE;
    else S = ExactSum(A, M-A[N], N-1);
        if (S != FALSE)
            add A[N] to S;
            return S;
        else return ExactSum(A, M, N-1);
}
```

What is the asymptotic worst-case running time of this algorithm as a function of \( |A| \)? Describe how you would construct an example for some large value of \( |A| \) where the worst case is achieved.
Problem 2

Write down the recurrence equation for the running times of the following procedures, but do not solve them (and certainly do not try to figure out what it is that the functions compute).

function f(N) {
    if (N==1) return 1;
    else if (N==2) return 2;
    else return 2*f(N-1) + f(N/2);
}

function g(N) {
    if (N==1) return 1;
    else {
        sum = 1;
        for (i=0; i<N; i++)
            sum = 1 + sum*g(N-1);
        return sum;
    }
}

function h(N) {
    if (N==1) return 3;
    else {
        sum = 1;
        i = 0;
        while (i < h(N-1))
            sum = sum + i;
        i = i + 1;
        return sum;
    }
}

Assume that in executing h(N), the value of h(N-1) is recomputed at each iteration of the loop (which is probably the case for most languages and most compilers).

Problem 3

(Siegel, Ex. 2.28 (b)-(f)). Use the method of recurrence trees to solve the following recurrence equations to within a constant factor:

\[ B(1) = 1. \]
\[ B(n) = n + 3B(n/3), \quad n > 1. \] Assume \( n \) is a power of 3.
\[ C(1) = 1. \]
\[ C(n) = n^2 + 3C(n/2), \quad n > 1. \] Assume \( n \) is a power of 2.
\[ D(1) = 1. \]
\[ D(n) = n^2 + 9D(n/3), \quad n > 1. \] Assume \( n \) is a power of 3.
\[ E(1) = 1. \]
\[ E(n) = n^2 + rE(n/r), \quad n > 1. \] Assume \( n \) is a power of \( r \), where \( r \) is a positive integer greater than 1.
\[ F(1) = 1. \]
\[ F(n) = n^2 + 2F(n/2), \quad n > 1. \] Assume \( n \) is a power of 2.