Algorithm Efficiency

- **There are often many approaches (algorithms) to solve a problem. How do we choose between them?**
- **At the heart of computer program design are two (sometimes conflicting) goals:**
  - To design an algorithm that is easy to understand, code and debug.
  - To design an algorithm that makes efficient use of the computer's resources.

Algorithm Efficiency

- Goal (1) is the concern of Software Engineering.
- Goal (2) is the concern of data structures and algorithm analysis.
- When goal (2) is important, how do we measure an algorithm's cost?

How to Measure Efficiency?

- Empirical comparison (run programs).
- Asymptotic Algorithm Analysis.

- Critical resources:
- Factors affecting running time:

How to Measure Efficiency?

- For most algorithms, running time depends on "size" of the input.
- Running time is expressed as T(n) for some function T on input size n.

Examples of Growth Rate

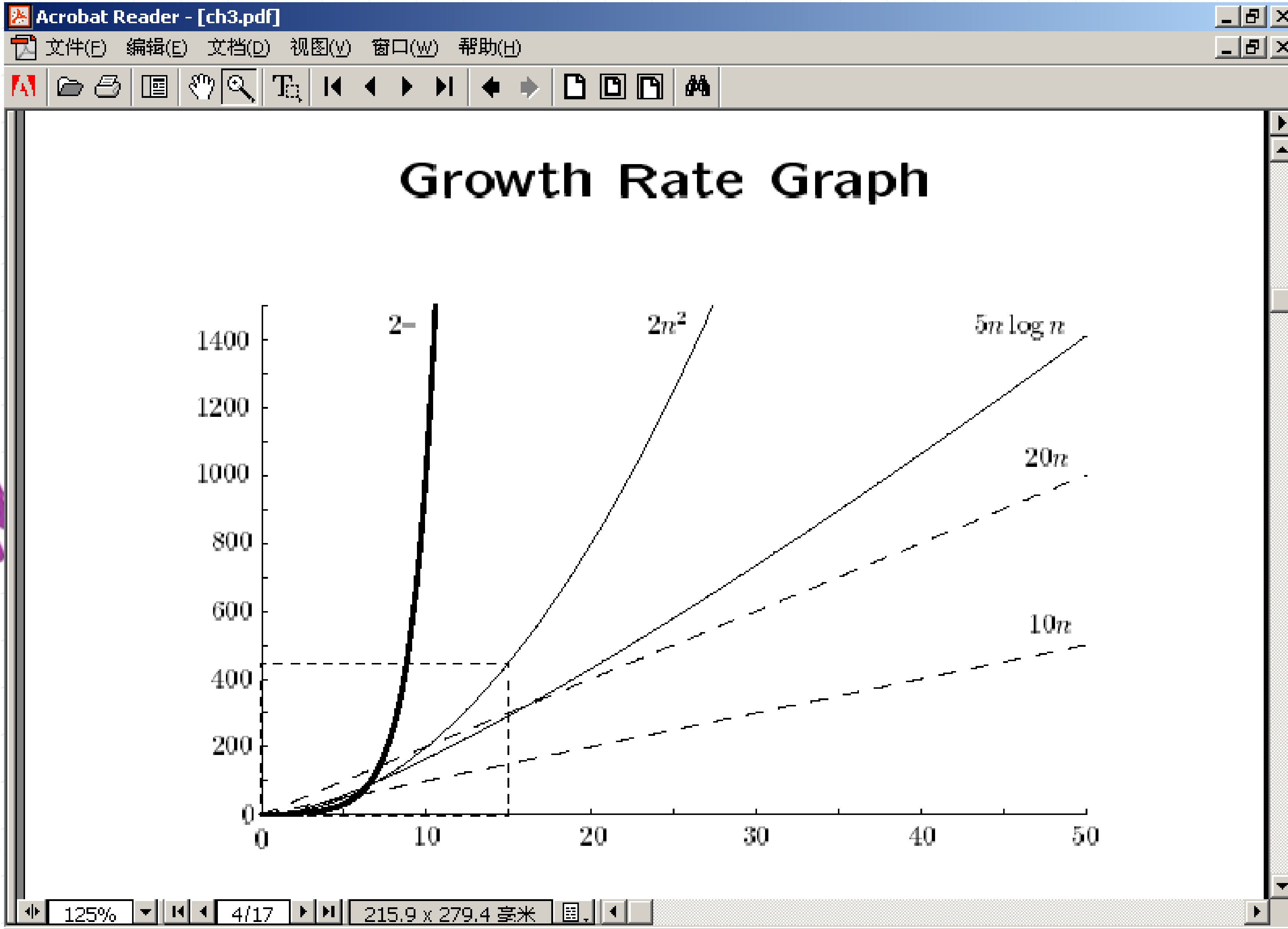
- Example 1:

