cs141 Workshop: Asymptotic Analysis

Perform an analysis of the following algorithms:

Algorithm arrayMax(A, n): Analysis: $\Theta(n)$

Input: An array A storing $n \geq 1$ integers.
Output: The maximum element in A.
currentMax $\leftarrow A[0]$
for $i \leftarrow 1$ to $n-1$ do
    if currentMax $< A[i]$ then
        currentMax $\leftarrow A[i]$
return currentMax

Algorithm arrayFind(x, A) Analysis: $\mathrm{O}(n)$

Input: An element x and an n-element array, A.
Output: The index i such that $x = A[i]$ or -1 if no element of A is equal to x.
$i \leftarrow 0$
while $i < n$ do
    if $x = A[i]$ then
        return i
    else
        $i \leftarrow i + 1$
return -1

Algorithm prefixAverages1(X) Analysis: $\Theta(n^2)$

Input: an n-element array of X numbers.
Output: an n-element array A of numbers such that $A[i]$ is the average of of elements $X[0], \ldots, X[i]$.
Let A be an array of n numbers.
for $i \leftarrow 1$ to $n-1$ do
    $a \leftarrow 0$
    for $j \leftarrow 0$ to i do
        $a \leftarrow a + X[j]$
    $A[i] \leftarrow a/(i+1)$
return array A

Algorithm prefixAverage2(X) Analysis: $\Theta(n)$

Input: an n-element array of X numbers.
Output: an n-element array A of numbers such that $A[i]$ is the average of of elements $X[0], \ldots, X[i]$.
Let A be an array of n numbers.
$s \leftarrow 0$

```
    for i ← 0 to n − 1 do
        s ← s + X[i]
        A[i] ← s /(i + 1)
    return array A
```

```
procedure matrixmultiply( n : integer);                    Analysis: Θ(n^3)
    var i, j, k : integer;
    begin
        for i ← 1 to n do
            for j ← 1 to n do begin
                C[i, j] ← 0 ;
                for k ← 1 to n do
                    C[i, j] ← C[i, j] + A[i, k] * B[k, j] ;
            end;
    end;
```

```
procedure mystery( n : integer);                           Analysis: Θ(n^2)
    var i, j, k : integer;
    begin
        k ← 0 ;
        for i ← 1 to n do
            if i mod 2 = 0 then
                for j ← 1 to i / 2 do
                    k ← k + j ;
    end;
```

```
procedure recursive( n : integer);                         Analysis: Θ(2^n)
    begin
        if n <= 1 then
            return(1);
        else
            return(recursive( n − 1 ) + recursive( n − 1 ));
    end;
```