Introduction to Algorithms

Department of Computer Science and Engineering

East China University of Science and Technology

sort

1、Insertion sort

- **Input:** sequence $<a_1, a_2, \ldots, a_n>$ *of n natural numbers*
- **Output:** permutation $<a'_1, a'_2, \ldots, a'_n>$ *such that* $a'_1 \le a'_2 \le \ldots \le a'_n$

- **Example**
- **– Input: <5, 2, 4, 6, 1, 3>**
- **– Output: <1, 2, 3, 4, 5, 6>**

Insertion sort

```
INSERTION-SORT(A)
for j ← 2 to length(A)
    do key ← A[j]
        // insert A[j] into the sorted sequence A[1..j-1]
        i ← j – 1
        while i > 0 and A[i] > key
            do A[i+1] ← A[i]
                i ← i – 1
        A[i+1] ← key
```

Pseudocode

// move item back

//find the insertion position

Example of insertion sort

8 2 4 9 3 6

Example of insertion sort

8 2 4 9 3 6

Example of insertion sort

8 2 4 9 3 6

2 8 4 9 3 6

Example of insertion sort

8 2 4 9 3 6
2 8 4 9 3 6

Example of insertion sort

8 2 4 9 3 6
2 8 4 9 3 6
2 4 8 9 3 6

Example of insertion sort

8 2 4 9 3 6
2 8 4 9 3 6
2 4 8 9 3 6

Example of insertion sort

8 2 4 9 3 6
2 8 4 9 3 6
2 4 8 9 3 6
2 4 8 9 3 6

Example of insertion sort

8 2 4 9 3 6
2 8 4 9 3 6
2 4 8 9 3 6
2 4 8 9 3 6

Example of insertion sort

8 2 4 9 3 6
2 8 4 9 3 6
2 4 8 9 3 6
2 4 8 9 3 6
2 3 4 8 9 6

Example of insertion sort

8 2 4 9 3 6
2 8 4 9 3 6
2 4 8 9 3 6
2 4 8 9 3 6
2 3 4 8 9 6

Example of insertion sort

8 2 4 9 3 6
2 8 4 9 3 6
2 4 8 9 3 6
2 4 8 9 3 6
2 3 4 8 9 6
2 3 4 6 8 9 ***done***

2、Correctness

loop invariant

- **Initialization:** It is true prior to the first iteration of the loop.
- **Maintenance:** If it is true before an iteration of the loop, it remains true before the next iteration.
- **Termination:** When the loop terminates, the invariant. usually along with the reason that the loop terminated. gives us a useful property that helps show that the algorithm is correct.

For insertion sort:

- **Initialization:** Just before the first iteration, $j = 2$.

For insertion sort:

- **Maintenance:**

For insertion sort:

- **Termination:** The outer **for** loop ends when $j > n$; this occurs when $j = n + 1$.

3、Analyzing algorithms

- predict the resources that the algorithm requires.

  running time.

- computational model

  Random-access machine (RAM) model

How do we analyze an algorithm's running time?

- depends on the input:
  an already sorted sequence is easier to sort.
- depends on the size of the input:
  short sequences are easier to sort than long ones.
- want upper bounds on the running time.
  --- guarantee to user.

Kinds of analyses

Worst-case: (usually)
- **$T(n)$ = maximum time of algorithm on any input of size n.**

Average-case: (sometimes)
- **$T(n)$ = expected time of algorithm over all inputs of size n.**
- **Need assumption of statistical distribution of inputs.**

Best-case: (bogus)
- **Cheat with a slow algorithm that works fast on *some* input.**

Worst-case analysis

***worst-case running time*: the longest running time for *any* input of size *n*.**

- **Why not analyze the average case?**

– Upper bound on the running time for any input
– For some algorithms, worst-case occur fairly often.
– Average case often as bad as worst case (but not always!)

Asymptotic analysis

Order of Growth

We will only consider order of growth of running time:
– We can ignore the lower-order terms, since they are relatively insignificant for very large *n*.
– We can also ignore leading term's constant coefficients, since they are not as important for the rate of growth in computational efficiency for very large *n*.
– We just said that best case was linear in *n* and worst/average case quadratic in *n*.

Running time

➢On a particular input, it is the number of primitive operations (steps) executed.

The running time of the algorithm is

$$\sum_{\text{all statements}} (\text{cost of statement})$$

INSERTION-SORT*(A)*	*cost*	*times*
1 for $j \leftarrow 2$ to n	c_1	n
2 do $key \leftarrow A[j]$	c_2	$n-1$
3 Insert $A[j]$ into the sorted sequence $A[1..j-1]$.	0	$n-1$
4 $i \leftarrow j-1$	c_4	$n-1$
5 while $i > 0$ and $A[i] > key$	c_5	Σt_j
6 do $A[i+1] \leftarrow A[i]$	c_6	$\Sigma(t_j - 1)$
7 $i \leftarrow i-1$	c_7	$\Sigma(t_j - 1)$
8 $A[i+1] \leftarrow key$	c_8	$n-1$

t_j : the number of times the while loop test in line 5 is executed for* that value of *j

- Let *T (n)* = running time of INSERTION-SORT.

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) \\ &\quad + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1) . \end{aligned}$$

Best case: The array is already sorted.

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1)$$

$$= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8)$$

$$T(n) = an + b$$

➢ ***T (n)* is a *linear function* of *n*.**

Worst case: The array is in reverse sorted order.

