Algorithm efficiency: introduction

1. **What is algorithm efficiency?**
2. **Pseudocode analysis**

1. What is algorithm efficiency?

Two quantities of interest:

• How much time does the algorithm require?

• How much computer memory does the algorithm require?

Focus on **time**.

Q: For some given computational task what do we really want?

A: We want to know **beforehand** how much time the task will take.

Software/hardware factors which will affect the time:

• a particular algorithm for doing the task

• a particular programming language

• a particular implementation of the algorithm in the programming language with error checks, data types, objects, etc.

• particular set of input data

• a particular compiler with all compiler options

• a particular computer with a particular CPU, cache memory, clock frequency, etc.

• a particular environment: what else is computer doing?

Too difficult to predict beforehand.

We can measure the computational time afterwards, once all factors have been fixed.

Does this help?

Not very often. Suppose one or more factors change.

Q: What can we do?

A: Analyze pseudocode.

Pseudocode ignores most of the above factors. What factors are left?

• a particular algorithm

• a particular set of input data

2. Pseudocode analysis

Q: What is pseudocode analysis?

A: counting the number of 'simple operations' and seeing how this depends on the 'size' of the input data

Q: What is a 'simple operation'?

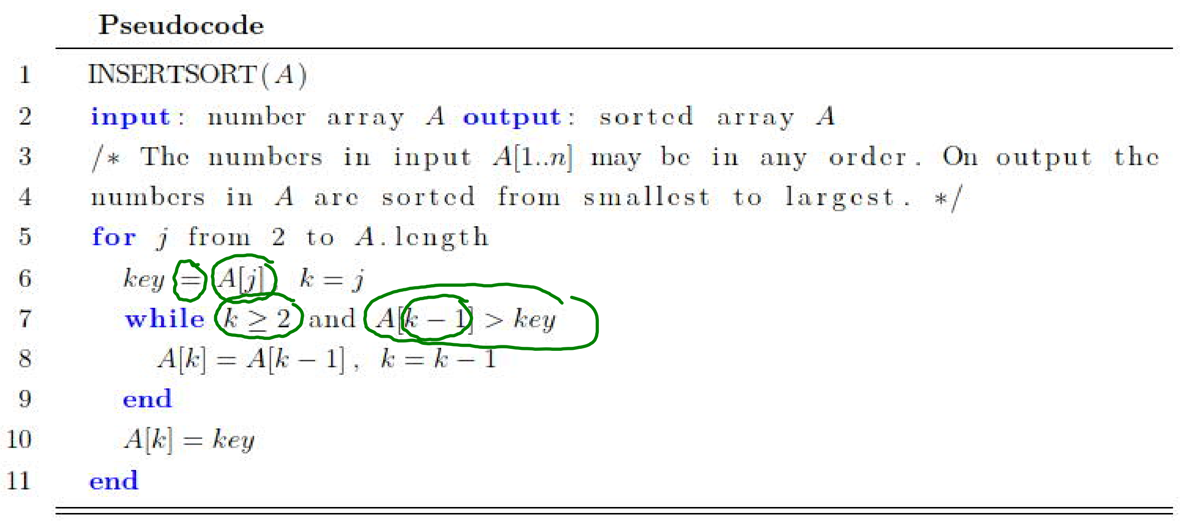
A: Any operation that does not depend on size of input data.

Simple operations:

- arithmetic operations: +, -, *, /
- **if**-statement, **else**-statement
- one iteration of **for** or **while** or **for-each**
- variable assignment
- accessing a single item in memory
- a single call to a procedure (NOT the execution of the procedure itself)

RAM-model of computer: all simple operations take the same time

Example



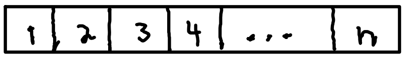
Pseudocode

```
INSERTSORT(A)
input: number array A output: sorted array A
/* The numbers in input A[1..n] may be in any order. On output the
numbers in A are sorted from smallest to largest. */
for j from 2 to A.length
  $key \boxed{=} \boxed{A[j]}$,  $k = j$
  while $\boxed{k \geq 2}$ and $\boxed{A[\boxed{k-1}] > key}$
    $A[k] = A[k-1]$,  $k = k-1$
  end
  $A[k] = key$
end
```

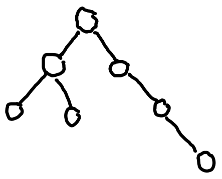
□

What is the 'size' of input data?

