Asymptotic Analysis (Part 2)

C.S. Marron
cmarron@umbc.edu

UMBC

CMSC 341 — Data Structures

Example: Prefix Sums

Problem Statement

- Given a numeric array A of length n.
- Compute the sums

$$p_j = \sum_{i=0}^{j} A[i]$$

  for $j = 0, 1, n - 1$.
- Ouptut the array $B = (p_0, p_1, \ldots, p_{n-1})$.
- We will assume integer values.

Prefix Sum: First Solution

```
void prefixSum(int *A, int *B, int n) {
  for ( int j = 0; j <= n-1; j++ ) {
    B[j] = 0 ;
    for ( int i = 0; i <= j; i++ ) {
      B[j] += A[i] ;
    }
  }
}
```

- Determine an asymptotic upper bound on the running time of `prefixSum()`.
- Does `prefixSum()` have a Θ bound?
- Answers: $O(n^2)$. Yes, it is $\Omega(n^2)$ and thus $\Theta(n^2)$.

Prefix Sum: Second Solution

```
void prefixSum2(int *A, int *B, int n) {
  B[0] = A[0] ;
  for ( int i = 1; i < n; i++ ) {
    B[i] = B[i-1] + A[i] ;
  }
}
```

- Determine an asymptotic upper bound on the running time.
- Does `prefixSum()` have a Θ bound?
- Show that this method returns the correct result.
- Answers: It is $O(n)$. Since it is also $\Omega(n)$, it is $\Theta(n)$.

Example: Exponentiation

Problem Statement

- Given a real number x and a non-negative integer n.
- Compute and return the power x^n.
- Ignore overflow. Pretend our numbers are arbitrary precision.

Naive Solution

```
float exp(float x, int n) {
  float ans = 1.0 ;
  for ( int i = 0; i < n; i++ ) {
    ans *= x;
  }
  return ans ;
}
```

Exponentiation: Recursive Solution

```
float exp2( float x, int n ) {
  if ( n == 0 ) {
    return 1.0 ;
  } else if ( n % 2 == 1) {
    float y = exp2( x, (n - 1) / 2 ) ;
    return y * y * x ;
  } else {
    float y = exp2( x, n / 2 ) ;
    return y * y ;
  }
}
```

- Determine an asymptotic upper bound on the running time.
- Does `exp2()` have a Θ bound?
- Show that this method returns the correct result.
- Answers: It is $O(\log n)$. Since it is also $\Omega(\log n)$, it is $\Theta(\log n)$.

Reading Assignment

Read the following sections in the textbook

1. Section 4.2.5: Using Big-Oh Notation
2. Section 4.2.6: A Recursive Algorithm for Computing Powers