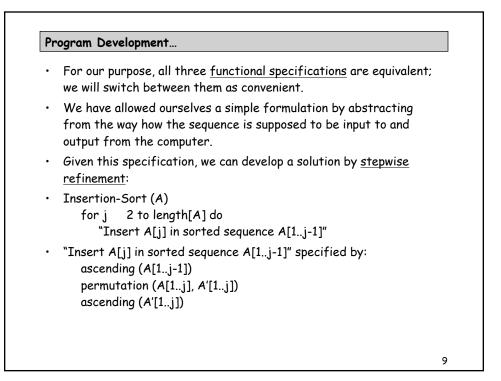
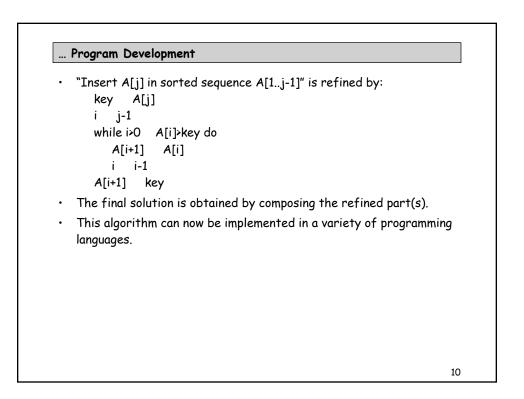
1. Introduction: Analysing and Designing Algorithms

- When solving a problem, we are well advised first to construct an exact model in terms of which we can express allowed solutions.
- Finding such a model is already half the solution. Any branch of mathematics or science can be called into service to help model the problem domain, e.g.:
 - Simultaneous linear equations (finding currents in electrical circuits, finding stresses in frames made of concrete beams)
 - Differential equations (predicting population growth, predicting the rate at which chemicals will react)
 - Formal grammars (compiling programming languages, database queries)
 - Graphs (transportation problems, optimal scheduling)
- Once we have a suitable mathematical model, we can specify a solution in terms of that model.

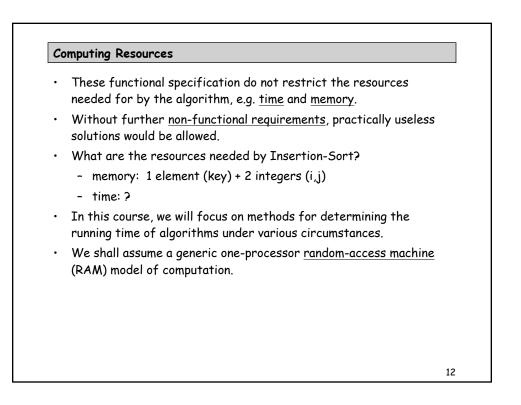
In	itroductory Example	
•	Suppose we model our problem domain by a sequence of numbers $A = a_1, a_2,, a_n$ and the solution consists in sorting them.	
•	Functional specification in terms of an abstract program (pseudocode):	
	A any A' such that permutation (A', A) ascending (A')	
•	Functional specification in terms of a relation:	
	permutation (A', A) ascending (A')	
•	Functional specification in terms of pre- and postcondition:	
	{A=X} sort {permutation (X, A) ascending (A)}	
	Note that the "logical variable" X is necessary to express the input-output relationship.	
•	Definitions: permutation: smallest relation satisfying for any x, s, t ₁ ,t ₂ permutation (,) permutation (x ° s, t ₁ ° x ° t ₂) permutation (s, t ₁ ° t ₂)	
	ascending (s) ($i 1 i < \text{length} [s] \cdot s[i] s_{+1}$)	
		8

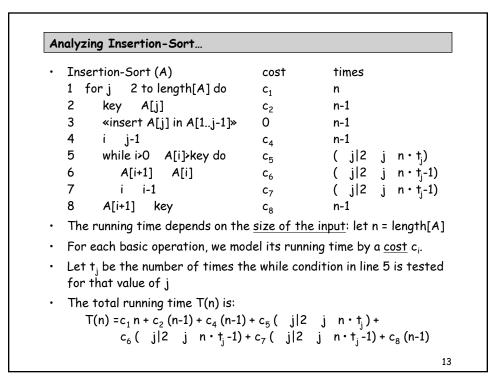


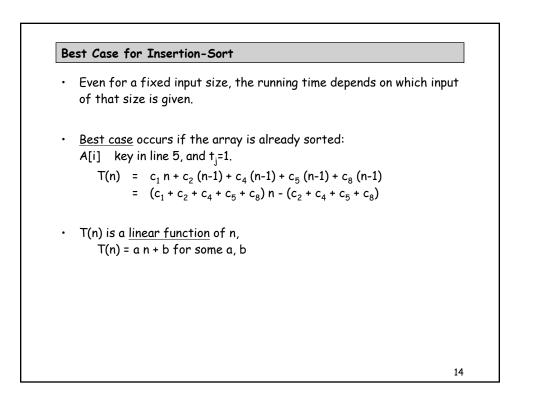


Implementation in Pascal

```
type T = array [1..N] of integer;
•
   procedure insertionsort (var A: T);
      var key, j, i: integer;
   begin
      for j \coloneqq 2 to N do
          begin
             key := A[j];
             i := j-1;
             while (i>0) and (A[i]>key) do
                 begin
                    A[i+1] := A[i];
                    i := i-1
                 end;
             A[i+1] := key
          end
   end;
```







