Designing and Analysing Algorithms. Insertion sort algorithm

The Sorting Problem

Input: a sequence of n numbers $\langle a_1, a_2, \ldots, a_n \rangle$

Output: a permutation (or reordering) of the numbers in increasing order $\langle a_1', a_2', \ldots, a_n' \rangle$, that is $a_1' \le a_2' \le \ldots \le a_n'$

Algorithm: well-defined computational procedure that takes a value (or set of values) as input and produces a value (or set of values) as output.

Data structures:

- provide a way to store and organize data
- operations used to access and modify data

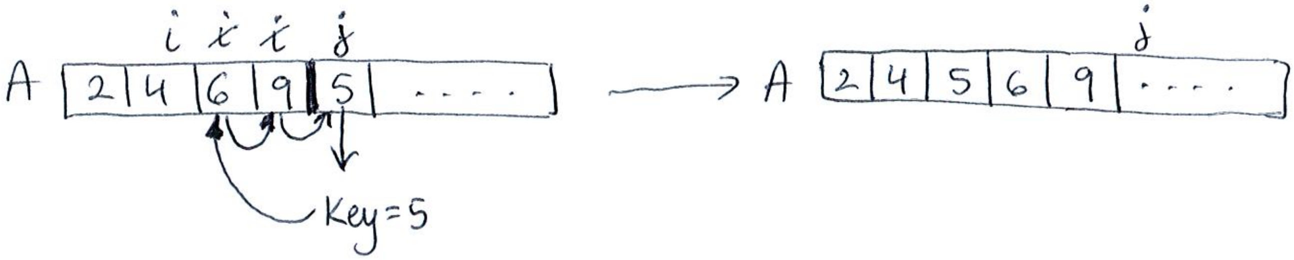
The Insertion sort algorithm

INSERTION-SORT(A)

```
for j = 2 to A.length
    key = A[j]
    //insert A[j] into the sorted sequence A[1..j-1]
    i = j-1
    while i>0 and A[i] > key
        A[i+1] = A[i]
        i = i-1
    A[i+1] = key
```

cost	times
c_1	n
c_2	$n-1$
0	$n-1$
c_4	$n-1$
c_5	$\sum_{j=2}^{n} t_j$
c_6	$\sum_{j=2}^{n} (t_j - 1)$
c_7	$\sum_{j=2}^{n} (t_j - 1)$
c_8	$n-1$

Example of one element insertion



Pseudocode conventions

- use indentation to show block structure
- arrays start from index 1
  e.g. $A[1..n]$, subarray $A[4..9]$
- use // for comment

Observation

- in a <u>for</u> loop, the instruction in the header executes one more time than the instructions in the body of the loop.

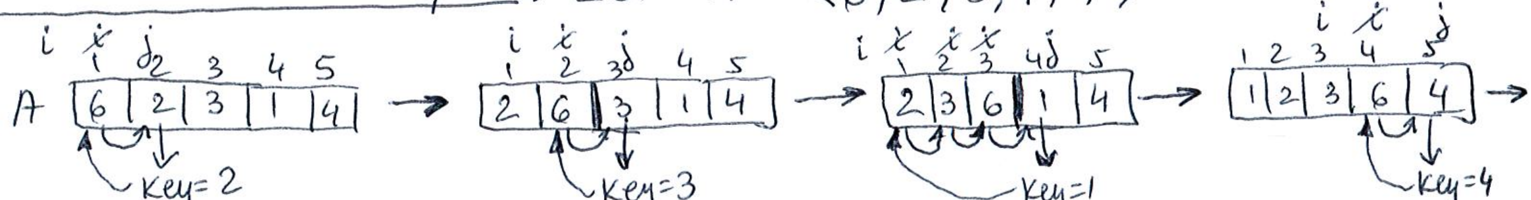
<u>example</u>

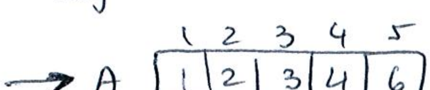
<u>for</u> $i = 2$ to 5	5 times
$x = x+1$	4 times
$y = y-2$	4 times

i takes the values: 2, 3, 4, 5, 6. Last time ($i=6$) the test in the header evaluates to false.

- the same observation applies to <u>while</u> loops.

<u>Insertion sort example</u>. Let $A = \langle 6, 2, 3, 1, 4 \rangle$





sorted array!

Properties

- incremental approach
- Sorting in place: at most a constant number of elements are stored outside of the array at any time.

Correctness

- an algorithm is <u>correct</u> if it halts and if it produces the correct answer

<u>Loop Invariant (LI)</u> - statement which is true at the start of each iteration of the loop.

