# Divide-and-Conquer

**General idea:**

**Divide a problem into subprograms of the same kind; solve subprograms using the same approach, and combine partial solution (if necessary).**



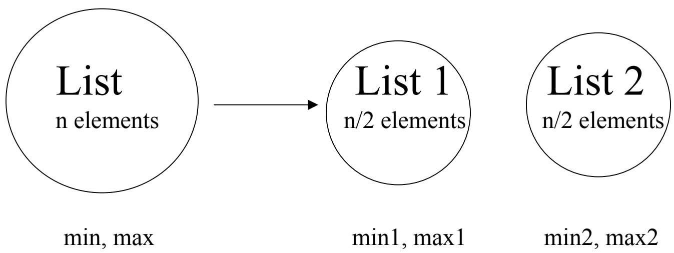
**1. Find the maximum and minimum**

*The problem:* Given a list of unordered $n$ elements, find max and min

*The straightforward algorithm:*

max ← min ← A (1);
for $i$ ← 2 to n do
  if A ($i$) > max, max ← A ($i$);
  if A ($i$) < min, min ← A ($i$);

Key comparisons: 2(n – 1)




**min = MIN ( min1, min2 )**
**max = MAX ( max1, max2)**



*The Divide-and-Conquer algorithm:*

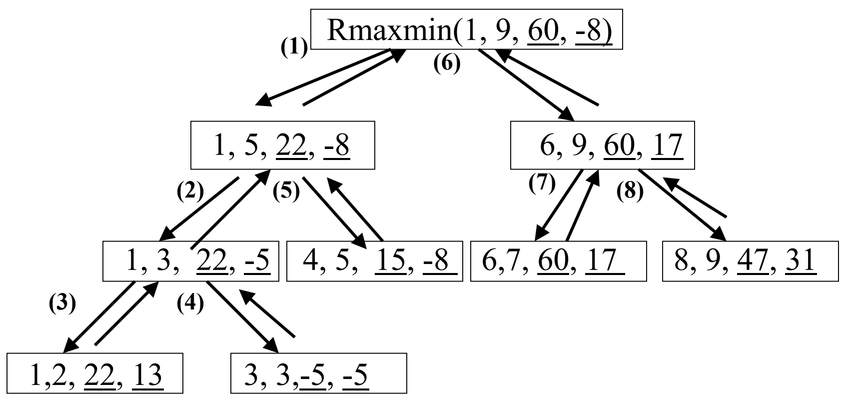
procedure Rmaxmin ($i$, $j$, *fmax*, *fmin*); // $i$, $j$ are index #, *fmax*,
begin // *fmin* are output parameters
case:
- $i = j$: *fmax* ← *fmin* ← A ($i$);
- $i = j - 1$: if A ($i$) < A ($j$) then *fmax* ← A ($j$); *fmin* ← A ($i$);
  else *fmax* ← A ($i$); *fmin* ← A ($j$);
- else: *mid* ← $\left\lfloor \frac{i+j}{2} \right\rfloor$;
  call Rmaxmin ($i$, *mid*, *gmax*, *gmin*);
  call Rmaxmin (*mid*+1, $j$, *hmax*, *hmin*);
  *fmax* ← MAX (*gmax*, *hmax*);
  *fmin* ← MIN (*gmin*, *hmin*);

end
end;

Eg: find max and min in the array:

22, 13, -5, -8, 15, 60, 17, 31, 47 ( n = 9 )

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Index: | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Array: | 22 | 13 | -5 | -8 | 15 | 60 | 17 | 31 | 47 |






*Analysis:* *For algorithm containing recursive calls, we can use recurrence relation to find its complexity*

T(n) - # of comparisons needed for Rmaxmin

Recurrence relation:

$$\begin{cases} T(n) = 0 & n = 1 \\ T(n) = 1 & n = 2 \\ T(n) = 2T(\frac{n}{2}) + 2 & \textit{otherwise} \end{cases}$$

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

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

$$= 2^2 \cdot (2T(\frac{n}{8}) + 2) + 2^2 + 2 = 2^3 \cdot T(\frac{n}{2^3}) + 2^3 + 2^2 + 2$$

• • •

Assume $n = 2^k$ for some integer $k$

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

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

$$= 1.5n - 2$$



They don't have to be the same constant.
We make them the same here for simplicity.
It will not affect the overall result.

