CSCI 6470 Algorithms (Fall 2024)

Liming Cai

School of Computing UGA

August 16, 2024

Chapter 1 Fundamentals

1. Worst-case time complexity
2. The big-O notation
3. Series and recurrence relations
4. Time complexity of recursive algorithms

1. Worst-case time complexity

- **Basic operations**: arithmetic ops, logic ops, assignment, branching in high-level programming language
  - Corresponding operations in assembly (machine) languages and corresponding micro-instructions

    Example: $C = A + B$ is compiled into assembly code and executed with micro-instructions

1. Worst-case time complexity

- **Basic operations**: arithmetic ops, logic ops, assignment, branching in high-level programming language
  - Corresponding operations in assembly (machine) languages and corresponding micro-instructions

    Example: $C = A + B$ is compiled into assembly code and executed with micro-instructions

  - Concept of machine cycle

    real time = the number of machine cycles needed to execute the basic operations in algorithm A

    but the number of machine cycles differ across different computers and system platforms, not suitable for measuring time complexity of algorithms

1. Worst-case time complexity

- **Time** (**complexity**) of an algorithm A on input x is the number of **basic operations** carried out by A on x, denotes as function $t(n, x)$, where n is the **size** of x.
    - instead of the number of machine cycles required for running the basic operations.
    - however, $t(x, n)$ is x (content)-dependent
- **The worst case time** complexity of algorithm A is function $T(n)$, such that for every $n \geq 0$,

$$\forall x, \text{ of size } n, t(n, x) \leq T(n)$$

independent of the content of input

1. Worst-case time complexity

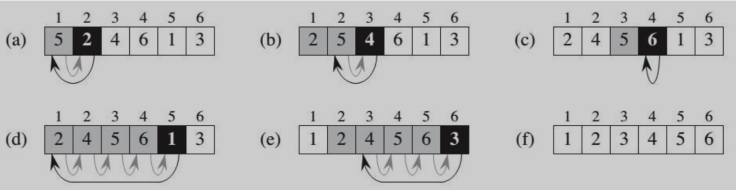
Example 1. Find $T(n)$ for iterative INSERTION SORT algorithm

But first, what is the idea of the insertion sort?

1. Worst-case time complexity

Example 1. Find $T(n)$ for iterative Insertion Sort algorithm

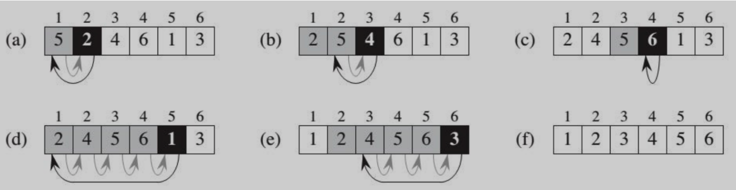
But first, what is the idea of the insertion sort?



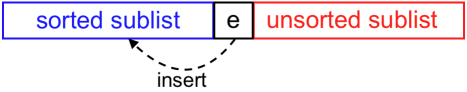
1. Worst-case time complexity

Example 1. Find $T(n)$ for iterative INSERTION SORT algorithm

But first, what is the idea of the insertion sort?



Schematic representation of the dynamic of insertion sort:



1. Worst-case time complexity

Algorithm INSERTION SORT

1. Worst-case time complexity

Algorithm INSERTION SORT

```
Function Insertion Sort(L, n);
for i = 2 to n
    e = L[i];
    j = i-1;
    while (L[j] > e) AND (j>0)
        L[j+1] = L[j];
        j = j - 1;
    L[j+1] = e;
return
```

1. Worst-case time complexity

Algorithm Insertion Sort

```
Function Insertion Sort(L, n);
1. for i = 2 to n
2.     e = L[i];
3.     j = i-1;
4.     while (L[j] > e) AND (j>0)
5.         L[j+1] = L[j];
6.         j = j - 1;
7.     L[j+1] = e;
8. return
```

- Count the number of basic operations:

  Line 1: $2 \times n + (n-1)$

  2,3,7: $2 \times (n-1)$

  4: $2 \times t_j$ $\longleftarrow$ t_j dependent on j, overall on input x

  5: $(t_j - 1)$

  6: $2 \times (t_j - 1)$

  Line 8: 1

1. Worst-case time complexity

- Count the number of basic operations:

$$t(n, x) = an + b + \sum_{j=1}^{n-1} c \times t_j$$

for some constants $c > 0$, a, and b

- Because t_j can only be as worst (big) as j

$$t(n, x) \leq an + b + \sum_{j=1}^{n-1} c \times j = T(n)$$

$$T(n) = c\frac{n-1}{2}n + an + b = \frac{c}{2}n^2 + (a - \frac{c}{2})n + b$$

$$= c_1 n^2 + c_2 n + c_3$$

1. Worst-case time complexity

Additional issues

- About time used for arithmetic operations

  e.g., $A + B$, where A and B are of scale $2^{1000000}$;
  time needed is $c \times \frac{1000000}{64} = c_1 \times 1000000$,
  the time complexity is related to the binary length of data

1. Worst-case time complexity

Additional issues

- About n, the **size** of input x, what does **size** refer to?

  (1) size n refers to the number of data items in the input as in INSERTION SORT

  Consider to sort 4 very large elements, e.g., of scale $2^{1000000}$
  $T(n) = c_1 n^2 + c_2 n + c_3$, then $T(4)$ is a small constant time,
  However, this is not an accurate measure because even just comparison of two large elements takes $c \times 1000000$ steps

1. Worst-case time complexity

(2) size n is the number of binary bits that encode the input x, denoted as $n = |x|$

then for INSERTION SORT on m elements, time is bounded by

$$= c_1 m^2 |x| + c_2 m |x| + c_3 |x|$$

$$\leq c_1 n^3 + c_2 n^2 + c_3 n = T(n)$$

Why?

Exercise: given algorithm

```
Function Fibonacci (x);
1. F[1] = 1;
2. F[2] = 1;
3. for i = 3 to x
4.     L[i] = L[i-1] + L[i-2];
5. return L[x];
```

What is $T(n)$ for `Fibonacci`? Is it really a linear function?

1. Worst-case time complexity

Additional issues

(3) How to find a simple upper bound for time expressions?

e.g., worst case time for INSERTION SORT

$$\begin{aligned} T(n) &= c_1 n^2 + c_2 n + c_3 \\ &\le (c_1 + c_2 + c_3) n^2 \\ &= cn^2 \end{aligned}$$

<u>**Exercise**</u>: find a simple upper bound for

$$T(n) = 5n^2 + 4n \log_2 n - 20n + 89$$

2. The big-O notation