The Efficiency of Algorithms

Chapter 3

Topics:

Attributes of Algorithms

A Choice of Algorithms

Measuring Efficiency

Analysis of Algorithms

When Things Get Out of Hand

Attributes of Algorithms

- Correctness
  - Give a correct solution to the problem!
- Efficiency
  - Time: How long does it take to solve the problem?
  - Space: How much memory is needed?
  - Benchmarking vs. Analysis
- Ease of understanding
  - Program maintenance
- Elegance

A Choice of Algorithms

- Possible to come up with several different algorithms to solve the same problem.
- Which one is the "best"?
  - Most efficient
    - Time vs. Space?
  - Easiest to maintain?
- How do we measure time efficiency?
  - Running time? Machine dependent!
  - Number of steps?

The Data Cleanup Problem

- We look at three algorithms for the same problem, and compare their time- and space-efficiency.
- Problem: Remove 0 entries from a list of numbers.

0	12	32	71	34	0	36	92	0	13

⇩

12	32	71	34	36	92	13

1. The Shuffle-Left Algorithm

- We scan the list from left to right, and whenever we encounter a 0 element we copy ("shuffle") the rest of the list one position left.

0	12	32	71	34	0	36	92	0	13

⇩

12	32	71	34	36	92	13	13	13	13

0	12	32	71	34	0	36	92	0	13
12	32	71	34	0	36	92	0	13	13
12	32	71	34	0	36	92	0	13	13
12	32	71	34	36	92	0	13	13	13
12	32	71	34	36	92	0	13	13	13
12	32	71	34	36	92	13	13	13	13

Shuffle-Left Animation

Legit: 7

12	32	71	34	36	92	13	13	13	13	
							↑	↑		
							L	**R**		

2. The Copy-Over Algorithm

- We scan the list from left to right, and whenever we encounter a nonzero element we copy it over to a new list.

0	12	32	71	34	0	36	92	0	13

⇩

12	32	71	34	36	92	13

The Copy-Over Animation

0	12	32	71	34	0	36	92	0	13	
										↑
										L
12	32	71	34	36	92	13				
							↑			
							N			

3. The Converging-Pointers Algorithm

- We scan the list from both left (L) and right (R). Whenever L encounters a 0 element, the element at location R is copied to location L, then R reduced.

0	12	32	71	34	0	36	92	0	13

⇩

13	12	32	71	34	92	36	92	0	13

Converging Pointers Animation

Legit: 7

13	12	32	71	34	92	36	92	0	13	
						↑↑				
						L R				

Data-Cleanup Algorithm Comparison

- Which one is the most space efficient?
  - Shuffle-left — no additional space
  - Copy-over — needs a new list
  - Converging-pointers — no additional space
- Which one is the most time efficient?
  - Shuffle-left — many comparisons
  - Copy-over — goes through list only once
  - Converging-pointers — goes through list only once
- How do we measure time efficiency?

Exercise

- Can you come up with a more efficient algorithm for the data-cleanup problem, that does:
  - not require any additional space
  - less copying than shuffle-left
  - maintain the order of the none-zero elements
- Hint:
  - Can the copy-over algorithm be modified to copy the element into the same list?

Measuring Efficiency

- Need a metric to measure time efficiency of algorithms:
  - How long does it take to solve the problem?
    - Depends on machine speed
  - How many steps does the algorithm execute?
    - Better metric, but a lot of work to count all steps
  - How many "fundamental steps" does the algorithm execute?
- Depends on size and type of input, interested in knowing:
  - *Best*-case, *Worst*-case, *Average*-case behavior
- Need to analyze the algorithm!

Sequential Search

1. Get values for *Name*, $N_1, \ldots, N_n$, $T_1, \ldots, T_n$
2. Set the value *i* to 1and set the value of *Found* to NO
3. Repeat steps 4 through 7 until *Found* = YES or *i* > *n*
4. If *Name* = N_i then
5. Print the value of T_i
6. Set the value of *Found* to YES
   Else
7. Add 1 to the value of *i*
8. If Found = NO then print "Sorry, name not in directory"
9. Stop

Sequential Search - Analysis

- How many steps does the algorithm execute?
  - Steps 2, 5, 6, 8 and 9 are executed at most once.
  - Steps 3, 4, and 7 depends on input size.
- Worst case:
  - Step 3, 4, and 7 are executed at most n-times.
- Best case:
  - Step 3 and 4 are executed only once.
- Average case:
  - Step 3, 4 are executed approximately (n/2)-times.
- Can use name comparisons as a fundamental unit of work!