Worst Case for Insertion-Sort

Worst case occurs if the array is in reverse sorted order: we must compare A[j] with all A[j-1], ..., A[1], so t_j = j
Noting that

(j|2 j n · j) = n (n+1) / 2 -1
(j|2 j n · j-1)= n (n-1) / 2

we get for T(n) in that case:

T(n) = (c₅ / 2 + c₆ / 2 + c₇ / 2) n² + (c₁ + c₂ + c₄ + c₅ / 2 + c₆ / 2 + c₇ / 2 + c₈) n - (c₂ + c₄ + c₅ + c₈)

T(n) is a <u>quadratic function</u> of n, T(n) = a n² + b n + c for some a, b, c

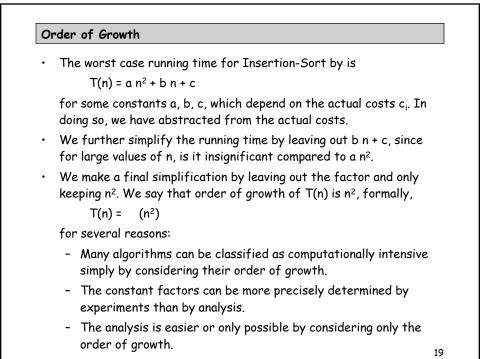
1	erage Case for Insertion-Sort
	For the <u>average case</u> , we randomly choose n numbers.
	When comparing A[j] with A[1],, A[j-1], in average half of the elements are less than A[j]. Hence $t_j = j / 2$.
	The analysis of T(n) in that case is similar to the worst case, except for the factor 2.
	In the average case, T(n) is also a quadratic function of n.

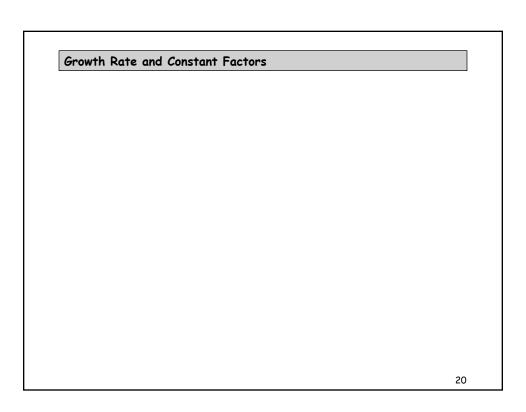
Algorithms ...

•

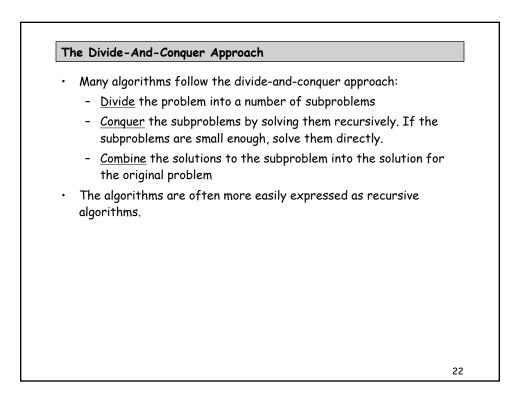
- To summarize, an algorithm is a program for a (possibly abstract) machine, for which we can ensure the correctness in terms of the model of the problem domain.
- Besides correctness, we are interested in the worst-case and the average-case running time with respect to an abstract computer, the random access machine.
 - We mostly consider only the worst-case running time:
 - It gives an upper bound on the running time.
 - The worst case occurs often, e.g. searching an element which is not present.
 - The average case is often roughly as bad as the worst case.
 - It is not clear what the average input is.
- For analyzing the running time and for implementing the algorithm, it has to be in a sufficiently refined form so that constructs can be faithfully mapped to the available machine (programming language).