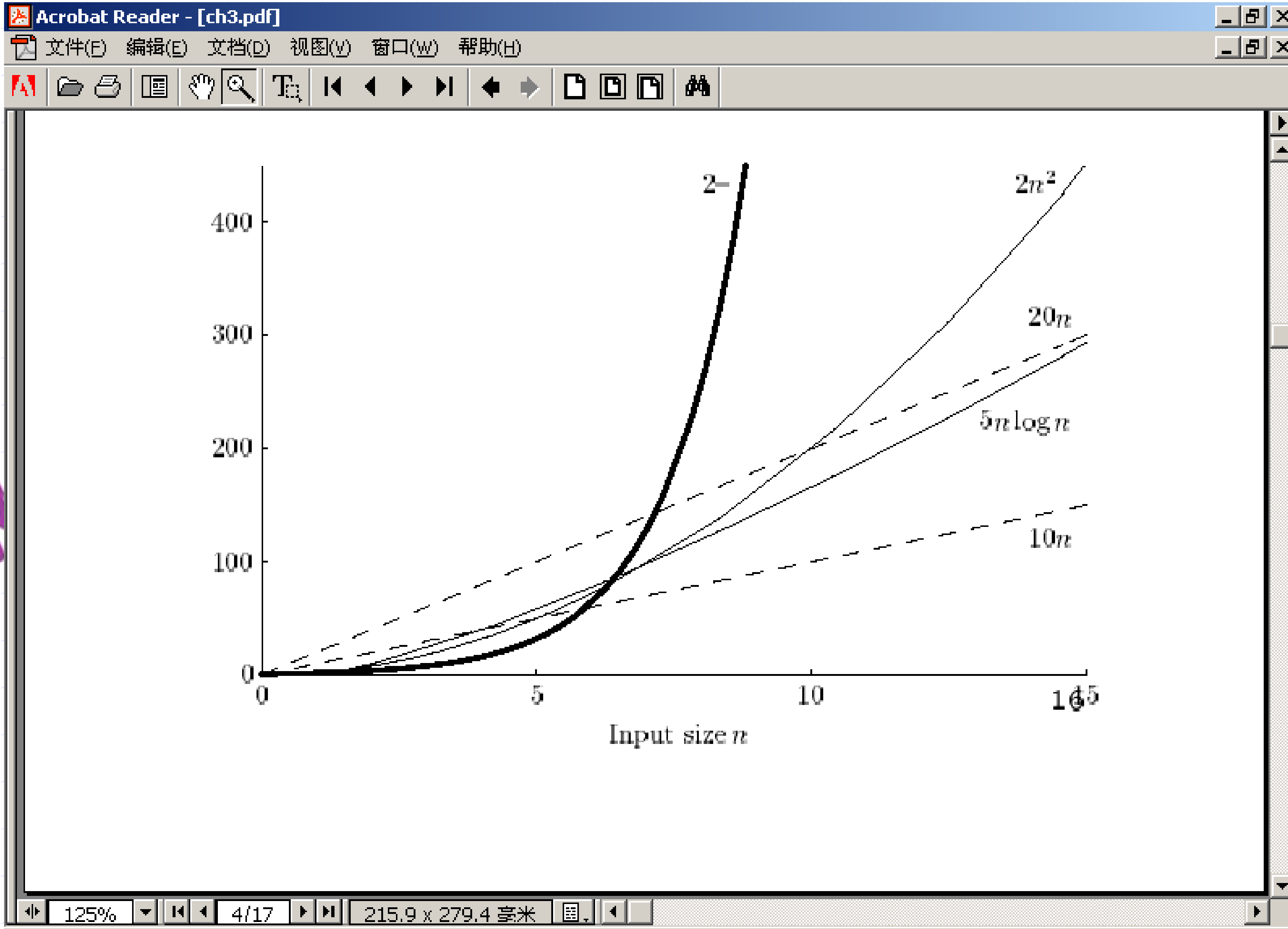
```
int largest(int* array, int n)    // Find largest value
{  int currlarge = array[0];      // Store largest seen
   for (int i=1; i<n; i++)        // For each element
        if (array[i] > currlarge) // If largest
             currlarge = array[i];
                                  // Remember it
   return currlarge;              // Return largest
}
```

