ساختمان داده‌ها و الگوریتم‌ها

فصل دوم

S.Najjar.G@Gmail.com

صمد نجار

فهرست مطالب

- مقدمه‌ای بر الگوریتم‌ها و مفاهیم پایه
- **معرفی پیچیدگی زمانی و حافظه‌ای و روشهای تحلیل مسائل**
- معرفی ساختمان داده‌های مقدماتی و الگوریتم‌های وابسته به آنها
  - آرایه
  - صف
  - پشته
  - لیست پیوندی
- تئوری درخت و گراف و الگوریتم‌های مرتبط
- الگوریتم‌های مرتب‌سازی و تحلیل پیچیدگی مربوط به آنها
- مباحث تکمیلی در ساختمان داده‌ها

سوال

- اگر دو الگوریتم داشته باشیم که یک هدف را انجام دهند، چگونه باید فهمید که کدام بهتر عمل می‌کنند؟

- معیار ارزیابی چه پارامترهایی می‌تواند باشد؟

Space Complexity
$S(P)=C+S_P(I)$

- Fixed Space Requirements (C)
  Independent of the characteristics of the inputs and outputs
  - instruction space
  - space for simple variables, fixed-size structured variable, constants
- Variable Space Requirements ($S_P(I)$)
  depend on the instance characteristic I
  - number, size, values of inputs and outputs associated with I
  - recursive stack space, formal parameters, local variables, return address

Example

Program 1.9: Simple arithmetic function

```
float abc(float a, float b, float c)
{
   return a + b + b * c + (a + b - c) / (a + b) + 4.00;
 }
```

$S_{abc}(I) = 0$

Program 1.10: Iterative function for summing a list of numbers

```
float sum(float list[ ], int n)
{
  float tempsum = 0;
  int i;
  for (i = 0; i<n; i++)
     tempsum += list [i];
  return tempsum;
}
```

$S_{sum}(I) = 0$

Example (Cont.)

*Program 1.11: Recursive function for summing a list of numbers (p.20)

```
float rsum(float list[ ], int n)
{
  if (n > 0)
        return rsum(list, n-1) + list[n-1];
  return 0;
 }
```

$S_{rsum}(I) = S_{rsum}(n)= 6n$

Space needed for one recursive call of Program 1.11

Type	Name	Number of bytes
parameter: float *	list	2
parameter: integer	n	2
return address:(used internally)		2(unless a far address)
TOTAL per recursive call		6

Time Complexity

$$T(P)=C+T_P(I)$$

- Compile time (C)
  independent of instance characteristics
- run (execution) time T_P
- Definition
  A *program step* is a syntactically or semantically meaningful program segment whose execution time is independent of the instance characteristics.
- Example
  - abc = a + b + b * c + (a + b - c) / (a + b) + 4.0
  - abc = a + b + c

Methods to compute the step count

- Introduce variable count into programs
- Tabular method
  - Determine the total number of steps contributed by each statement
    step per execution × frequency
  - add up the contribution of all statements

Variable Count Method (Example)

***Program 1.12:** Program 1.10 with count statements*

2n + 3 Steps

```
float sum(float list[ ], int n)
{
    float tempsum = 0; count++; /* for assignment */
    int i;
    for (i = 0; i < n; i++) {
        count++;            /*for the for loop */
        tempsum += list[i]; count++;  /* for assignment */
    }
    count++;        /* last execution of for */
    return tempsum;
    count++;        /* for return */
}
```

Variable Count Method (Example Cont.)

***Program 1.13:** Simplified version of Program 1.12

```
float sum(float list[ ], int n)
{
    float tempsum = 0;
    int i;
    for (i = 0; i < n; i++)
         count += 2;
    count += 3;
    return 0;
}
```

2n + 3 Steps

Variable Count Method (Example Cont.)

