Algorithm Design Techniques

An Example
- The Problem
- Algorithm 1: Cubic Time
- Algorithm 2: Quadratic Time
- Algorithm 3: $O(n \log n)$ Time
- Algorithm 4: Linear Time
- Comparison of Algorithms

Principles

The Problem

Definition. Given the real vector $x[n]$, compute the maximum sum found in any contiguous subvector.

An Example. If the input vector is

31	-41	59	26	-53	58	97	-93	-23	84
		↑ 2				↑ 6			

then the program returns the sum of $x[2..6]$, or 187.

A Cubic Algorithm

Idea. For all pairs of integers i and j satisfying $0 \le i \le j < n$, check whether the sum of $x[i..j]$ is greater than the maximum sum so far.

Code.

```
maxsofar = 0
for i = [0, n)
    for j = [i, n)
        sum = 0
        for k = [i, j]
            sum += x[k]
        /* sum is sum of x[i..j] */
        maxsofar = max(maxsofar, sum)
```

Run Time. $O(n^3)$.

A Quadratic Algorithm

Idea. The sum of $x[i..j]$ is close to the previous sum, $x[i..j-1]$.

Code.

```
maxsofar = 0
for i = [0, n)
    sum = 0
    for j = [i, n)
        sum += x[j]
        /* sum is sum of x[i..j] */
        maxsofar = max(maxsofar, sum)
```

Run Time. $O(n^2)$.

Other Quadratic Algorithms?

Another Quadratic Algorithm

Idea. A ‘‘cumulative array’’ allows sums to be computed quickly. If $ytd[i]$ contains year-to-date sales through month i, then sales from March through September are given by $ytd[sep] - ytd[feb]$.

Implementation. Use the cumulative array *cumarr*. Initialize $cumarr[i] = x[0] + \cdots + x[i]$. The sum of the values in $x[i..j]$ is $cumarr[j] - cumarr[i-1]$.

Code for Algorithm 2b.

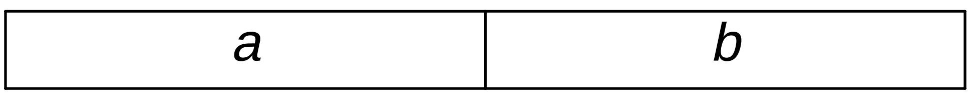
```
cumarr[-1] = 0
for i = [0, n)
    cumarr[i] = cumarr[i-1] + x[i]
maxsofar = 0
for i = [0, n)
    for j = [i, n)
        sum = cumarr[j] - cumarr[i-1]
        /* sum is sum of x[i..j] */
        maxsofar = max(maxsofar, sum)
```

Run Time. $O(n^2)$.

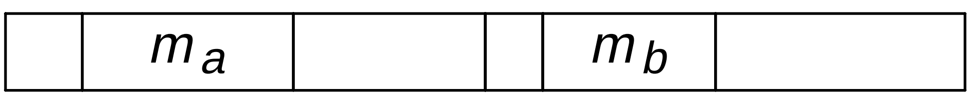
An $O(n \log n)$ Algorithm

The Divide-and-Conquer Schema. To solve a problem of size n, recursively solve two subproblems of size $n/2$ and combine their solutions.

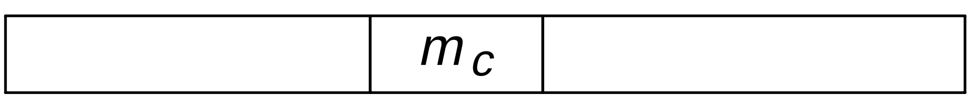
The Idea. Divide into two subproblems.



Recursively find maximum in subvectors.



Find maximum crossing subvector.



Return max of m_a, m_b and m_c.

Run Time. $O(n \log n)$.

Code for the $O(N \log N)$ Algorithm

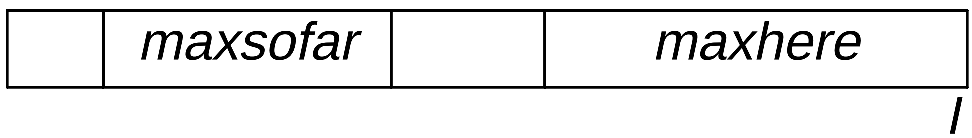
```
float maxsum3(l, u)
    if (l > u)  /* zero elements */
        return 0
    if (l == u)  /* one element */
        return max(0, x[l])

    m = (l + u) / 2
    /* find max crossing to left */
    lmax = sum = 0
    for (i = m; i >= l; i--)
        sum += x[i]
        lmax = max(lmax, sum)
    /* find max crossing to right */
    rmax = sum = 0
    for i = (m, u]
        sum += x[i]
        rmax = max(rmax, sum)

    return max(lmax+rmax,
               maxsum3(l, m),
               maxsum3(m+1, u))
```

A Linear Algorithm

Idea. How can we extend a solution for $x[0..i-1]$ into a solution for $x[0..i]$? Key variables:



Code.

```
maxsofar = 0
maxhere = 0
for i = [0, n)
    /* invariant: maxhere and maxsofar
       are accurate for x[0..i-1] */
    maxhere = max(maxhere + x[i], 0)
    maxsofar = max(maxsofar, maxhere)
```

Run Time. $O(n)$.