Order of Magnitude

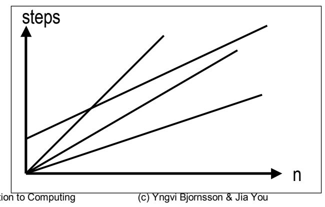
- We are:
  - Not interested in knowing the exact number of steps the algorithm performs.
  - Mainly interested in knowing *how the number of steps grows with increased input size!*
- Why?
  - Given large enough input, the algorithm with faster growth will execute more steps.
- Order of magnitude, O(...), measures how the number of steps grows with input size n.

Order of Magnitude

- Not interested in the exact number of steps, for example, algorithm where total steps are:
  - n
  - 5n
  - 5n+345
  - 4500n+1000
- are all of order O(n)
  - For all the above algorithms, the total number of steps grows approx. proportionally with input size (given large enough n).

Linear Algorithms - O(n)

- If the number of steps grows in proportion, or linearly, with input size, its a linear algorithm, O(n).
  - Sequential search is linear, denoted O(n)
- On a graph, will show as a straight line



Non-linear Algorithm

- Think of an algorithm for filling out the n-times multiplication table.

	1	...	n
1			
...			
n			

- As n increases the work the algorithm does will increase by n*n or n^2, the algorithm is $O(n^2)$

Data Cleanup - Analysis

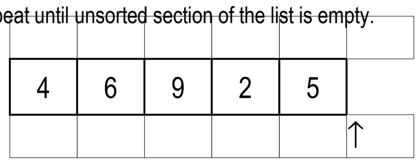
	Shuffle-Left		Copy-Over		Converging pointers	
	Time	Space	Time	Space	Time	Space
Best Case	O(n)	n	O(n)	n	O(n)	n
Worst Case	$O(n^2)$	n	O(n)	2n	O(n)	n
Average Case	$O(n^2)$	n	O(n)	$n \le x \le 2n$	O(n)	n

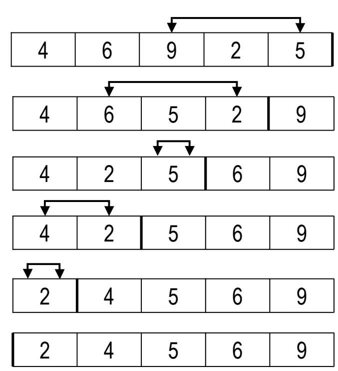
Sorting

- Sorting is a very common task, for example:
  - sorting a list of names into alphabetical order
  - numbers into numerical order
- Important to find efficient algorithms for sorting
  - Selection sort
  - Bubble sort
  - Quick sort
  - Heap sort
- We will analyze the complexity of selection sort.

Selection Sort

- Divide the list into a unsorted and a sorted section, initially the sorted section is empty.
- Locate the largest element in the unsorted section and replace that with the last element of the unsorted section.
- Move the marker between the unsorted and sorted section one position to the left.
- Repeat until unsorted section of the list is empty.





Selection Sort - Animation

- Exchange the largest element of the unsorted section with the last element of the unsorted section
- Move marker separating the unsorted and sorted section one position to the left (forward in the list)
- Continue until unsorted section is empty.

2	4	5	6	9	
↑					

Selection Sort - Analysis

- What order of magnitude is this algorithm?
  - Use number of comparisons as a fundamental unit of work.
- Total number of comparisons:

$$(n-1) + (n-2) + \ldots + 2 + 1$$
$$= (n-1) / 2 * n$$
$$= \tfrac{1}{2} n^2 - \tfrac{1}{2} n$$

- This is a $O(n^2)$ algorithm.
- Worst, best, average case behavior the same (why?)

Binary Search

- How do we look up words in a list that is already sorted?
  - Dictionary
  - Phone book
- Method:
  - Open up the book roughly in the middle.
  - Check in which half the word is.
  - Split that half again in two.
  - Continue until we find the word.

