designing poly-time algorithms

example problem: max subarray

given array of small integers $a[1,\ldots,n]$, compute

$$\max_{i\le j}\sum_{k=i}^{j} a[k]$$

e.g. $\textsc{MaxSubarray}([31,-41,\mathbf{59},\mathbf{26},-\mathbf{53},\mathbf{58},\mathbf{97},-93,-23,84]) = 187$

algorithmic design techniques

1. enumeration
2. iteration
3. simplification & delegation *(aka divide & conquer)*
4. recursion inversion *(aka dynamic programming)*

enumeration for max subarray

evaluate every possible solution

```
MaxSubarray(a[1,...,n])
    for each pair (i,j) with 1 ≤ i < j ≤ n
        compute a[i]+a[i+1]+···+a[j-1]+a[j]
        keep max sum found so far
    return max sum found
```

analysis $(O(n^2)$ pairs$) \times (O(n)$ time to compute each sum$) = O(n^3)$ time

iteration for max subarray

don't compute sums from scratch:

$\sum_{k=i}^{j} a[k]$ can be computed from $\sum_{k=i}^{j-1} a[k]$ in $O(1)$ time

(really just clever enumeration)

MaxSubarray(a[1,...,n])
```
    for i = 1, ..., n
        sum = 0
        for j = i, ..., n
            sum = sum + a[j]
            keep max sum found so far
    return max sum found
```

analysis $(O(n)$ i-iterations$) \times (O(n)$ j-iterations$) \times (O(1)$ time to update sum$) = O(n^2)$

simplification & delegation for max subarray

max subarray either has value

- $\textsc{MaxSubarray}(a[1, \ldots, \frac{n}{2}])$,
- or $\textsc{MaxSubarray}(a[\frac{n}{2}, \ldots, n])$,
- or $\textsc{MaxSuffix}(a[1, \ldots, \frac{n}{2}])$+$\textsc{MaxPrefix}(a[\frac{n}{2}, \ldots, n])$

compute $\textsc{MaxSuffix}$ and $\textsc{MaxPrefix}$ in linear time by modifying previous algorithm

divide & conquer

$$\textsc{MaxSubarray}(a[1, \ldots, n]) = \max \begin{cases} \textsc{MaxSubarray}(a[1, \ldots, \frac{n}{2}]) \\ \textsc{MaxSubarray}(a[\frac{n}{2}, \ldots, n]) \\ \textsc{MaxSuffix}(a[1, \ldots, \frac{n}{2}]) + \textsc{MaxPrefix}(a[\frac{n}{2}, \ldots, n]) \end{cases}$$

analysis $(O(n)$ time for non-recursive work$) \times (O(\log n)$ depth$) = O(n \log n)$

recursion inversion for max subarray

the max subarray either uses the last element or doesn't:

$$\textsc{MaxSubarray}(a[1,\ldots,n]) = \max\begin{cases}\textsc{MaxSubarray}(a[1,\ldots,n-1])\\ \textsc{MaxSuffix}(a[1,\ldots,n]\end{cases},$$

$$\textsc{MaxSuffix}(a[1,\ldots,n]) = \max\{0, \textsc{MaxSuffix}(a[1,\ldots,n-1]) + a[n]\}$$

dynamic programming evaluate this non-recursively by computing

- first $\textsc{MaxSubarray}(a[1])$ and $\textsc{MaxSuffix}(a[1])$
- then $\textsc{MaxSubarray}(a[1,2])$ and $\textsc{MaxSuffix}(a[1,2])$ from above
- then $\textsc{MaxSubarray}(a[1,2,3])$ and $\textsc{MaxSuffix}(a[1,2,3])$ from above
- and so on

analysis computing $\textsc{MaxSubarray}(a[1,\ldots,n])$ and $\textsc{MaxSuffix}(a[1,\ldots,n]$
from $\textsc{MaxSubarray}(a[1,\ldots,n-1])$ and $\textsc{MaxSuffix}(a[1,\ldots,n-1])$
takes $O(1)$ time
$O(n)$ things to compute $= O(n)$ time

does algorithm design matter?

TABLE I. Summary of the Algorithms

Algorithm		1	2	3	4
Lines of C Code		8	7	14	7
Run time in microseconds		$3.4N^3$	$13N^2$	$46N \log N$	$33N$
Time to solve problem of size	10^2	3.4 secs	130 msecs	30 msecs	3.3 msecs
	10^3	.94 hrs	13 secs	.45 secs	33 msecs
	10^4	39 days	22 mins	6.1 secs	.33 secs
	10^5	108 yrs	1.5 days	1.3 min	3.3 secs
	10^6	108 mill	5 mos	15 min	33 secs
Max problem solved in one	sec	67	280	2000	30,000
	min	260	2200	82,000	2,000,000
	hr	1000	17,000	3,500,000	120,000,000
	day	3000	81,000	73,000,000	2,800,000,000

Digital Equipment Corporation VAX-11/750 in 1984