Fall 2020 ECE20017

Algorithm Analysis and Design

September 2020

Joseph S. Shin
(formerly Sung Yong Shin)

Algorithm Analysis

How to compare algorithms ?

Given algorithms that solve a problem, how do we compare these algorithms ?

The growth rate of the time or space requirements to solve a (arbitrarily) **large** instance of the problem.

Asymptotic Growth rate

The size of the problem

problem size (or input size): the amount of data given as the input

Examples:

sorting : the length of the list containing the data

graph coloring : |V| + |E| for G = (V, E)

matrix multiplication : the number of elements in the matrices

Running time

# of primitive operations (steps) executed

How to compare algorithms ?

Given an algorithms to solve a problem, how do we compare algorithms ?

The rate of growth of the time or space requirements to solve a large instance of the problem.

Asymtotic Growth rate

The size of the problem

Input size: the amount of data given as the input

Example

sorting : the length of the list containing the data: n

graph coloring : $|V| + |E|$ for $G = (V, E)$

matrix multiplication : the number of elements in the matrices

Running time

# of primitive operations (steps) executed

Time Complexity

The number of time steps for an algorithm to solve a problem
Expressed as a function of problem size.

Space Complexity

The amount of memory space for an algorithm to solve a problem
Expressed as a function of problem size.

Worst case complexity

The worst possible complexity over all inputs of a given problem size.

Average case complexity

The average complexity over all inputs of a given problem size

(Tight) Lower bound L(n)

The minimum time complexity over **all possible** algorithms

(Tight) Upper bound U(n)

The minimum time complexity over **all known** algorithms

Optimal algorithm

An algorithm whose time complexity is the same as the lower bound of the problem, L(n)

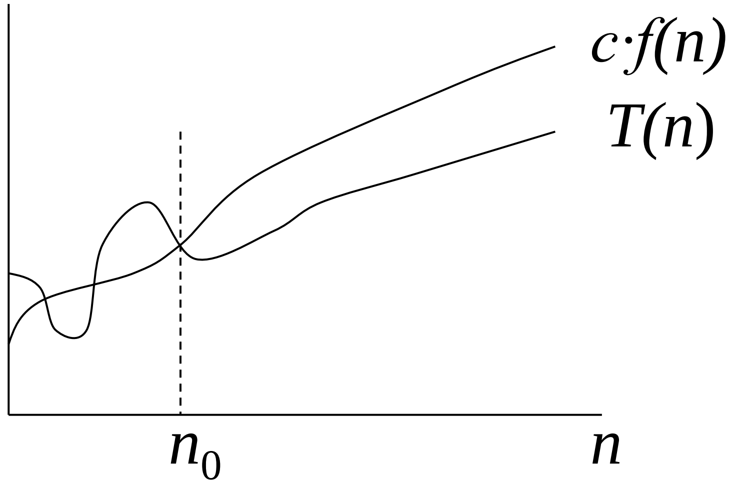
Measuring efficiency of algorithms

time complexity : T(n)

space complexity : S(n)

"Big-Oh" notation: e.g., $T(n) = O(f(n))$

$T(n)$ is $O(f(n))$ ⇔ There exist positive constants c and n_0 such that $T(n) \le c \cdot f(n)$ for all $n \ge n_0$



Let $T(n) = n^2/2 + 3n + 10$

Then $T(n) = O(n^2)$! Why?

$T(n) \leq c \cdot n^2$ for all $n \geq 1$ and $c = 27/2$!!!

Is $T(n)$ also $O(n^3)$?

Is $T(n)$ also $O(n)$?