**Theorem:**

$$T(n) = \begin{cases} b & n = 1 \\ aT(\frac{n}{c}) + bn & n > 1 \end{cases}$$

where are $a$, $b$, $c$ constants

**Claim:**

$$T(n) = \begin{cases} O(n) & a < c \\ O(nLog_c n) & a = c \\ O(n^{Log_c a}) & a > c \end{cases}$$

**Proof:** Assume $n = c^k$

$$T(n) = a \cdot T(\frac{n}{c}) + bn$$

$$T(n) = a \cdot (a \cdot T(\frac{n}{c^2}) + b \cdot \frac{n}{c}) + bn = a^2 \cdot T(\frac{n}{c^2}) + ab \cdot \frac{n}{c} + bn$$

$$= a^2 \cdot (a \cdot T(\frac{n}{c^3}) + b \cdot \frac{n}{c^2}) + ab \cdot \frac{n}{c} + bn$$

$$= a^3 \cdot T(\frac{n}{c^3}) + bn \cdot (\frac{a^2}{c^2} + \frac{a}{c} + 1)$$

$$\ldots$$

$$= a^k \cdot T(\frac{n}{c^k}) + bn \cdot (\frac{a^{k-1}}{c^{k-1}} + \frac{a^{k-2}}{c^{k-2}} + \ldots + \frac{a^0}{c^0})$$

$$= a^k \cdot b + bn \cdot ((\frac{a}{c})^{k-1} + (\frac{a}{c})^{k-2} + \ldots + (\frac{a}{c})^0)$$

$$= a^k \cdot b \cdot \frac{n}{c^k} + bn \cdot ((\frac{a}{c})^{k-1} + (\frac{a}{c})^{k-2} + \ldots + (\frac{a}{c})^0)$$

$$= bn \cdot (\frac{a}{c})^k + bn \cdot ((\frac{a}{c})^{k-1} + (\frac{a}{c})^{k-2} + \ldots + (\frac{a}{c})^0)$$

$$= bn \cdot (\sum_{i=0}^{k} (\frac{a}{c})^i)$$



If a < c, $\sum_{i=0}^{k} (\frac{a}{c})^i$ is a constant

$T(n) = bn \cdot$ a constant

$\therefore\ T(n) = O(n)$

if $a = c, \sum_{i=0}^{k} (\frac{a}{c})^i = k + 1$

$$T(n) = bn \sum_{i=0}^{k} 1 = bn \cdot (k+1)$$

$$= bn \cdot (Log_c n + 1)$$

$$= O(nLogn)$$

If $a > c$, $\sum_{i=0}^{k} (\frac{a}{c})^i = \frac{(\frac{a}{c})^{k+1} - 1}{(\frac{a}{c}) - 1}$

$$T(n) = bn \cdot \sum_{i=0}^{k} (\frac{a}{c})^i = bn \cdot \frac{(\frac{a}{c})^{k+1} - 1}{(\frac{a}{c}) - 1}$$

$$\leq bn \cdot (\frac{a}{c})^k = b \cdot a^k = b \cdot a^{Log_c n}$$

$$= b \cdot n^{Log_c a}$$

$$= O(n^{Log_c a})$$



## 2. Integer multiplication

*The problem:*

Multiply two large integers ($n$ digits)

*The traditional way:*

Use two for loops, it takes operations

*The Divide-and-Conquer way:*

Suppose large integers, , divide into two part $a$ and $b$, same as into $c$ and $d$.