Examples of Growth Rate

- Example 2:

```
sum = 0;
for (i=1; i<=n; i++)
    for (j=1; j<=n; j++)
        sum++;
```


Acrobat Reader - [ch3.pdf]
文件(F) 编辑(E) 文档(D) 视图(V) 窗口(W) 帮助(H)
Growth Rate Graph
2ⁿ
2n²
5n log n
20n
10n
1400
1200
1000
800
600
400
200
0
0
10
20
30
40
50
125%
4/17
215.9 x 279.4 毫米


Acrobat Reader - [ch3.pdf]
文件(F) 编辑(E) 文档(D) 视图(V) 窗口(W) 帮助(H)
2^n
$2n^2$
$20n$
$5n \log n$
$10n$
400
300
200
100
0
0
5
10
15
Input size n
125%
4/17
215.9 x 279.4 毫米

Best, Worst and Average Cases

- **Not all inputs of a given size take the same time.**
- **Sequential search for K in an array of n integers:**
  - Begin at first element in array and look at each element in turn until K is found.

Best, Worst and Average Cases

- Best Case:
- Worst Case:
- Average Case:
- While average time seems to be the fairest measure, it may be difficult to determine.
- When is worst case time important?

Faster Computer or Algorithm?

- What happens when we buy a computer 10 times faster?

Acrobat Reader - [ch3.pdf]

T(n)	n	n'	Change	n'/n
$10n$	$1,000$	$10,000$	$n' = 10n$	10
$20n$	500	$5,000$	$n' = 10n$	10
$5n \log n$	250	$1,842$	$\sqrt{10}n < n' < 10n$	7.37
$2n^2$	70	223	$n' = \sqrt{10}n$	3.16
2^n	13	16	$n' = n + 3$	$--$

n: Size of input that can be processed in one hour (10,000 steps).

n': Size of input that can be processed in one hour on the new machine (100,000 steps).

Asymptotic Analysis: Big-oh

- **definition:** (Big-Oh) $T(N)$ is $O(F(N))$ if there are positive constants c and N_0 such that $T(N) \leqslant cF(N)$ when $N \geqslant N_0$.
- **definition:** (Big-Omega) $T(N)$ is $\Omega(F(N))$ if there are positive constants c and N_0 such that $T(N) \geqslant cF(N)$ when $N \geqslant N_0$.
- **definition:** (Big-Theta) $T(N)$ is $\Theta(F(N))$ if and only if $T(N)$ is $O(F(N))$ and $T(N)$ is $\Omega(F(N))$.
- **definition:** (Little-Oh) $T(N)$ is $o(F(N))$ if and only if $T(N)$ is $O(F(N))$ and $T(N)$ is not $\Omega(F(N))$.

