Complexity Workshop

Here is a small example of how to think about and work with algorithm complexity.

Data Structure Efficiencies

The elementary unit of analysis is "*memory accesses*", which may include storing an item (constructors), locating an item (recognizers), and retrieving an item (accessors).

Arrays:	`A[i]`	given index `i`, go straight to `A[i]`	`O[1]`
Trees:	`((a b)(c d))`	a series of branching decisions locates item `i`	`O[ln n]`
Lists:	`(a b c)`	look through all items for `i`	`O[n]`

The Memory Hierarchy (1999 technology)

Type	*typical access time*	*(^10 ns)*	*typical capacity (bytes)*	*(^2 bits)*
registers	2-5 ns	0	64-512	9-12
primary cache	4-10 ns	1	8K-256K	16-21
secondary cache	20-100 ns	2	512K-4M	22-25
main memory	50-1000 ns	3	8M-4G	26-35
disk	5-15 ms	7	500M-1T	32-41
tape	1-50 s	10	unlimited	

1 ns = 10^-9 s   1 ms = 10^-3 s

Pragmatics of Nested Loops

Nested loops effectively do a brute-force search over all items. Consider searching a three dimensional matrix, indexed by `(i,j,k)`:

```
for each i do
  for each j do
    for each k do
      <process>
```

If every item must be processed (e.g.: a pixel-based graphics display process), then the loops are unavoidable, and the best case is also the worst case.

However, if one item must be found out of n, then we can avoid some effort. The two effort avoidance techniques are

data structure organization and *smart, knowledgeable search*

When is Organization better than Knowledge?

Search is an exponential process. Ignoring effort other than direct search, and assuming the data can be structured as a binary decision process, then

`Search-effort = 2^n` n is number of items in search pool

How deeply can loops be nested to have the same efficiency as search? Ignoring polynomial coefficients:

`Loop-effort = n^k` k is the number of nested loops

Point of equal effort:

`2^n = n^k`

Solving for k:

`n/ln n = k`

Table of n/ln n:

ln n	*n*	*n/ln n*
1	2	1
2	4	2
3	8	2.7
4	16	4
5	32	6.4
6	64	10
7	128	18
8	256	32
9	512	57
10	1024	102
11	2048	186
12	4096	341

Conclusion: worst case search through even small sets is worse than many nested loops.

Pragmatics

In fact, worst case search requires every decision to be incorrect. This is very difficult to achieve, since it requires perfect anti-knowledge.

Let P be the fraction of times that a incorrect decision is made. For example, in a sorted binary tree, the correct decision is made every time, since the structure of sorting gives the needed contextual information. In the perfectly sorted case, with no search errors:

n decisions requires ln n steps, 2^n decisions requires n steps.

Conclusion: Sorting turns search into looping

In general, making P correct decisions:

$$2^{(nP)} = n^k$$

$$Pn/\ln n = k$$

Here is how knowledge effects search effort:

		all wrong	*half/half*	*1 in 10*	*1 in 100 wrong*
		P=1	***P=2^-1***	***P=2^-3***	***P=2^-7***
ln n	***n***	***n/ln n***			
1	2	1			
2	4	2	1		
3	8	2.7	1.4		
4	16	4	2		
5	32	6.4	3.2		
6	64	10	5	1.2	
7	128	18	9	2	
8	256	32	16	4	
9	512	57	29	7	
10	1024	102	51	13	
11	2048	186	93	23	1.5
12	4096	341	170	42	2.7

When we guess correctly half of the time, the cross-over point between search and brute-force looping is at about 30 items for k=3. As errors decrease to 1 in 100, the number of items increases to over 4000. Thus, partial knowledge about the location of an item greatly increases the number of items that can be searched before a search strategy becomes less desirable than brute-force looping.