CPSC 320

Intermediate Algorithm Design and Analysis

Welcome. Will Evans.

Quiz. 20 minutes. Anonymous.

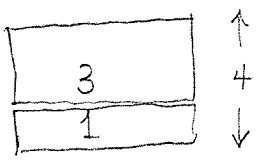
Learning Goals

1. Design Algs for computational problems
   techniques for thinking iteratively and recursively

2. Evaluate the performance of these algorithms.
   (e.g. running time)
   mathematical tools for handling recurrence relations and summations

3. Derive lower bounds on the time (or space or...) it takes to solve a problem.
   What is the best algorithm for the job?
   Is the job hard to solve quickly?

Puzzle Shims



Assume shims have integer thicknesses.

Optimization Problem

A shim is a piece of metal or plastic of a precise thickness.

Stack several shims to get other thicknesses.

Example: With shims 1mm, 3mm, 4mm you can create thicknesses

0, 1, 3, 4, 5, 7, 8 what about 6? 2?

What is the best set of 3 shims?

can create the most thicknesses with no gaps (like 6 and 2)

We can only hope to create $2^3 = 8$ thicknesses.

Why? – only 2^3 subsets

1, 2, 4 allows 8 thicknesses 0, 1, ..., 7

Why? – every subset represents a unique binary number

Best set of k shims: $2^0, 2^1, 2^2, \ldots, 2^{k-1}$

gives all thicknesses from 0 to $2^k - 1$

prove this by induction hint

Smallest number of shims to get all thicknesses from 0 to n?

Ans: $\lceil \log_2(n+1) \rceil$ shims: $2^0, 2^1, \ldots, 2^{\lceil \log_2(n+1) \rceil - 1}$

If $n = 8$ we need 4 shims: 1, 2, 4, 8

Seems wasteful. We can get 0, 1, 2, ..., 15 with these shims

but recall with 3 shims we can get only 8 different thicknesses. not 9.

In general, with $k \leq \lceil \log_2(n+1) \rceil - 1$ shims we get

$$2^k \leq 2^{\lceil \log_2(n+1) \rceil - 1} < 2^{(\log_2(n+1)+1)-1} = n+1$$

different thicknesses.

Lee Valley sells a set of 14 shims

20×4 10×4 2×4 5×2

that get all thicknesses from 0 to 138 except 1, 3, 135, 137

With 14 shims you could get all thicknesses up to

$2^{14} - 1 = 16,383$

Readings

CLRS

Chapters 1 & 2.

Algorithms

- Take input. Examples: k (the number of shims)
  $a_1, a_2 \ldots a_n$ (a sequence of n numbers)
- Produce Output: Examples: k thicknesses of the best k shims
  a sequence $a_1' a_2' \ldots a_n'$ that is a permutation of the input sequence such that $a_1' \le a_2' \le \ldots \le a_n'$
- Consist of a sequence of computational steps

Example Insertion Sort

Input $a_1, a_2 \ldots a_n$

(How to be a computer) → {Sequence of steps view versus {Sequence of states or snapshots view

Idea

(How to design an algorithm.)

If we can sort $a_1 a_2 \ldots a_{j-1}$ into an array B then we can sort $a_1 a_2 \ldots a_j$ by inserting a_j into its proper spot in B.

How? ← Binary search ✗, SkipList ✓, Self balancing search tree ✓, Priority Queue ✓

Insertion Sort $(a_1, a_2 \ldots a_n)$

```
B[1] = a_1
for j = 2 to n {
    key = a_j
    i = j-1
    while i>0 and B[i] > key
        B[i+1] = B[i]
        i = i-1
    B[i+1] = key
return B
```

(insert)

Loop invariant (or j^{th} snapshot)

Before j^{th} iteration of for-loop: $B[1 \ldots j-1]$ contains $a_1, a_2 \ldots a_{j-1}$ in sorted order

Running Time

$$T(n) \le 1 + \sum_{j=2}^{n} (1+1+(j-1)2+1)$$

$$= 1 + \sum_{j=2}^{n} (2j+1)$$

$$= 1 + n-1 + 2\left(\frac{n(n+1)}{2} - 1\right) \in O(n^2)$$

Loop invariant (if true) implies the algorithm is correct. (Before $n+1^{st}$ iteration $B[1...n]$ is sorted input.)