Meanings

- **Meanings of the various growth functions**

Mathematical Expression	Relative Rates of Growth
$T(N) = O(F(N))$	Growth of $T(N)$ is $\leq$ growth of $F(N)$.
$T(N) = \Omega(F(N))$	Growth of $T(N)$ is $\geq$ growth of $F(N)$.
$T(N) = \Theta(F(N))$	Growth of $T(N)$ is $=$ growth of $F(N)$.
$T(N) = o(F(N))$	Growth of $T(N)$ is $<$ growth of $F(N)$.

Asymptotic Analysis: Big-oh

- Usage: The algorithm is in $O(n^2)$ in [best, average, worst] case.
- Meaning: For all data sets big enough (i.e., $n > n_0$), the algorithm always executes in less than $cf(n)$ steps [in best, average or worst case].

Asymptotic Analysis: Big-oh

- Upper Bound.
- Example: if $T(n) = 3n^2$ then $T(n)$ is in $O(n^2)$.
- Wish tightest upper bound:
- While $T(n) = 3n^2$ is in $O(n^3)$, we prefer $O(n^2)$.

Simplifying Rules:

- If $f(n)$ is in $O(g(n))$ and $g(n)$ is in $O(h(n))$,then $f(n)$ is in $O(h(n))$.
- If $f(n)$ is in $O(kg(n))$ for any constant $k > 0$, then $f(n)$ is in $O(g(n))$.
- If $f_1(n)$ is in $O(g_1(n))$ and $f_2(n)$ is in $O(g_2(n))$, then $(f_1 + f_2)(n)$ is in $O(\max(g_1(n), g_2(n)))$.

Simplifying Rules:

- If $f_1(n)$ is in $O(g_1(n))$ and $f_2(n)$ is in $O(g_2(n))$ then $f_1(n)f_2(n)$ is in $O(g_1(n)g_2(n))$.

Running Time of a Program

- **Example 1:**

  ```
  a = b;
  ```

  This assignment takes constant time, so it is(1).

- **Example 2:**

  ```
  sum = 0;
  for (i=1; i<=n; i++)
      sum += n;
  ```

Running Time of a Program

- **Example 3:**

```
sum = 0;
for (j=1; j<=n; j++) // First for loop
    for (i=1; i<=j; i++) // is a double loop
        sum++;
for (k=0; k<n; k++) // Second for loop
    A[k] = k;
```

More Examples

- Example 4.

```
sum1 = 0;
for (i=1; i<=n; i++) // First double loop
    for (j=1; j<=n; j++) // do n times
        sum1++;
sum2 = 0;
for (i=1; i<=n; i++) // Second double loop
    for (j=1; j<=i; j++) // do i times
        sum2++;
```

Other Control Statements

- while loop: analyze like a for loop.
- if statement: Take greater complexity of then/else clauses.
- switch statement: Take complexity of most expensive case.
- Subroutine call: Complexity of the subroutine.

Analyzing Problems

- Upper bound: Upper bound of best known algorithm.
- Lower bound: Lower bound for every possible algorithm.

Space Bounds

- Space bounds can also be analyzed with asymptotic complexity analysis.
- Time: Algorithm
- Space: Data Structure

the maximum contiguous subsequence sum problem

- if the input is {–2, **11**, **–4**, **13**, –5, 2}, then the answer is 20, which represents the contiguous subsequence encompassing items 2 through 4 (shown in boldface type).

- As a second example, for the input { 1, –3, **4**, **–2**,**–1**, **6** }, the answer is 7 for the subsequence encompassing the last four items.

- The problem statement gives a maximum contiguous subsequence sum of 0 for the case in which all input integers are negative.

- the obvious $O(N^3)$ algorithm
- an improved $O(N^2)$ algorithm
- a linear algorithm