CS203 Programming with Data Structures
Introduction to Algorithm Analysis

Chengyu Sun
California State University, Los Angeles

Algorithm and Algorithm Analysis

- Algorithm - a well defined set of instructions to complete a task
- Algorithm analysis - estimate the time/space required by an algorithm
  - Optimization
  - Comparison

Algorithm != Code

- The same algorithm can be implemented by different people, in different languages, under different conditions
- To analyze algorithm, we need to abstract away the implementation details

The Model

- A computer with infinite amount of memory
- Simple instructions only (addition, multiplication, assignment etc.)
- Sequential execution
- Each instruction takes one *unit* of time

Running Time

- Given size of the input N
  - $T_{best}(N)$ – best-case running time
  - $T_{worst}(N)$ – worst-case running time
  - $T_{avg}(N)$ – average running time

Example 1: Max

```
int max( int a[] )
{
    int max=a[0];
    for( int i=1 ; i < a.length ; ++i )
        if( max < a[i] ) max = a[i];
    return max;
}
```

- N??, $T_{best}(N)$??, $T_{worst}(N)$??, $T_{avg}(N)$??

Example 2: Search

```
int search( int x, int a[] )
{
    for( int i=0 ; i < a.length ; ++i )
        if( a[i] == x ) return I;
    return -1;
}
```

- N??, $T_{best}(N)$??, $T_{worst}(N)$??, $T_{avg}(N)$??

The Big-O Notation

- T(N) = O(f(N)) if there are positive constants c and n_0 such that $T(N) \leq cf(N)$ when $N \geq n_0$
- O(f(N)) – *Order of f(N)*

Time Complexity

- In the Big-O notation

Running Time T(N)	Time Complexity
3	O(1)
2057 + N	O(N)
$100N^2$	$O(N^2)$
$1321312+ 23244255N + N^2$	$O(N^2)$

Two Simple Rules

- Constants do not matter
- Higher-order terms dominate lower-order terms

Growth Rate of Some Functions

N	logN	NlogN	N^2	2^N
1	0	0	1	2
2	1	2	2	4
4	2	8	16	16
8	3	24	64	256
16	4	64	256	65536
32	5	160	1024	4294967296

Growth Rate of N^2 and $N^2+4N+20$

N	N^2	$N^2+4N+20$
10	300	360
50	2500	2720
100	10000	10420
1000	1000000	1004020
10000	100000000	100040020

Understand the Definition

- How do we show 2057+N = O(N)??

  T(N) = 2057+N and f(N) = N, and based on the definition, we need to find c and n_0 such that

  $2057+N \leq cN$ for $N>n_0$

  We can pick c=2 and n_0=2057. Because

  $2057 < N$ for $N>2057$ and

  $N \leq N$

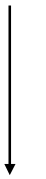
  We have

  $N+2057 \leq 2N$ for $N>2057$

Typical Complexities

Time Complexity	Name
O(1)	Constant
O(logN)	Logarithmic
$O(\log^2 N)$	Log-squared
O(N)	Linear
O(NlogN)	
$O(N^2)$	Quadratic
$O(N^3)$	Cubic
2^N	Exponential

efficient



expensive

Example 3: Binary Search

```
int binarySearch( int x, int a[] )
{
    int index = -1, left = 0, right = a.length-1, mid;
    while( left <= right ) {
        mid = (left+right)/2;
        if( a[mid] > value ) right = mid-1;
        else if( a[mid] < value ) left = mid+1;
        else { index = mid; break; }
    }
    return index;
}
```

- Time complexity: best-case??, worst-case??, average case??

Beyond Basics

- o, Ω, and Θ
- Proofs
- Complex complexities, e.g.

  $T(N) = T(N-1) + T(N-2) + 2$