Base case ($j=1$) loop invariant is true

If $(j-1)^{st}$ loop invariant is true then j^{th} loop invariant is true. Why?

Design philosophy of Insertion Sort ≡ Solve partial input (consume input)

Design philosophy of Selection Sort ≡ Produce partial output (produce output)

Selection Sort

Suppose $a_1' \le a_2' \le \dots \le a_n'$ is the correct output.

Idea If we can remove from the input (and output)

$a_1', a_2', a_3', \dots a_{j-1}'$

Then we can remove (and output) a_j'

a_j' is the minimum of what remains

```
Selection Sort (A[1...n])
    for j = 1 to n
        mini = j
        for i = j+1 to n
            if A[i] < A[mini] then mini = i
        swap(A[j], A[mini])
        print A[j]
```

Loop invariant

Before j^{th} iteration of for loop first $j-1$ numbers of output have been printed in order and $A[j...n]$ contains remaining input.

Running time

$$\sum_{j=1}^{n} \left(3 + \sum_{i=j+1}^{n} (1)\right) = \sum_{j=1}^{n} (3 + (n-j)) = 3n + n^2 - \sum_{j=1}^{n} j$$

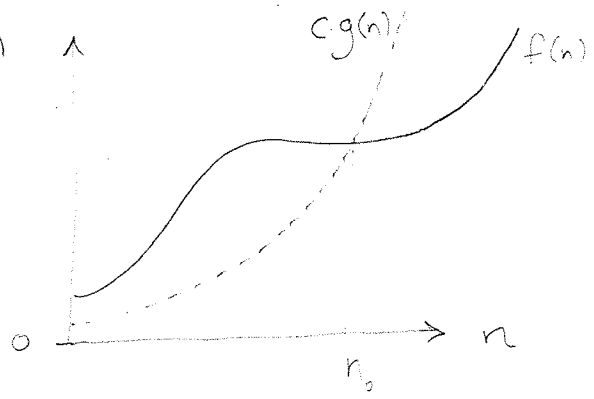
$$= 3n + n^2 - \frac{n(n+1)}{2}$$

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

$$\in O(n^2)$$

Big O Notation (refresher)

$$O(g(n)) = \{ f(n) : \exists \text{ positive } c, n_0 \text{ such that } 0 \le f(n) \le c \cdot g(n) \text{ for all } n \ge n_0 \}$$



f(n) and g(n) are positive functions

$f(n) \in O(g(n))$ means $g(n)$ is an asymptotic upper bound on $f(n)$.

$$\underbrace{\frac{n^2}{2} + \frac{5n}{2}}_{f(n)} \le \frac{n^2}{2} + \frac{5n^2}{2} = \underbrace{3n^2}_{g(n)} \qquad c = 3, \; n_0 = 1$$

How do you show $f(n) \notin O(g(n))$? e.g. $\frac{n^2}{2} + \frac{5n}{2} \notin O(n)$

Use proof by contradiction e.g.

Suppose $\frac{n^2}{2} + \frac{5n}{2} \in O(n)$

Then $\exists c, n_0$ s.t. $\frac{n^2}{2} + \frac{5n}{2} \le c \cdot n$ for all $n > n_0$

(Pick such a c and n_0)

$$\Leftrightarrow \quad \frac{n}{2} + \frac{5}{2} \le c \quad \text{for all } n > n_0$$

$$\Leftrightarrow \quad n \le 2c - 5 \quad \text{for all } n > n_0$$

But this can't be true for all $n > n_0$

Suppose we pick $n > \max\{ n_0, 2c-5 \}$.

Now $n > n_0$ so $n \le 2c-5$ but n is also $> 2c-5$

$\Rightarrow\Leftarrow$