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 (n(n+1)/2 - 1) + c_6 (n(n-1)/2) + c_{7(} n(n-1)/2) + c_8(n-1)$$

$$=(c_5/2 + c_6/2 + c_7/2)\, n^2 + (c_1 + c_2 + c_4 + c_5/2 - c_6/2 - c_7/2 + c_8)n - (c_2 + c_4 + c_5 + c_8) .$$

$$T(n) = an^2 + bn + c$$

➢ ***T (n) is a quadratic function of n.***

Asymptotic analysis

算法复杂性在渐近意义下的阶：

渐近意义下的记号：O、Ω、θ、o、ω
设f(n)和g(n)是定义在正数集上的正函数。

算法复杂性分析

算法复杂性在渐近意义下的阶：

渐近意义下的记号：O、Ω、θ、o
设f(N)和g(N)是定义在正数集上的正函数。

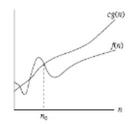
O的定义：如果存在正的常数C和自然数N_0，使得当$N \geq N_0$时有f(N)≤Cg(N)，则称函数f(N)当N充分大时上有界，且g(N)是它的一个上界，记为f(N)=O(g(N))。即f(N)的阶不高于g(N)的阶。

根据O的定义，容易证明它有如下运算规则：
(1)O(f)+O(g)=O(max(f,g))；
(2)O(f)+O(g)=O(f+g)；
(3)O(f)O(g)=O(fg)；
(4)如果g(N)=O(f(N))，则O(f)+O(g)=O(f)；
(5)O(Cf(N))=O(f(N))，其中C是一个正的常数；
(6)f=O(f)。

33

O-notation

- $O(g(n))$ = {$f(n)$: there exist positive constants c and n_0 such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$} .



– O(.) is used to asymptotically upper bound a function.
– O(.) is used to bound worst-case running time.

- $g(n)$ is an ***asymptotic upper bound*** for $f(n)$.
- If $f(n) \in O(g(n))$, we write $f(n) = O(g(n))$

算法复杂性分析

Ω的定义：如果存在正的常数C和自然数N_0，使得当$N \geq N_0$时有f(N)≥Cg(N)，则称函数f(N)当N充分大时下有界，且g(N)是它的一个下界，记为f(N)=Ω(g(N))。即f(N)的阶不低于g(N)的阶。

θ的定义：定义f(N)= θ(g(N))当且仅当f(N)=O(g(N))且f(N)= Ω(g(N))。此时称f(N)与g(N)同阶。

o的定义：对于任意给定的ε>0，都存在正整数N_0，使得当N≥N0时有f(N)/Cg(N)≤ε，则称函数f(N)当N充分大时的阶比g(N)低，记为f(N)=o(g(N))。
例如，4NlogN+7=o($3N^2$+4NlogN+7)。

35

Example

1 $2n^2 = O(n^3)$
with $c = 1$ and $n_0 = 2$.

2 $1/3n^2 - 3n \in O(n^2)$
because $1/3n^2 - 3n \leq cn^2$ if $c = 1/3$ and $n > 1$.

3 $an^2 + bn + d \in O(n^2)$
because $an^2 + bn + d \leq (a + |b| + |d|)n^2$ and for $c > a + |b| + |d|$ and $n \geq 1$, $an^2 + bn + d \leq cn^2$.

Note:

– When we say "the running time is O(n^2)" *we mean that* the worst-case running time is O(n^2) – *the best case* might be better.

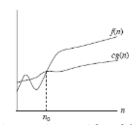
– Use of O-notation often makes it much easier to analyze algorithms; we can easily prove the O(n^2) insertion-sort time bound.

– We often abuse the notation a little:

- We often use O(n) *in equations:*
- *e.g.* $2n^2 + 3n + 1 = 2n^2 + O(n)$
- We use O(1) to denote constant time.

Ω-notation

- $\Omega(g(n)) = \{ f(n)$: there exist positive constants c and n_0 such that $0 \le cg(n) \le f(n)$ for all $n \ge n_0\}$.



- $g(n)$ is an ***asymptotic lower bound*** for $f(n)$.

Example

1 $1/3n^2 – 3n = \Omega(n^2)$
because $1/3n^2 – 3n \ge cn^2$ *if* $c = 1/6$ and $n > 18$.

2 $an^2 + bn + k = \Omega(n^2)$

3 $an^2 + bn + k = \Omega(n)$ *(lower bound)*

Note:

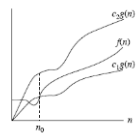
– When we say "the running time is Ω(n^2)" we mean that the best-case running time is Ω(n^2) – the worst case might be worse.

– Insertion-sort:

- Best case: Ω(n) – when the input array is already sorted.
- Worst case: O(n^2) – when the input array is reverse sorted.
- We can also say that the worst case running time is Ω(n^2).

Θ-notation

- $\Theta(g(n))$ = { $f(n)$: there exist positive constants c_1, c_2, and n_0 such that $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ for all $n \ge n_0$} .



- $g(n)$ is an ***asymptotically tight bound*** for $f(n)$.

Example

$$n^2/2 - 2n = \Theta(n^2)$$

$c_1 = 1/4$ $c_2 = 1/2$ $n_0 = 8$

2 $1/3n^2 - 3n \in \Theta(n^2)$
$c_1 = 1/6$ $c_2 = 1/3$ and $n_0 > 18$.

o-notation

- $o(g(n))$ = { $f(n)$: for all constants $c > 0$, there exists a constant $n_0 > 0$ such that $0 \le f(n) < cg(n)$ for all $n \ge n_0$} .
- ***bn + k = o(n²)***
- ***N→enough bigger Lim bn/o(n²) = 0***

ω-notation

- $\omega(g(n))$ = { $f(n)$: for all constants $c > 0$, there exists a constant $n_0 > 0$ such that $0 \le cg(n) < f(n)$ for all $n \ge n_0$}

- 非渐近紧确的下界
- ***b n² + k = ω (n)***
- ***Lim bn² /ω (n)= 无究大***