Example: Bubble Sort

Given a set of n real numbers, sort them in the ascending order.

```
1.  for i:=1 to n-1 do
2.     for j:=n down to i+1 do
3.        if A[j-1] > A[j] then
             { swap A[j-1] and A[j]}
4.              temp := A[j-1];
5.              A[j-1] := A[j];
6.              A[j] := temp;
7.        end {if}
8.     end{for}
9.  end{for}
```

# of passes

# of basic OP

worst case

1. n
2. n (n + 1) / 2 - 1
3. **(n - 1) n / 2**
4. (n - 1) n / 2
5. (n - 1) n / 2
6. (n - 1) n / 2
7. (n - 1) n / 2
8. (n +1) n / 2 - 2
9. n - 1

$i = 1$: n
$i = 2$: $n - 1$
....
$i = n - 1$: 2
$i = n$: 0

$i = 1$: $n - 1$
$i = 2$: $n - 2$
....
$i = n - 1$: 1
$i = n$: 0

$$n (n + 1) - 3 + 5(n - 1) n/2 + 2n - 1$$
$$= \mathbf{7/2\, n^2 + n / 2 - 4}$$

⇩

$$\mathbf{O(n^2)}$$

Algorithm Design

The Maximum Subsequence Sum Problem

Given a sequence of n numbers, $X=(x_1, x_2, \ldots, x_n)$, find a subsequence $X^* \subseteq X$ such that

(i) the numbers in X^* is contiguous in X

(ii) the sum of the numbers in X^* is the maximum over all contiguous subsequences of X

X =	(31,	-41,	59,	26,	-53,	58,	97,	-93,	-23,	84)
	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}

187

3 7

$X^*=X[3..7]$

Observation

$$X = (x_1, x_2, x_3, \dots , x_n)$$

What if $x_i > 0$ for all $1 \le i \le n$?

What if $x_i < 0$ for all $1 \le i \le n$?

Brute Force Algorithm

How many subsequences?

X[L..U]

L	U	
1	1, 2, ⋮, n	n
2	2, 3, ⋮, n	n-1
3, ⋮, ⋮	3, 4, ⋮, n	n-2
⋮	⋮	⋮
n	n	1
		$\frac{n(n+1)}{2}$

For each subsequence, we need at most n-1 additions

why?

$\therefore$ O(n)

Then, the total number of operations is at most

$$\left(n(n+1)/2 \right) \times (n-1)$$

$\therefore$ O(n^3)

Procedure Max-sub

```
begin
        MaxSoFar ← 0
        for L ← 1 to n do
                for U ← L to n do
                        Sum ← 0
                        for i ← L to U do
                                Sum ← Sum+x[i]
                        end-do
                        MaxSoFar ← max { MaxSoFar, Sum}
                end-do
        end-do
end
```

Do we need this?

$O(n^3)$

$O(n^2)$ Algorithm

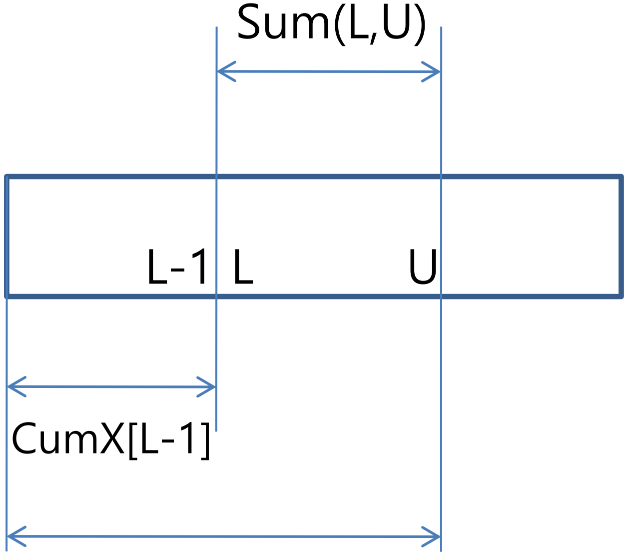
Sum of X[L..U] = Sum[L, U] $1 \le L \le n,\ L \le U \le n$

$$\text{Sum}(L,U) = \begin{cases} X[U] & \text{if } L = U \\ \text{Sum}(L,U-1)+X[U] & \text{otherwise} \end{cases}$$

```
Procedure Max-sub
    begin
        MaxSoFar ← 0
        for L ← 1 to n do
            Sum ← 0
            for U ← L to n do
                Sum ← Sum+X[U]
                MaxSoFar ← max { MaxSoFar, Sum}
            end-do
        end-do
    end
```