***Program 1.14:** Program 1.11 with count statements added*

```
float rsum(float list[ ], int n)
{
        count++;     /*for if conditional */
        if (n > 0) {
                count++;  /* for return and rsum invocation */
                return rsum(list, n-1) + list[n-1];
        }
        count++;
        return 0;
}
```

2n + 2 Steps

Variable Count Method (Example Cont.)

***Program 1.15:** Matrix addition*

```
void add( int a[ ] [MAX_SIZE], int b[ ] [MAX_SIZE],
                              int c [ ] [MAX_SIZE], int rows, int cols)
{
    int i, j;
    for (i = 0; i < rows; i++)
       for (j= 0; j < cols; j++)
          c[i][j] = a[i][j] +b[i][j];
 }
```

Variable Count Method (Example Cont.)

***Program 1.16:** Matrix addition with count statements (p.25)

```
void add(int a[ ][MAX_SIZE], int b[ ][MAX_SIZE],
                           int c[ ][MAX_SIZE], int row, int cols )
{
   int i, j;
   for (i = 0; i < rows; i++){
        count++; /* for i for loop */
        for (j = 0; j < cols; j++) {
           count++; /* for j for loop */
           c[i][j] = a[i][j] + b[i][j];
           count++; /* for assignment statement */
        }
        count++;    /* last time of j for loop */
   }
  count++;         /* last time of i for loop */
}
```

2rows * cols + 2 rows + 1

Variable Count Method (Example Cont.)

***Program 1.17:** Simplification of Program 1.16*

```
void add(int a[ ][MAX_SIZE], int b [ ][MAX_SIZE],
                           int c[ ][MAX_SIZE], int rows, int cols)
{
    int i, j;
    for( i = 0; i < rows; i++)
   {
       for (j = 0; j < cols; j++)
           count += 2;
       count += 2;
    }
    count++;
}
```

2rows * cols + 2 rows + 1

Suggestion: Interchange the loops when rows >> cols

Tabular Method (Example)

Iterative function to sum a list of numbers

steps/execution

Statement	s/e	Frequency	Total steps
float sum(float list[ ], int n)	0	0	0
{	0	0	0
float tempsum = 0;	1	1	1
int i;	0	0	0
for(i=0; i <n; i++)	1	n+1	n+1
tempsum += list[i];	1	n	n
return tempsum;	1	1	1
}	0	0	0
Total			2n+3

Tabular Method (Example Cont.)

Step count table for recursive summing function

Statement	s/e	Frequency	Total steps
float rsum(float list[ ], int n)	0	0	0
{	0	0	0
if (n)	1	n+1	n+1
return rsum(list, n-1)+list[n-1];	1	n	n
return list[0];	1	1	1
}	0	0	0
Total			2n+2

Tabular Method (Example Cont.)

Step count table for matrix addition

Statement	s/e	Frequency	Total steps
Void add (int a[ ][MAX_SIZE] • • •)	0	0	0
{	0	0	0
int i, j;	0	0	0
for (i = 0; i < row; i++)	1	rows+1	rows+1
for (j=0; j< cols; j++)	1	rows • (cols+1)	rows • cols+rows
c[i][j] = a[i][j] + b[i][j];	1	rows • cols	rows • cols
}	0	0	0
Total			2rows • cols+2rows+1

Home Work 1

Trade off between time and space complexity

Asymptotic Notation

- Determining step counts help us to compare the time complexities of two programs and to predict the growth in run time as the instance characteristics change.
- But determining exact step counts could be very difficult. Since the notion of a step count is itself inexact, it may be worth the effort to compute the exact step counts.