- <u>Initialization</u> - LI is true at the start of the first iteration
- <u>Maintenance</u> - if the LI is true at the start of an iteration, then it remains true at the start of the next iteration
- <u>Termination</u> - write the LI when the loop terminates. Use this statement to show alg. correctness.

Insertion sort algorithm

<u>Loop Invariant</u>: At the start of each iteration j of the <u>for</u> loop, the subarray $A[1..j-1]$ contains the elements originally in $A[1..j-1]$ but in sorted order.

- Initialization

  $j = 2$

  $A[1..j-1] = A[1]$ original elm. + sorted ✓

- Maintenance

  assume $A[1..j-1]$ are original elms. + sorted
  j^{th} iteration places $A[j]$ in the correct location $\Big\} \Rightarrow$

  $\Rightarrow A[1..j]$ are original elms. + sorted

- Termination

  let $n = A.length$

  when the loop terminates, $j = n+1$

  LI statement : $A[1..j-1] = A[1..n]$ are original elms. + sorted

  alg. correctness!

Analyzing Algorithms

- predict the resources the alg. requires: computational time (or running time), memory, bandwidth, so on.

Random-Access Machine (RAM) model

- instructions are executed one after another
- all primitive instructions take a constant time:
  - arithmetic : $+, -, *, /$, remainder, $\lfloor\ \rfloor, \lceil\ \rceil$
  - data movement : load, store, copy
  - control : conditional/unconditional branch, subroutine call and return

- integer and floating-point data types

<u>How do we express the running time (RT)?</u>

- express the RT using <u>asymptotic notations</u>, as a function of the <u>input size</u>

<u>Input size</u>: - depends on the problem being studied
- for the sorting problem, it is n - the number of elements to be sorted

$T(n)$ - running time for an input size n

$$\boxed{T(n) = \sum_{\text{all statements}} (\text{cost of the statement}) \cdot (\text{no. of times the statement is executed})}$$

<u>RT for Insertion sort</u>

let $n = A.length$

<u>for</u> loop: j takes the values $2, 3, 4, \ldots, n+1$

let t_j - number of times the header of the <u>while</u> loop executes in the j^{th} iteration

Then the <u>while</u> loop header executes $(t_2 + t_3 + t_4 + \ldots + t_n)$ times, which can be written as $\sum_{j=2}^{n} t_j$

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

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

Best case

$t_j = 1$ for all $j = 1, 2, \ldots, n$

- when the input array is already sorted in increasing order

$$\sum_{j=2}^{n} t_j = \sum_{j=2}^{n} 1 = \underbrace{1 + 1 + \ldots + 1}_{(n-1) \text{ times}} = n - 1$$

$$\sum_{j=2}^{n} (t_j - 1) = \sum_{j=2}^{n} 0 = 0$$

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

$$\left.\begin{array}{l} \boxed{T(n) = a \cdot n + b} \\ a, b \text{ - constants} \end{array}\right\} \Rightarrow T(n) \text{ is a } \underline{\text{linear function}} \text{ of input size } n$$

Worst case

- when the input array is sorted in decreasing order

then $t_j = j$ for all $j = 1, 2, \ldots, n$

Arithmetic series : $\boxed{1 + 2 + 3 + \ldots + n = \frac{n \cdot (n+1)}{2}}$

$$\sum_{j=2}^{n} t_j = \sum_{j=2}^{n} j = 2 + 3 + 4 + \ldots + n = \frac{n(n+1)}{2} - 1$$

arithmetic series

$$\sum_{j=2}^{n} (t_j - 1) = \sum_{j=2}^{n} (j - 1) = 1 + 2 + 3 + \ldots + (n-1) = \frac{(n-1) \cdot n}{2}$$

arithmetic series

$$T(n) = (c_1 + c_2 + c_4 + c_8)n - (c_2 + c_4 + c_8) + c_5\left(\frac{n(n+1)}{2} - 1\right) +$$
$$+ (c_6 + c_7)\frac{(n-1)n}{2}$$

$$T(n) = \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + c_8 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2}\right)n -$$
$$- (c_2 + c_4 + c_8 + c_5)$$

$$\left.\begin{array}{l}\boxed{T(n) = an^2 + bn + c} \\ a, b, c - \text{constants}\end{array}\right\} \Rightarrow T(n) \text{ is a quadratic function of input size } n$$

Order of growth

- computed as follows:
  - drop lower-order terms
  - ignore the constant in the leading term

Worst case RT Insertion sort:

$\Rightarrow an^2$

$\Rightarrow n^2$

$$\boxed{T(n) = \Theta(n^2)}$$

- worst case RT for Insertion sort has the order of growth n^2

An algorithm is more efficient than another algorithm if its worst case running time has a smaller order of growth.