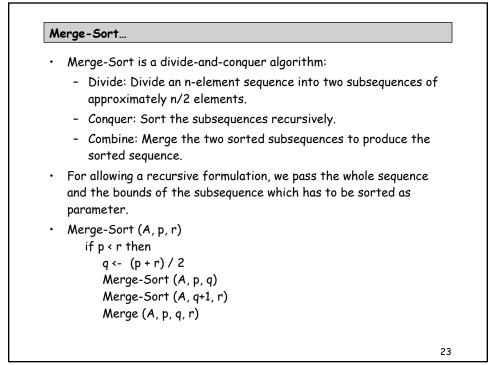
•		(n), for example q	algorithms, we often consider Juadratic (n²), n lg n, or linear s is decisive term.			
•	behaves naturally in	n that it is faster particular if we h	for example, Insertion-Sort if the array is already ave a sorted sequences with t the end.			
		thms do better. I	ng time in the order of n², However, other faster sorting y as Insertion-Sort.			
•	In designing algorithms, we prefer for loops to while loops. They guarantee termination and make running time easier to analyze: for i a to b do S(i) =					
	skip		if a>b			





Suppose ea	ch operati	on takes	1 nanoseco	onds (10 ⁻⁹	seconds	5)
n	lg n	n	n lg n	n²	2 ⁿ	n!
10	0.003 <i>µ</i> s	0.01 <i>µ</i> s	0.033 <i>µ</i> s	0.1 <i>µ</i> s	1 <i>µ</i> s	3.63ms
20	0.004 <i>µ</i> s	0.02 <i>µ</i> s	0.086 <i>µ</i> s	0.4 <i>µ</i> s	1ms	77.1years
30	0.005 <i>µ</i> s	0.02 <i>µ</i> s	0.147 <i>µ</i> s	0.9 <i>µ</i> s	1sec	>10 ¹⁵ years
100	0.007 <i>µ</i> s	0.1 <i>µ</i> s	0.644 <i>µ</i> s	10 <i>µ</i> s	>1013ye	ars
10,000	0.013 <i>µ</i> s	10 <i>µ</i> s	130 <i>µ</i> s	100ms		
1,000,000	0.020 <i>µ</i> s	1ms	19.92 <i>µ</i> s	16.7min		
 (n!) al (2ⁿ) a (n²) a 	gorithms o Igorithms	are usele: are pract	s insignific ss well bef tical for n th useful,	ore n = 20 < 40.		ignificantly
faster.						





•	The auxiliary procedure Merge (A, p, q, r) merges the sorted subsequences A[pq] and A[q+1r]. It can be specified by:	
	p q q r ascending (A[pq]) ascending (A[q+1r]) permutation (A'[pr], A[pr]) ascending (A'[pr])	
•	We assume that it can be implemented in (n), for example by using an auxiliary sequence.	
•	The entire sequence A can be sorted by calling Merge-Sort (A, 1, length[A])	
•	What is the running time of Merge-Sort?	
		24

Analyzing Divide-And-Conquer Algorithms

- Let T(n) be the running time for a problem of size n.
- If the problem is small, say n c, we assume that a direct solution takes constant time:

T(n) = (1) if n c

• Otherwise, we divide the problem into a subproblems, each of which is 1/b the size of the original. Suppose it takes D(n) time to divide the problem and C(n) time to combine the solutions.

$$T(n) = a T(n/b) + D(n) + C(n)$$
 if $n > c$

• Such equations are called <u>recurrences</u>. They are common in analyzing the running time of algorithms. We will study solutions of recurrences later.

•	Divide: Just computes the middle of the subse constant time: D(n) = (1)	quence, thus takes
•	Conquer: We solve 2 subproblems of size appro a = 2, b = 2	oximately n/2:
•	Combine: Merge takes (n): C(n) = (n)	
•	Noting that (n) + (1) is still (n), we get: T(n) = (1) if n = 1 2 T(n/2) + (n) if n > 1	
•	Later we will see that: T(n) = (n lg n)	

Remarks on Merge-Sort

- Merge-Sort has a running time of (n lg n) Insertion-Sort has a running time of (n²)
- This implies that for sufficiently large n, Merge-Sort is superior to Insertion-Sort. For a certain Pascal implementation, Merge-Sort is 7 times faster than Insertion-Sort for n=256 and 11 times faster for n=512.
- Merge-Sort takes approximately the same amount of time whether the sequence is sorted, random, or inversely sorted.
- Procedure Merge requires n elements extra memory. However, since the sequences are accessed only sequentially, Merge-Sort is better suited for <u>external sorting</u>. Later we will see that 3 or 4 files are sufficient.
- Versions of Merge exists which do not require extra memory at the cost of additional moves. However, other <u>internal sorting</u> algorithms are superior even to the fastest version of Merge-Sort

Complexity of Algori [•]	thms vs. Problems		
• The running time o <u>complexity</u> .	f an algorithm is als	o referred to as its <u>:</u>	time
The memory requin <u>space complexity</u> .	red by an algorithm	is also referred to as	s its
• Merge-Sort has a	time complexity of	(n lg n)	
Insertion-Sort has	a time complexity	of (n ²)	
algorithm? ANSWER: We will see faster than (n l <u>o</u> constant factors.), i.e. no algorithm car hms may still differ	
• We therefore can complexity of (n		m of sorting has a tir	ne