- <u>How does the algorithm behave as the problem size gets very large?</u>

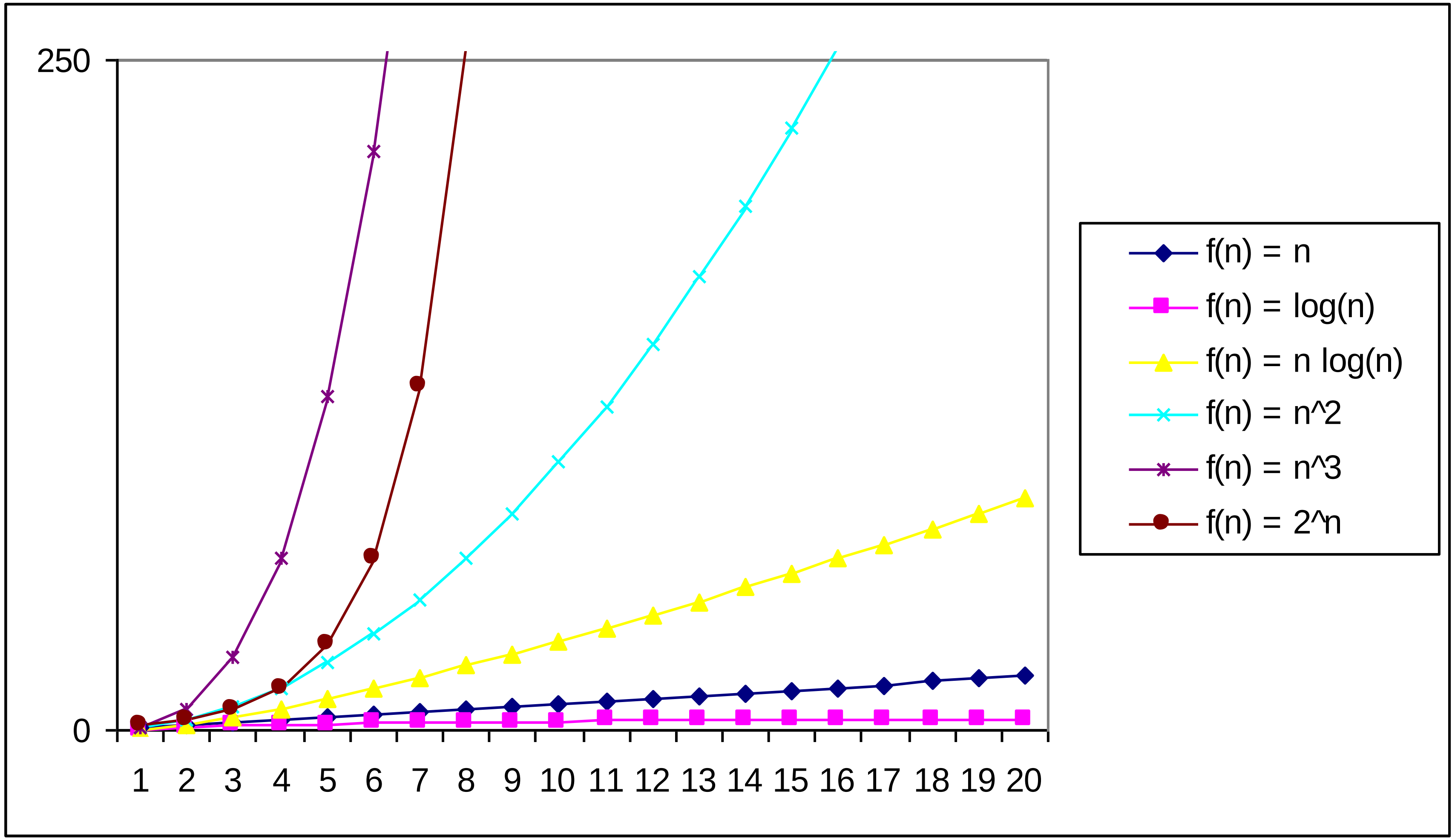
Asymptotic Notation (O)