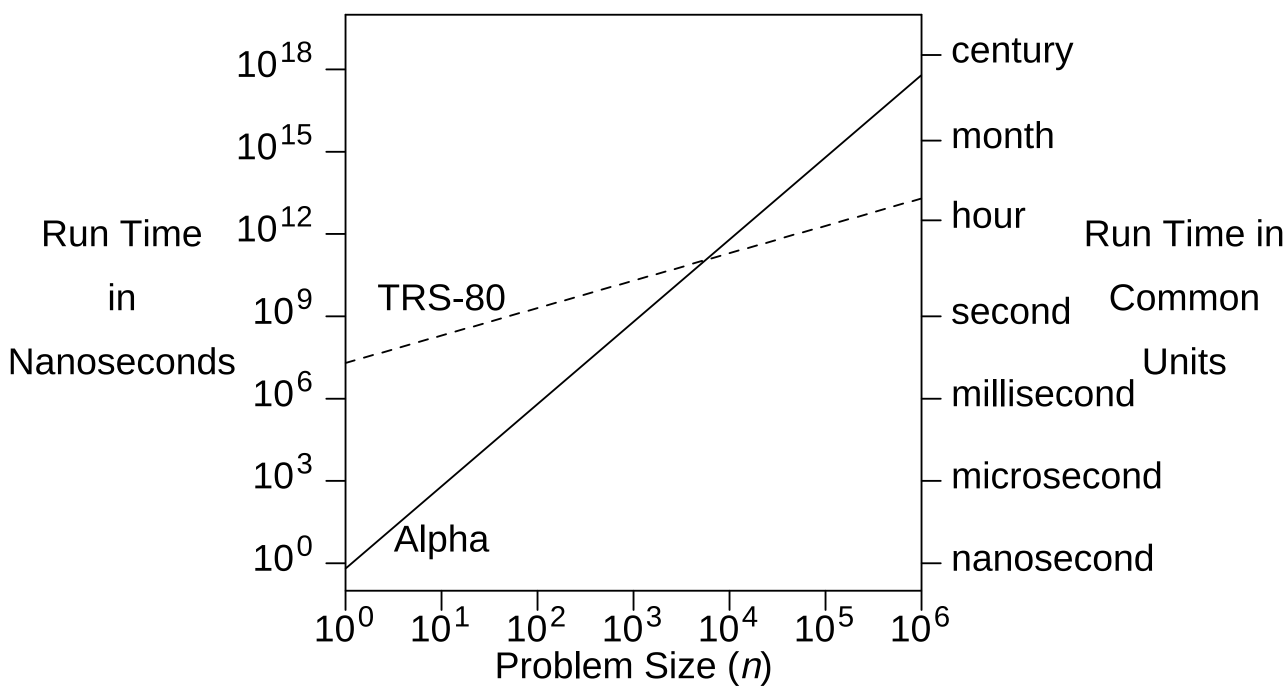
Summary of the Algorithms

ALGORITHM		**1**	**2**	**3**	**4**
Run time in nanoseconds		$1.3n^3$	$10n^2$	$47n \log_2 n$	$48n$
Time to solve a problem of size	10^3	1.3 secs	10 msecs	.4 msecs	.05 msecs
	10^4	22 mins	1 sec	6 msecs	.5 msecs
	10^5	15 days	1.7 min	78 msecs	5 msecs
	10^6	41 yrs	2.8 hrs	.94 secs	48 msecs
	10^7	41 millenia	1.7 wks	11 secs	.48 secs
Max size problem solved in one	sec	920	10,000	1.0×10^6	2.1×10^7
	min	3600	77,000	4.9×10^7	1.3×10^9
	hr	14,000	6.0×10^5	2.4×10^9	7.6×10^{10}
	day	41,000	2.9×10^6	5.0×10^{10}	1.8×10^{12}
If n multiplies by 10, time multiplies by		1000	100	10+	10
If time multiplies by 10, n multiplies by		2.15	3.16	10–	10

An Extreme Comparison

Algorithm 1 at 533MHz is $0.58n^3$ nanoseconds. Algorithm 4 interpreted at 2.03MHz is $19.5n$ milliseconds, or $19,500,000n$ nanoseconds.

n	1999 ALPHA 21164A, C, CUBIC ALGORITHM	1980 TRS-80, BASIC, LINEAR ALGORITHM
10	0.6 microsecs	200 millisecs
100	0.6 millisecs	2.0 secs
1000	0.6 secs	20 secs
10,000	10 mins	3.2 mins
100,000	7 days	32 mins
1,000,000	19 yrs	5.4 hrs



Design Techniques

Save state to avoid recomputation.

Algorithms 2 and 4.

Preprocess information into data structures.

Algorithm 2b.

Divide-and-conquer algorithms.

Algorithm 3.

Scanning algorithms.

Algorithm 4.

Cumulatives.

Algorithm 2b.

Lower bounds.

Algorithm 4.