$x$: $\boxed{a \mid b}$ $\therefore$ $x = a \cdot 10^{\frac{n}{2}} + b$

$y$: $\boxed{c \mid d}$ $\therefore$ $y = c \cdot 10^{\frac{n}{2}} + d$

$$x \cdot y = (a \cdot 10^{\frac{n}{2}} + b)(c \cdot 10^{\frac{n}{2}} + d)$$

$$= \underline{ac} \cdot 10^{n} + \underline{bd} + 10^{\frac{n}{2}}(\underline{ad} + \underline{bc})$$

So, transform the problem of multiply two integers of $n$-digit into four subproblems of multiply two integers of $\frac{n}{2}$ -digit



Worst-Case is:

$$T(n) = 4 \cdot T(\frac{n}{2}) + bn$$

$$= O(n^{Log_2 4}) = O(n^2)$$

however, it is same as the traditional way.

Therefore, we need to improve equation show before:

• • •

$$= ac \cdot 10^{n} + bd + 10^{\frac{n}{2}} \cdot (ad + bc)$$

$$= \underline{ac} \cdot 10^{n} + \underline{bd} + 10^{\frac{n}{2}} \cdot (\underline{(a+b)(c+d)} - ac - bd)$$

Worst-Case is:

$$T(n) = 3 \cdot T(\frac{n}{2}) + bn$$

$$= O(n^{Log_2 3}) \approx O(n^{1.58})$$

*The algorithm by Divide-and-Conquer:*

Large-int function multiplication $(x,y)$

begin

$n$ = MAX ( # of digits in $x$, #of digits in $y$);

if ($x$ = 0) or ($y$ = 0), return 0;

else if ($n$ = 1), return $x*y$ in the usual way;

else

$m = \left\lfloor \frac{n}{2} \right\rfloor$;

$a = x$ divide $10^m$;

$b = x$ rem $10^m$;

$c = y$ divide $10^m$;

$d = y$ rem $10^m$;

$p1$= MULTIPLICATION ($a, c$);

$p2$= MULTIPLICATION ($b, d$);

$p3$ = MULTIPLICATION ($a+b, c+d$);

return $p_1 \cdot 10^{2m} + p_2 + 10^m (p_3 - p_1 - p_2)$;

end;



example: $x = 143, y = 256$

| | | | | | | |
|---|---|---|---|---|---|---|
| $P$ | $x$ = 143<br>$y$ = 256 | $n$ = 3 | $m$ = 1 | $a$ = 14<br>$b$ = 3<br>$c$ = 25<br>$d$ = 6 | $p_1 = p'$<br>$p_2$= 18<br>$p_3 = p''$ | $P$=36608 |
| $P'$ | $x$ = 14<br>$y$ = 25 | $n$ = 2 | $m$ = 1 | $a$ = 1<br>$b$ = 4<br>$c$ = 2<br>$d$ = 5 | $p_1$ = 2<br>$p_2$ = 20<br>$p_3$ = 35 | $P'$=350 |
| $P''$ | $x$ = 17<br>$y$ = 31 | $n$ = 2 | $m$ = 1 | $a$ = 1<br>$b$ = 7<br>$c$ = 3<br>$d$ = 1 | $p_1$ = 3<br>$p_2$ = 7<br>$p_3$ = 32 | $P''$=527 |

## 3. Merge Sort

*The problem:*

Given a list of $n$ numbers, sort them in non-decreasing order

idea: Split each part into 2 equal parts and sort each part using Mergesort, then merge the tow sorted sub-lists into one sorted list.

*The algorithm:*

procedure MergeSort ($low$, $high$)

begin

if $low < high$

then

$mid \leftarrow \left\lfloor \frac{low + high}{2} \right\rfloor$

call MergeSort ($low$, $mid$);

call MergeSort ($mid$+1, $high$);

call Merge ($low$, $mid$, $high$);

end;



procedure Merge ($low$, $mid$, $high$)

begin $i \leftarrow low, j \leftarrow mid + 1, k \leftarrow low$

while (i $\leq mid$ and $j \leq high$)

if $A(i) < A(j)$, then $U(k) \leftarrow A(i)$; $i$++;

*else* $U(k) \leftarrow A(j)$; $j$++;

$k$++;

if ($i > mid$)

move $A(j)$ through $A(high)$ to $U(k)$ through $U(high)$;

*else*

move $A(i)$ through $A(mid)$ to $U(k)$ through $U(high)$;

end;

*Analysis:*

Worst-Case for MergeSort is:

$$T(n) = 2 \cdot T(\frac{n}{2}) + bn$$
$$= O(nLogn)$$

Average-Case? Merge sort pays no attention to the original order of the list, it will keep divide the list into half until sub-lists of length 1, then start merging.