- Definition(Big oh)
- $f(n) = O(g(n))$ iff there exist positive constants c and n_0 such that $f(n) \leq cg(n)$ for all n, $n \geq n_0$.
- Examples
  - $3n+2=O(n)$/* $3n+2\leq 4n$ for $n\geq 2$ */
  - $3n+3=O(n)$/* $3n+3\leq 4n$ for $n\geq 3$ */
  - $100n+6=O(n)$ /* $100n+6\leq 101n$ for $n\geq 10$ */
  - $10n^2+4n+2=O(n^2)$ /* $10n^2+4n+2\leq 11n^2$ for $n\geq 5$ */
  - $6*2^n+n^2=O(2^n)$ /* $6*2^n+n^2 \leq 7*2^n$ for $n\geq 4$ */

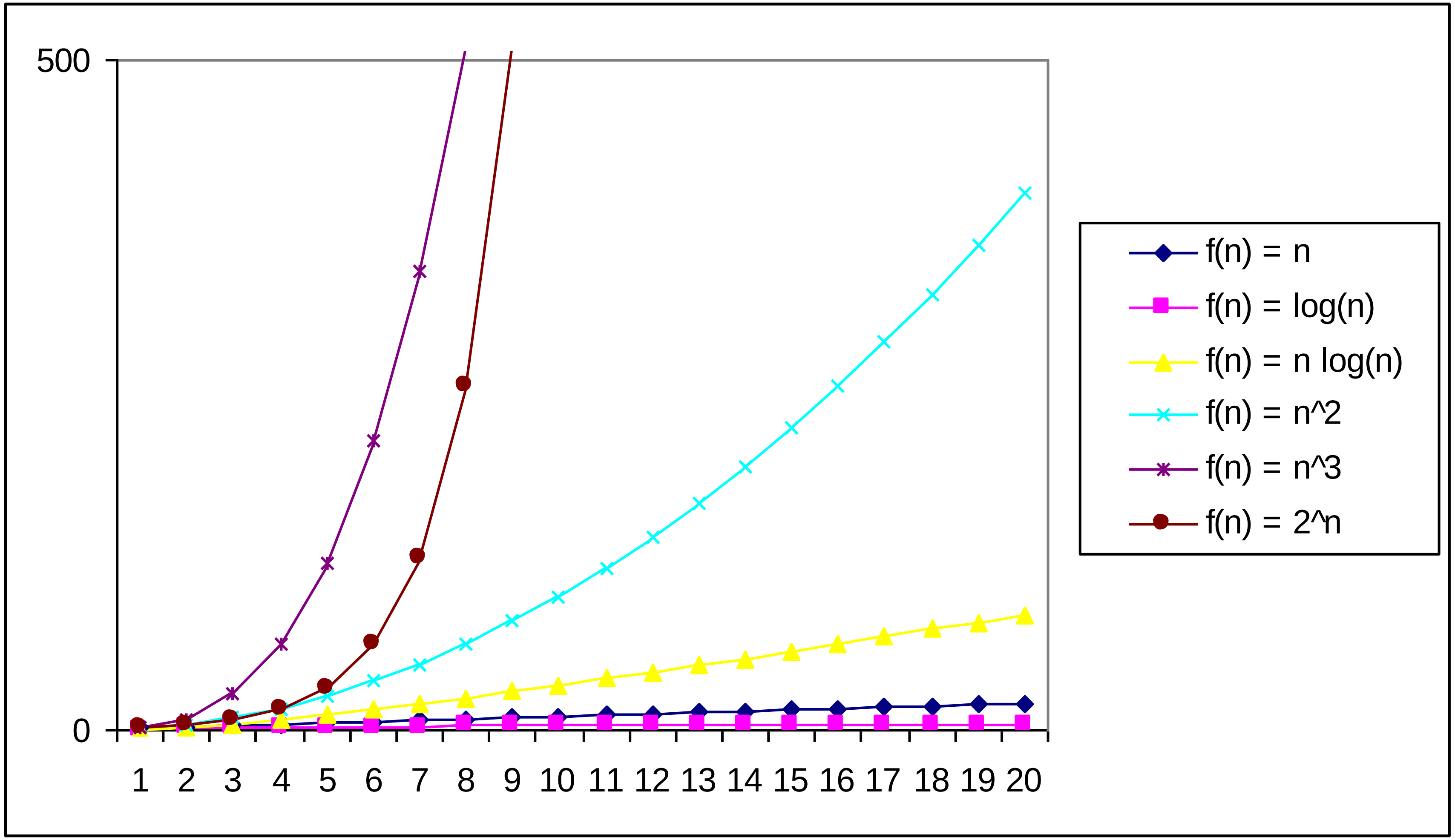
Asymptotic Notation Cont. (O)