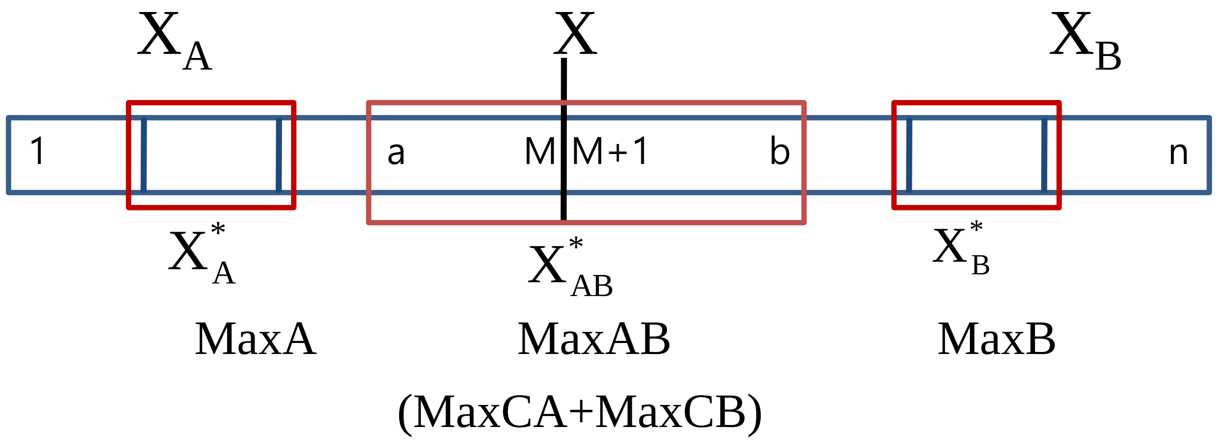
$O(n^2)$

A Yet Another O(n^2) Algorithm




```
Procedure Max-sub
    begin
        CumX[0] ← 0
        for i ← 1 to n do
            CumX[ i ] ← CumX[i-1]+X[ i ]      Preprocessing
        end-do
        MaxSoFar ← 0
        for L ← 1 to n do
            for U ← L to n do
                Sum ← CumX[U]-CumX[L-1]
                MaxSoFar ← max { MaxSoFar, Sum}
            end-do
        end-do
    end
```

"Divide and Conquer" Algorithm



$M = \lfloor (n+1)/2 \rfloor$

$$\text{MaxX} = \max\{ \underset{X_A^*}{\text{MaxA}}, \underset{X_B^*}{\text{MaxB}}, \underset{X_{AB}^*}{\text{MaxAB}} \}$$

