Assignment 3: Algorithm Analysis

1. **(5 points)** Give a big-Oh notation, in terms of n, of the running time of the algorithm shown in the following code fragment:

```
/** Returns the sum of the prefix sums of given array. */
public static int example3(int[ ] arr) {
    int n = arr.length, total = 0;
    for (int j = 0; j < n; j++) // loop from 0 to n-1
        for (int k = 0; k <= j; k++) // loop from 0 to j
            total += arr[j];
    return total;
}
```

2. **(5 points)** Consider the problem of matrix multiplication, where the input includes two matrices, A (of dimension $n \times k$) and B (of dimension $k \times m$), the output is matrix C (of dimension $n \times m$). Recall that each element $C[i][j]$ in matrix C is the sum of the element-wise products of the i-th row of A and the j-th column of B. Consider the following pseudocode for matrix multiplication. Express the running time of the algorithm in big-Oh notation with the above dimension parameters.

```
Algorithm matProduct(A, B):
    for (i = 0:n-1) do
        for (j = 0:m-1) do
            C[i][j] = 0
            for (h = 0:k-1) do
                C[i][j] += A[i][h] * B[h][j]
```

3. (a) **(10 points)** The following pseudocode shows a recursive algorithm for computing 2^n for any integer $n \geq 0$. It is based on the recurrent relationship: $2^n = 2^{n-1} + 2^{n-1}$. Only

consider addition as a primitive operation. Use the substitution method to show the running time of this algorithm in big-Oh notation.

```
Algorithm recurPower(n):
    if (n == 0) then
        return 1
    else
        // addition as a primitive operation
        return recurPower(n-1) + recurPower(n-1)
```

(b) **(5 points)** The following pseudocode shows a non-recursive algorithm for computing 2^n. Show the running time of this algorithm in big-Oh notation. Here, you can consider multiplication as a constant-time primitive operation.

```
Algorithm iterPower(n):
    if (n == 0) then
        return 1
    else
        power = 1
        for (i = 0:n-1) do
            // consider multiplication as a primitive
               operation
            power = power * 2
        return power
```

4. **(10 points)** Suppose there is a recursive algorithm and its running time $T(n)$ satisfies a recurrence relationship $T(n) = 2T(n/2) + n$ when the input size $n > 1$, and $T(1) = 1$. Use the substitution method to show the running time $T(n)$ in the big-Oh notation.