- O(1): constant
- O(n): linear
- O(n^2): quadratic
- O(n^3): cubic
- O(2^n): exponential
- O(logn) logarithmic
- O(nlogn) log linear

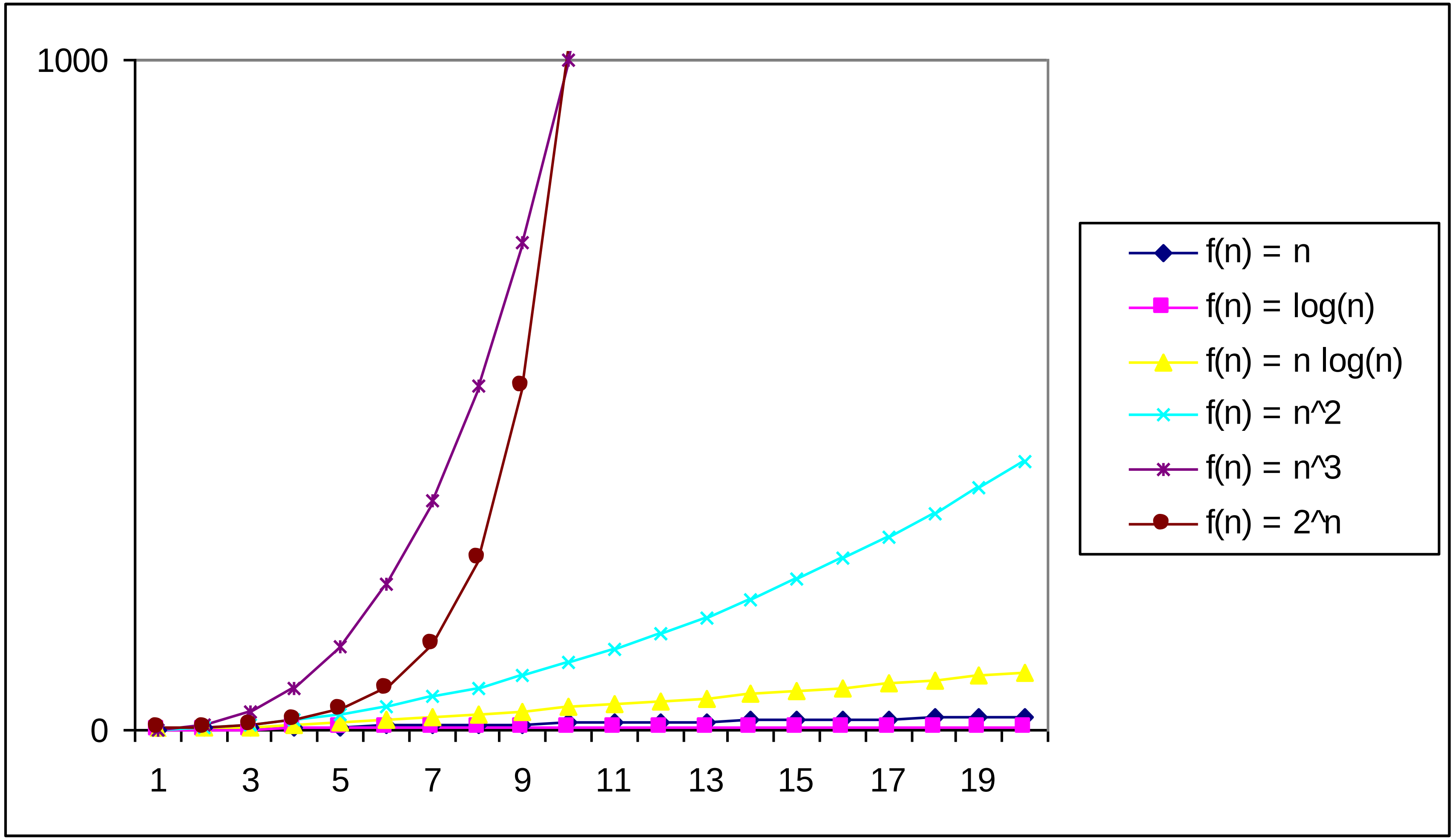
Practical Complexity



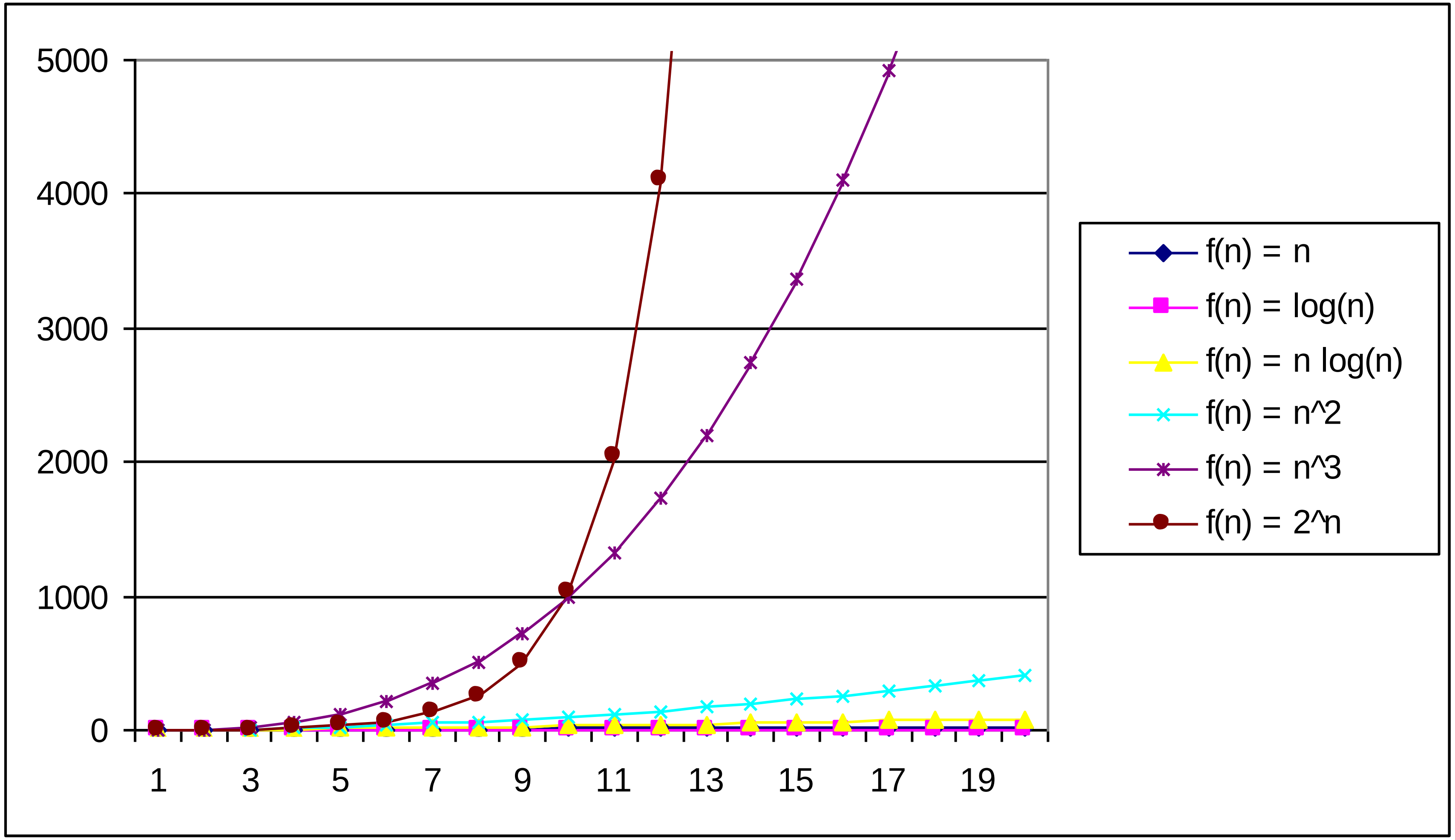
Practical Complexity Cont.