Therefore Average-Case is the same as the Worst-Case.



## 4. Quick Sort

*The problem:*

Same as Merge Sort

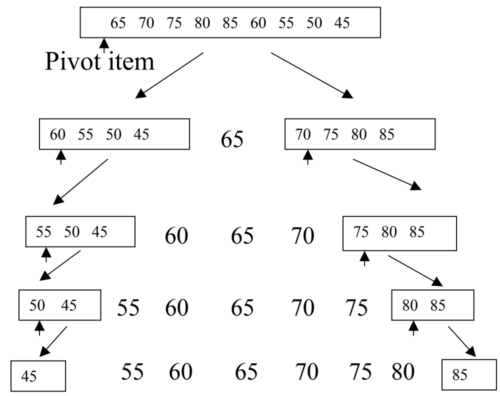
*Compared with Merge Sort:*

- Quick Sort also use Divide-and-Conquer strategy
- Quick Sort eliminates merging operations

## *How it works?*

Use partition

Eg: Sort array 65, 70, 75, 80, 85, 60, 55, 50, 45






## *The algorithm:*

procedure **QuickSort** (*p*, *q*)
begin
if $p < q$,
then call Partition (*p*, *q*, *pivotposition*);
call QuickSort (*p*, *pivotposition* –1);
call QuickSort (*pivotposition*+1, *q*);
end;

procedure **Partition** (*low*, *high*, *pivotposition*)
begin
$v \leftarrow A(low);$
$j \leftarrow low;$
for $i \leftarrow (low + 1)$ to *high*
if $A(i) < v$,
$j$++;
$A(i) \leftrightarrow A(j);$
$pivotposition \leftarrow j;$
$A(low) \leftrightarrow A(pivotposition);$
end;

*Analysis:*

Worst-Case:

Call Partition $O(n)$ times, each time takes $O(n)$ steps. *So* $O(n^2)$, and it is worse than Merge Sort in the Worst-Case.

Best-Case:

Split the list evenly each time.

$$O(nLogn)$$



Average-Case:

Assume pivot is selected randomly, so the probability of pivot being the $k^{\text{th}}$ element is equal, $\forall k$.

$$prob(pivot \leftarrow k) = \frac{1}{n}, \qquad \forall k$$

$$\begin{aligned} C(n) &= \text{\# of key comparison for } n \text{ items} \\ &= (n-1) + C\cdot(k-1) + C\cdot(n-k) \end{aligned}$$

• average it over all $k$, ($C_A(n)$ is average performance)

$$C_A(n) = (n-1) + \frac{1}{n}\sum_{k=1}^{n}(C_A(k-1) + C_A(k-2))$$

• multiply both side by $n$

$$n\cdot C_A(n) =$$

$$n(n-1) + 2\cdot(C_A(0) + C_A(1) + \cdots + C_A(n-1)) \quad (1)$$

• replace $n$ by $n$-1

$$(n-1)\cdot C_A(n-1) =$$
$$(n-1)(n-2)+2\cdot(C_A(0)+\cdots+C_A(n-2)) \quad (2)$$

• subtract (2) from (1)

$$n\cdot C_A(n)-(n-1)C_A(n-1) = 2\cdot(n-1)+2\cdot C_A(n-1)$$
$$\Rightarrow n\cdot C_A(n) = 2\cdot(n-1)+(n+1)\cdot C_A(n-1)$$