Binary Search - Example

	Ann	Bob	Dave	Garry	Nancy	Pat	Sue
Position:	1	2	3	4	5	6	7

To find Nancy, we go through

Garry (mid point at 4)
Pat (mid point of 5-7)
Nancy (mid point of a single item)

Binary Search - Odd number of elements

	Ann	Bob	Dave	Garry	Nancy	Pat	Sue
Position:	1	2	3	4	5	6	7

Whom that can be found

in one step: Garry
in two steps: Bob, Pat
in three steps: all remaining persons

Binary Search - Even number of elements

	Ann	Bob	Dave	Garry	Nancy	Pat
Position:	1	2	3	4	5	6

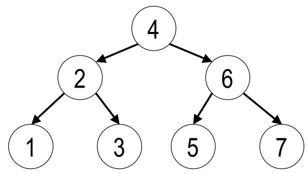
Let's choose the end of first half as midpoint.
Whom that can be found

in one step: Dave
in two steps: Ann, Nancy
in three steps: all remaining persons

Binary Search - Analysis

- Looking for a name is like walking branches in a tree

	Ann	Bob	Dave	Garry	Nancy	Pat	Sue
Position:	1	2	3	4	5	6	7

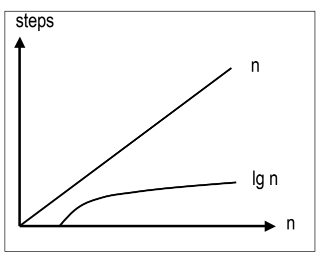


Binary Search - Analysis (cont.)

- We cut the number of remaining names in half.
- The number of times a number *n* can be cut if half and not get below 1 is called
  - Logarithm of n to the base 2
  - Notation: $\log_2 n$ or lg n
- Max. number of name comparisons = depth of tree.
  - 3 in the pervious example.
  - n names then approx. lg n comparisons needed
- Binary search is O(lg n)

Logarithm vs. Linear

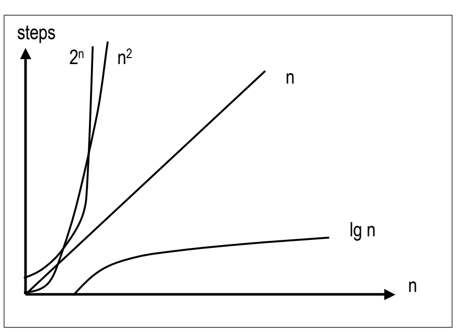
n	lg n
8	3
16	4
32	5
64	6
128	7
...	
32768	15
...	
1048576	20



When Things Get Out of Hand

- Polynomial algorithms (exponent is a constant)
  - For example: lg n, n, n^2, n^3, ... , n^{3000}, ...
  - More generally: n^a
- Exponential algorithms (exponent function of n)
  - For example: 2^n
  - More generally: a^n
- An exponential algorithm:
  - Given large enough n will always performs more work than a polynomially bounded one.
- Problem for which there exist only exponential algorithms are called intractable
  - Solvable, but not within practical time limits
  - Most often it is infeasible to solve but the smallest problems!

Growth Rate



Example of growth

N	10	50	100	1000
lg(n)	.0003 sec	.0006 sec	.0007 sec	.001 sec
n	.001 sec	.005 sec	.01 sec	0.1 sec
n^2	.01 sec	.25 sec	1 sec	1.67 min
2^n	.1024 sec	3570 years	$4*10^{16}$ centuries	Too big

Summary

- We are concerned with the efficiency of algorithms
  - Time- and Space-efficiency
  - Need to analyze the algorithms
- Order of magnitude measures the efficiency
  - E.g. O(lg n), O(n), $O(n^2)$, $O(n^3)$, $O(2^n)$, ...
  - Measures how fast the work grows as we increase the input size n.
  - Desirable to have slow growth rate.

Summary

- We looked at different algorithms
  - Data-Cleanup: Shuffle-left $O(n^2)$, Copy-over O(n), Converging-pointers O(n)
  - Search: Sequential-search O(n), Binary-search 0(lg n)
  - Sorting: Selection-sort $O(n^2)$
- Some algorithms are exponential
  - Not polynomially bounded
  - Problems for which there exists only exponential algorithms are called intractable
  - Only feasible to solve small instances of such problems