$T(n) = 2T(n/2) + M(n)$

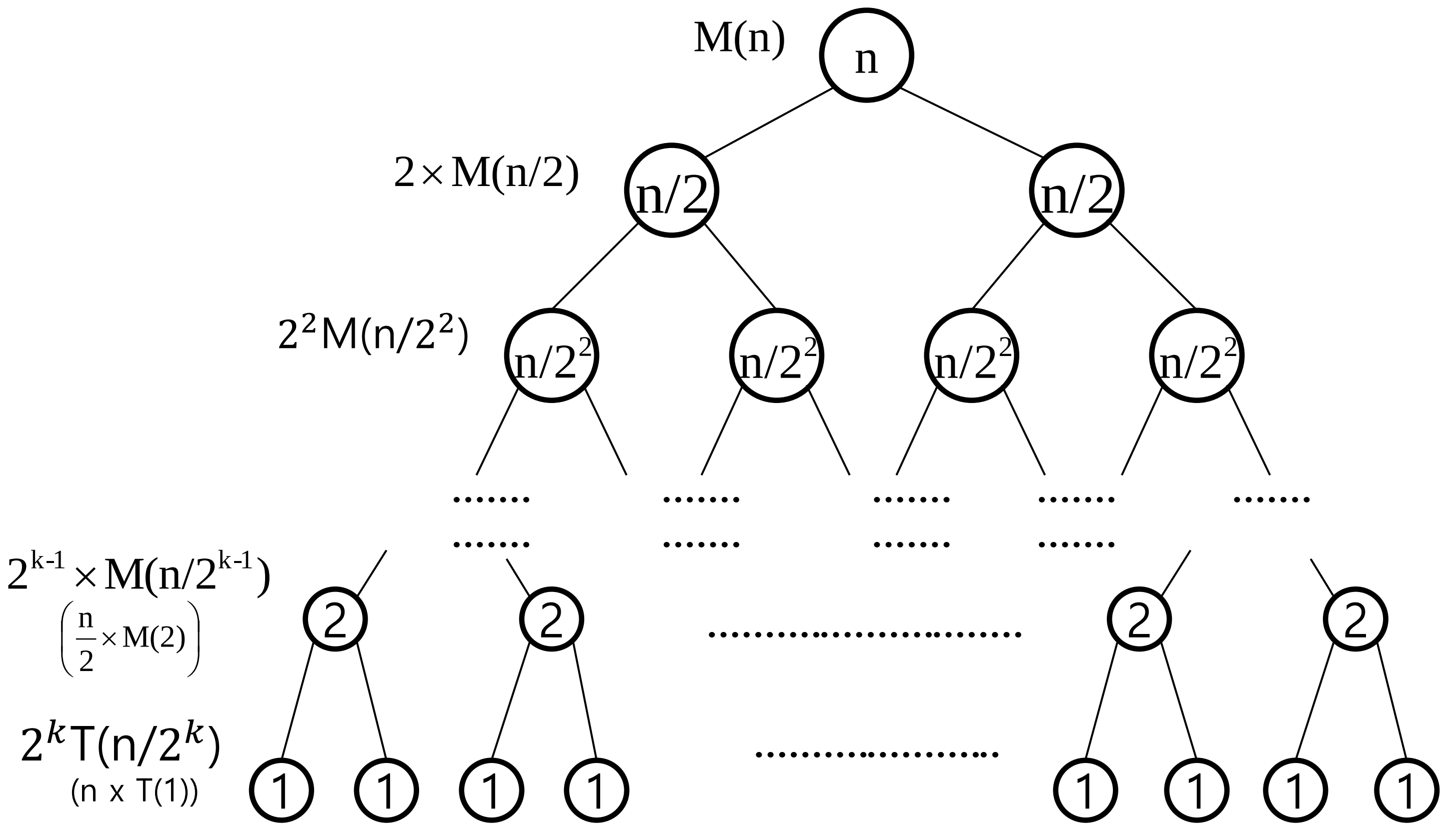
Compute MaxA: $T(n/2)$

Compute MaxB: $T(n/2)$

$\boxed{\text{Divide and Merge}}$: $M(n)$

Need to compute MaxAB

$$T(n)=2T(n/2)+M(n)$$



$\boxed{n = 2^k}$

How many levels? $\log_2 n$

Why?

$$n/2^k = 1 \quad \therefore\ k = \log_2 n$$

What is T(1)?

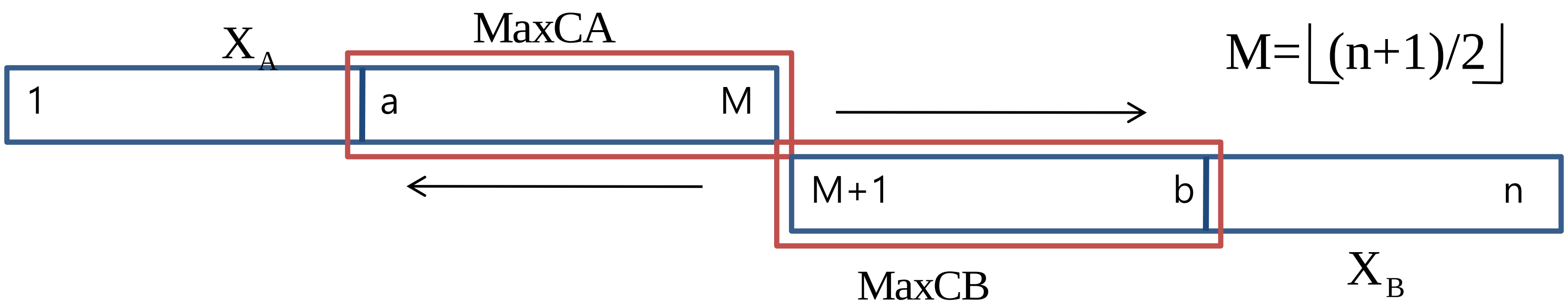
In this case, no further division since L=U!