- if input is array, then usually length of array



- if input polynomial, then usually degree of polynomial $a_0 + a_1x + \dots + a_nx^n$
- if input is a number, then often the number of bits needed to represent number 10110101
- if input is binary tree, then sometimes the height of the tree, sometimes the number of nodes (vertices)



Goal of pseudocode analysis is *f(n)*

f(n) = total count of simple operations for input data of size *n*

Asymptotic analysis: only consider those parts *f(n)* that grows the fastest as *n* increases

Example

Pseudocode

```
INSERTSORT(A)
input: number array A output: sorted array A
/* The numbers in input A[1..n] may be in any order. On output the
numbers in A are sorted from smallest to largest. */
for j from 2 to A.length
   $key = A[j]$, $k = j$
   while $k \geq 2$ and $A[k-1] > key$
      $A[k] = A[k-1]$, $k = k-1$
   end
   $A[k] = key$
end
```

Count of times line 8 executed in INSERTSORT

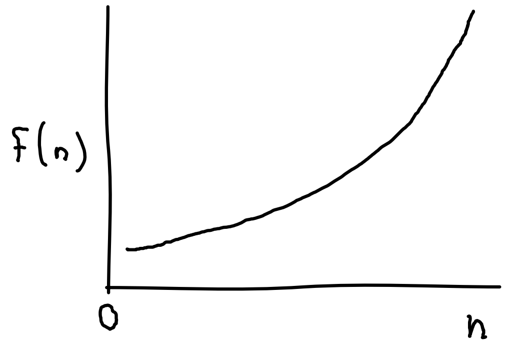
j	most number of times line 8 executed
n	$(n-1)$
$n-1$	$(n-2)$
$\vdots$	
i	$(i-1)$
$\vdots$	
2	1

$$\sum_{r=1}^{n-1} r = \frac{n(n-1)}{2}$$

$f_8(n) =$ number of simple operations needed to execute line 8 in worst case

$$= \frac{5n(n-1)}{2}$$

$$= \frac{5}{2}n^2 - \frac{5}{2}n$$



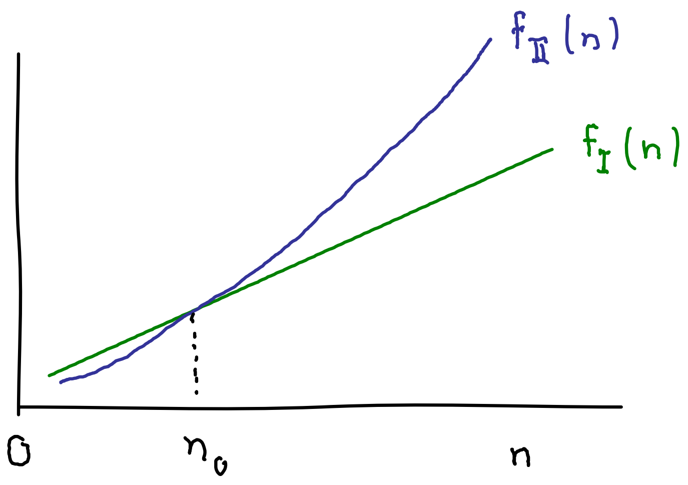
$\square$

Q: Can pseudocode analysis help with our original task of predicting how long some computation task will take?

A: No.

Q: What can we do with pseudocode analysis?

A: Compare alternative algorithms.


$f_{II}(n)$
$f_{I}(n)$
n_0
n