Algorithms

R4 Cheng

January 30, 2025

Def. Introduction of precise, umambiguous, and correct procedures for solving general problems

We care how many clicks in debug mode

Runtime is expressed by counting the number of steps, as funtion of **the size of the input**

Asymptotic Notations:

- Big O: upper bound on the growth rate
- Little o: "strict" upper bound on the growth rate
- Big Ω: lower bound on the growth rate
- Θ: tight bound on the growth rate

$f(n)$: the runtime of an algorithm for input size n

$g(n)$: a simplified function without constants, lower order terms, e.g. n^2, $n \log n$

Big-O Definition: It is said $f(n) = O(g(n)) \iff$ there exists a constant $c > 0$ and $n_0 > 0$ such that

$$f(n) \leq c \cdot g(n) \text{ for all } n \geq n_0$$

Little-o Definition:

$$f(n) = o(g(n)) \iff \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

Big-Ω Definition: It is said $f(n) = \Omega(g(n)) \iff$ there exists a constant $c > 0$ and $n_0 > 0$ such that

$$f(n) \geq c \cdot g(n) \text{ for all } n \geq n_0$$

Caveats about constants

For $\log_4(n)$, is 4 ignorable? No, it matters

Remark. $log_a(x) = log_a blog_b(x)$

Common Efficiency Class

Class	Name	
1	constant	No reasonable examples, most cases infinite input size requires infinite run time
$\log n$	Logarithmic	Each operation reduces the problem size by half, Must not look at the whole input, or a fraction of the input, otherwise will be linear
n	linear	Algorithms that scans a list of n items (sequential search)
$n \log n$	linearithmic	Many D&C algorithms e.g., merge sort
n^2	quadratic	Double embedded loops, insertion sort
n^3	cubic	Three embedded loops, some linear algebra algo.
2^n	exponential	Typical for algo that generates all subsets of a n element set.
$n!$	factorial	Typical for algo that generates all permutations of a n-element set

Merge Sort

$$T(n) = 2T(\frac{n}{2}) + 2n + c$$

Remark. $2n$: *compare and copy n elements* $= 2n$

Proof

$$T(n) = 2T(\frac{n}{2}) + 2n + c$$
$$\Rightarrow T(\frac{n}{2}) = 2T(\frac{n}{4}) + (n + c)$$
$$\Rightarrow T(\frac{n}{4}) = 2T(\frac{n}{8}) + (\frac{n}{2} + c)$$

Bring in $T(\frac{n}{2})$

$$\Rightarrow T(n) = 2\{2T(\frac{n}{4}) + (n + c)\} + 2n + c$$
$$= 2^2 T(\frac{n}{2^2}) + 2 \cdot 2n + c + 2c$$

Bring in $T(\frac{n}{4})$

$$= 2^2\{2T(\frac{n}{8}) + (\frac{n}{2} + c)\} + 2 \cdot 2n + c + 2c$$

$$= 2^3 T(\frac{n}{2^3}) + 3 \cdot 2n + c + 2c + 4c$$

$$\Rightarrow T(n) = 2^k T(\frac{n}{2^k}) + k \cdot 2n + (2^k - 1)c$$

Since $T(1) = 1$, $\frac{n}{2^k} = 1 \Rightarrow k = \log_2(n)$

$$\Rightarrow T(n) = n \cdot 1 + \log_2(n) \cdot 2n + (n-1)c$$

$$\Rightarrow T(n) = O(nlogn)$$

Master Theorem

$$T(n) = aT(\frac{n}{b}) + cn^d$$

$$\Rightarrow T(n) = (1 + \frac{a}{b^d}^2 + \cdots + \frac{a}{b^d}^k)cn^d$$

a: number of subproblems

b: factor by which the problem size is reduced

d: exponent in the running time of the "combine" step

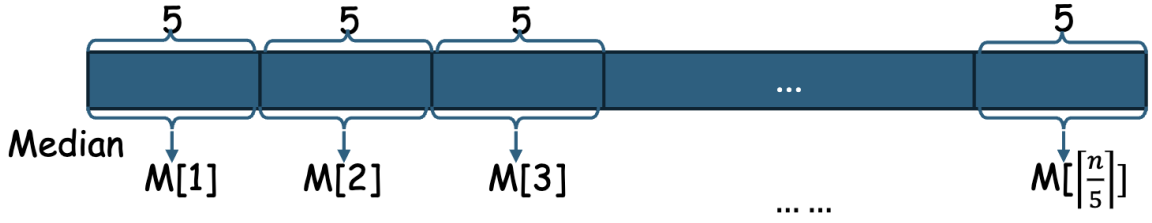
Case 1: $a < b^d$, or equiv. $d > log_b a, T(n) = O(n^d) \Rightarrow$ the cost of the "combine" step dominates the cost of the "split" step

Case 2: $a = b^d$, or equiv. $d = log_b a, T(n) = O(n^d \log n)$

Case 3: $a > b^d$, or equiv. $d < log_b a, T(n) = O(n^{log_b a})$

Smart Selection of Pivot

Break array A into chunks of fives and find the median of each chunk, constructing a new array M of medians.



And **median** of M aka. MM is the pivot.

Claim: MM is at least $\geq \frac{3}{10}A$ and $\leq \frac{3}{10}A$

Proof: MM is the median of medians, so $\frac{1}{2}$ medians in M are $\leq MM$. $\because M = \frac{A}{5} \Rightarrow \frac{A}{10}$ medians are $\leq MM$. Moreover, For each $M[i]$, there are 3 elements $\leq M[i] \therefore \frac{3A}{10} \leq MM$

```
Select(A, k)
1. for i = 1, ..., n/5
2.     M[i] = median (a[5(i-1)+1], 5i)
3. MM = Select(M, n/10)
4. v = index of MM in A
5. r = partition(A, v)
6. if k=r: return A[r]
7. if k<r: Select(A[1,...,r-1], k)
8. if k>r: Select(A[r+1,...,n], k-r)
```

$$T(n) \leq O(n) + T(n/5) + T(7n/10) \Rightarrow T(n) = O(n)$$

Dynamic Programming

Fibonacci

$$\begin{aligned} T(n) &= T(n-1) + T(n-2) + c \\ \Rightarrow T(n) &\leq 2T(n-1) + c \\ \Rightarrow T(n) &\leq 2^2 T(n-2) + 2c + c \\ \Rightarrow T(n) &\leq 2^k T(n-k) + kc \\ k = n-1 &\Rightarrow T(n) \leq O(2^n) \end{aligned}$$

$$T(n) \geq 2T(n-2) + c = \Omega(2^{\frac{n}{2}})$$

Applications

- Unix diff for comparing files
- Bellman-Ford for shortest path
- CKY algorithm for natural language parsing
- etc.

Edit Distance

	E	X	E	C	U	T	I	O	N	
N										
O										
I										
T										
N										
E										
T										
N										
I										

Knapsack

0-1 knapsack problem

Intuition: Go greedy

1. Greedily choose the most valuable item?
2. Greedily choose the most value/weight item?

Sadly, both can be found a counterexample.

Define $V(n, W)$ = the optimal value achievable using item $1, 2, \ldots, n$ with weight limit W

$\Rightarrow$

Subproblems:

1. Choose the current item: $V(n-1, W-W_n) + v_n$
2. Don't choose the current item: $V(n-1, W)$

$$\Rightarrow V(n, W) = \max \begin{cases} V(n-1, W-W_n) + v_n \\ V(n-1, W) \end{cases}$$

Unbounded knapsack problem

$$V(W) = \max(V(W - W_n) + v_n)$$