$$\therefore\ \mathrm{Max}(X[L..U]) = \begin{cases} X[L] \text{ if } X[L] > 0 \ (X^* = X[L..L]) \\ 0 \quad \text{ if } X[L] \leq 0 \ (X^* = \phi) \end{cases}$$

What is M(n) if n > 1?

It depends on how to compute MaxAB!

How to Compute MaxAB



$M = \lfloor (n+1)/2 \rfloor$

MaxAB = MaxCA + MaxCB

where

MaxCA = max{ Sum(1, M), Sum(2, M), ……………………, Sum(M-1, M), Sum(M, M) }

MaxCB = max{ Sum(M+1, M+1), Sum(M+1, M+2), ……, Sum(M+1, n-1), Sum(M+1, n) }

Note

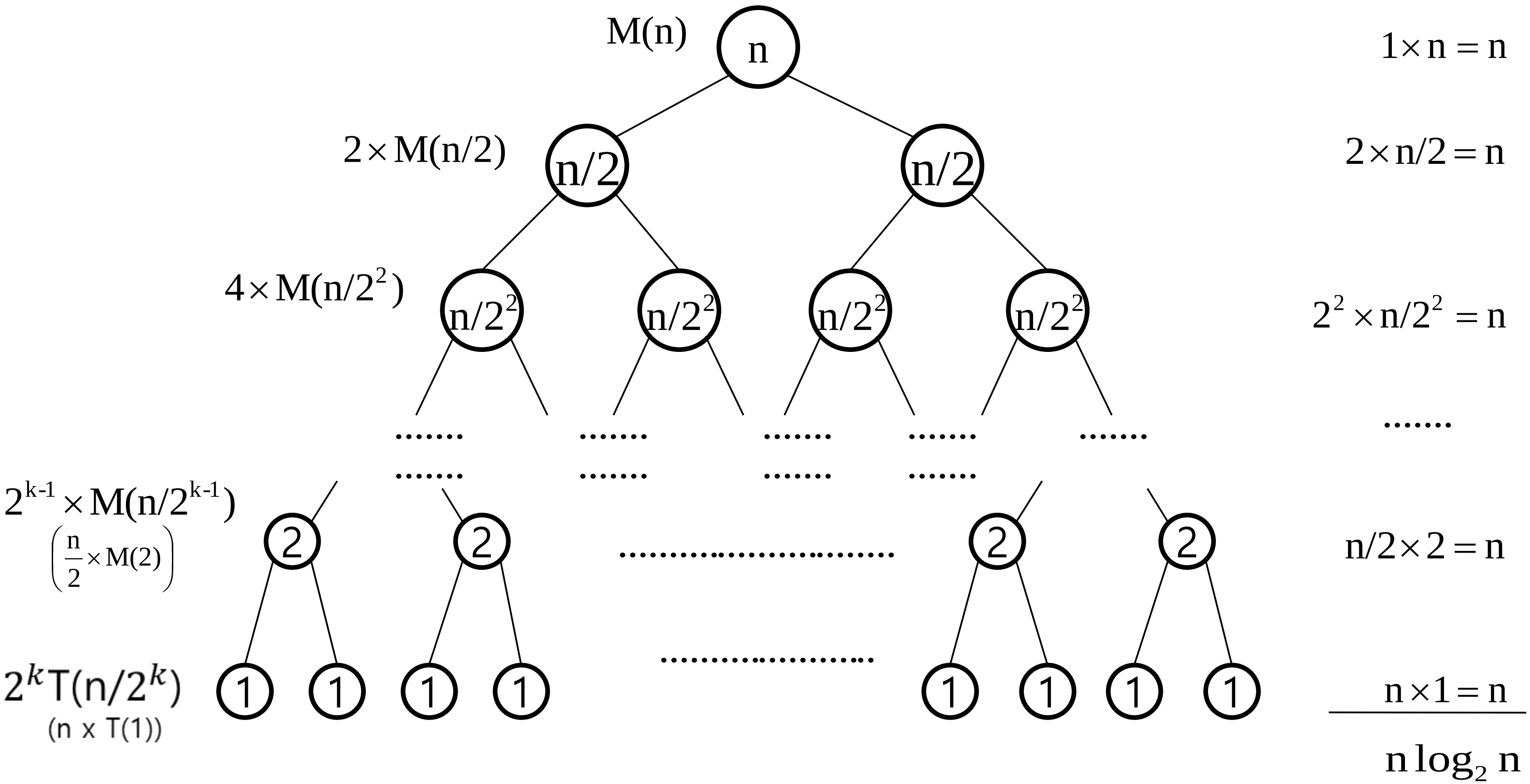
$$\text{Sum}[i, M] = \begin{cases} X[M] & \text{if } i = M \\ \text{Sum}(i+1, M) + X[i] & \text{otherwise} \end{cases}$$