Practical Complexity Cont.



Practical Complexity Cont.



Asymptotic Notation (Cont.)

- Definition: [Omega] $f(n) = \Omega(g(n))$ iff there exist positive constants c and n_0 such that $f(n) \geq cg(n)$ for all n, $n \geq n_0$.
- Example:
  - $3n + 2 = \Omega(n)$
  - $100n + 6 = \Omega(n)$
  - $10n^2 + 4n + 2 = \Omega(n^2)$

Asymptotic Notation (Cont.)

Definition: $f(n) = \Theta(g(n))$ iff there exist positive constants c_1, c_2, and n_0 such that $c_1 g(n) \le f(n) \le c_2 g(n)$ for all n, $n \ge n_0$.

Asymptotic Notation (Cont.)

Theorem 1.2: If $f(n) = a_m n^m + \ldots + a_1 n + a_0$, then $f(n) = O(n^m)$.

Proof:

$$f(n) \le \sum_{i=0}^{m} |a_i| n^i$$

$$\le n^m \sum_{0}^{m} |a_i| n^{i-m}$$

for $n \ge 1$

$$\le n^m \sum_{0}^{m} |a_i|$$

So, $f(n) = O(n^m)$

Asymptotic Notation (Cont.)

Theorem 1.3: If $f(n) = a_m n^m + \ldots + a_1 n + a_0$ and $a_m > 0$, then $f(n) = \Omega(n^m)$.

Theorem 1.4: If $f(n) = a_m n^m + \ldots + a_1 n + a_0$ and $a_m > 0$, then $f(n) = \Theta(n^m)$.

	Time for $f(n)$ instructions on a 10^9 instr/sec computer						
n	$f(n)=n$	$f(n)=\log_2 n$	$f(n)=n^2$	$f(n)=n^3$	$f(n)=n^4$	$f(n)=n^{10}$	$f(n)=2^n$
10	.01μs	.03μs	.1μs	1μs	10μs	10sec	1μs
20	.02μs	.09μs	.4μs	8μs	160μs	2.84hr	1ms
30	.03μs	.15μs	.9μs	27μs	810μs	6.83d	1sec
40	.04μs	.21μs	1.6μs	64μs	2.56ms	121.36d	18.3min
50	.05μs	.28μs	2.5μs	125μs	6.25ms	3.1yr	13d
100	.10μs	.66μs	10μs	1ms	100ms	3171yr	$4*10^{13}$ yr
1,000	1.00μs	9.96μs	1ms	1sec	16.67min	$3.17*10^{13}$ yr	$32*10^{283}$ yr
10,000	10.00μs	130.03μs	100ms	16.67min	115.7d	$3.17*10^{23}$ yr	
100,000	100.00μs	1.66ms	10sec	11.57d	3171yr	$3.17*10^{33}$ yr	
1,000,000	1.00ms	19.92ms	16.67min	31.71yr	$3.17*10^{7}$ yr	$3.17*10^{43}$ yr	

μs = microsecond = 10^{-6} seconds
ms = millisecond = 10^{-3} seconds
sec = seconds
min = minutes
hr = hours
d = days
yr = years

Home Work 2