∴ O(M) time to compute MaxCA

Similarly, MaxCB can also be computed O(M) time

∴ M(n)=O(M) + O(M)=O(n)

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2T(n/2) + n & \text{if } n > 1 \end{cases} \qquad T(n) = O(n \log n)$$



```
function Max-Sub(L, U)
    begin
        if L=U then return(Max{0, X[L]})

        M ← ⌊(L + U)/2⌋
        MaxA = Max-Sub(L, M)
        MaxB = Max-Sub(M+1,U)

        Sum ← 0; MaxCA ← 0
        For i ← M downto 1 do
                Sum ← Sum + X[i]
                MaxCA ← max{ MaxCA, Sum }
        end-do

        Sum ← 0; MaxCB ← 0
        For i ← M+1 to n do
                Sum ← Sum+X[i]
                MaxCB ← max{ MaxCB, Sum }
        end-do
        MaxAB ← MaxCA + MaxCB
        return( max{ MaxA, MaxB, MaxAB } )
    end
```

O(n) Algorithm

Given MaxSoFar and MaxTail for X[1..k-1], how to update them for X[1..k] ?

Initially,
Maxtail = {0, X[1]}
Maxsofar = Maxtail

In general,
MaxTail = max{0, MaxTail + X[k]}
MaxSoFar = max{MaxSoFar, MaxTail}

```
Procedure Max-sub
    begin
        MaxmSoFar ← 0;   MaxTail ← 0
        for k ← 1 to n
                MaxTail ← max{ 0, MaxTail+X[k] }
                MaxSoFar ← max{ MaxSoFar, MaxTail }
        end-do
    end
```

Example

MaxTail ← max{ 0, MaxTail+X[k] }
MaxSoFar ← max{ MaxSoFar, MaxTail }

$$X[1:10] = (31, -41, \underset{\uparrow 3}{59}, 26, -53, 58, \underset{\uparrow 7}{97}, -93, -23, 84)$$

k	MaxTail	MaxSoFar
1	31 (X[1..1]	31 (X[1..1])
2	0 (Ø)	31 (X[1..1])
3	59 (X[3..3])	59 (X[3..3])
4	85 (X[3..4])	85 (X[3..4])
5	32 (X[3..5])	85 (X[3..4])
6	90 (X[3..6])	90 (X[3..6])
7	187 (X[3..7])	187 (X[3..7])
8	94 (X[3..8])	187 (X[3..7])
9	71 (X[3..9])	187 (X[3..7])
10	155 (X[3..10])	187 (X[3..7])

Summary

ALGORITHM		1	2	3	4
Run time in nanoseconds		$1.3 n^3$	$10 n^2$	$47 n \log_2 n$	$48 n$
Time to solve a problem of size	10^3	1.3 secs	10 msecs	.4 msecs	.05 msecs
	10^4				
	10^5				
	10^6				
	10^7				
Max size problem solved in one	sec				
	min				
	hr				
	day				
If n multiplies by 10, time multiplies by					
If time multiplies by 10, n multiplies by					

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

Extreme Comparison