• divide $n(n + 1)$ for both sides

$$\frac{C_A(n)}{n+1} = \frac{C_A(n-1)}{n}+\frac{2\cdot(n-1)}{n(n+1)}$$
$$= \frac{C_A(n-2)}{n-1}+\frac{2\cdot(n-2)}{(n-1)\cdot n}+\frac{2\cdot(n-1)}{n(n+1)}$$
$$= \frac{C_A(n-3)}{n-2}+\frac{2\cdot(n-3)}{(n-2)\cdot(n-1)}+\frac{2\cdot(n-2)}{(n-1)\cdot n}+\frac{2\cdot(n-1)}{n(n+1)}$$
$$\cdots$$

$$\begin{bmatrix} Note: & \\ & C_A(1)=0 \end{bmatrix}$$

$$= \frac{C_A(1)}{2}+2\cdot\left(\frac{1}{2\cdot 3}+\frac{2}{3\cdot 4}+\cdots+\frac{n-1}{n(n+1)}\right)$$
$$= 0+2\cdot\sum_{k=2}^{n}\frac{k-1}{k(k+1)}$$
$$\le 2\cdot\sum_{k=2}^{n}\frac{1}{k} \le 2\cdot\int_1^n\frac{1}{k}dk = 2\cdot(Log_e n - Log_e 1)$$
$$= O(Logn)$$
$$\Rightarrow C_A(n) = O(nLogn)$$

$\therefore$ Best-Case = Average-Case

## 5. Selection

*The problem:*

Given a list of $n$ elements find the $k^{th}$ smallest one

Note: special cases: when $k = 1$, *min*
when $k = n$, *max*

*The solving strategy:*

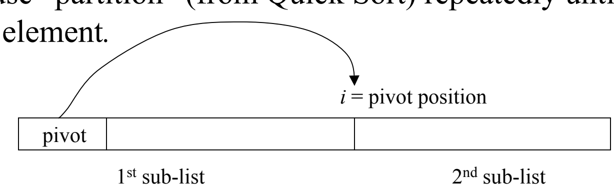
If use sorting strategy like MergeSort, it takes $O(nLogn)$

If use Quick Sort, Best-Case is $O(n)$, Worst-Case will call partition $O(n)$ times, each time takes $O(n)$, so Worst-Case is $O(n^2)$.



## *Select1 – the first algorithm*

*idea:* use "partition" (from Quick Sort) repeatedly until we find the $k^{th}$ element.




For each iteration:

If pivot position = $k \Rightarrow$ Done!

If pivot poaition < $k \Rightarrow$ to select $(k\text{-}i)^{th}$ element in the $2^{nd}$ sub-list.

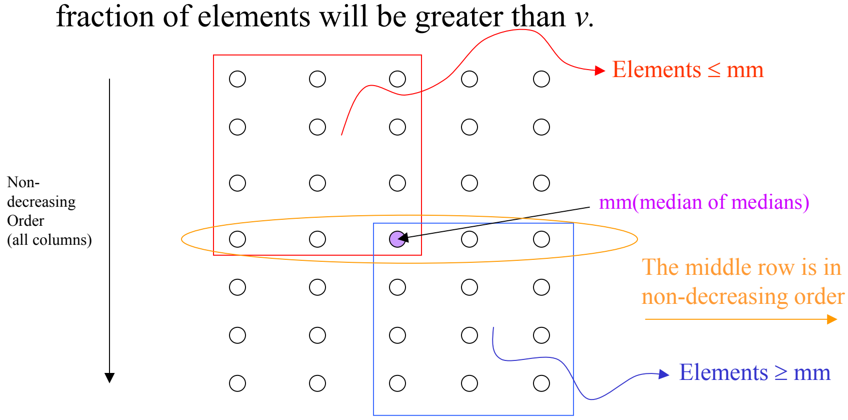
If pivot poaition > $k \Rightarrow$ to select $k^{th}$ element in the $1^{st}$ sub-list.

```
procedure Select1 (A, n, k)     // A is array, n is # of array, k is key
    begin
        m ← 1;
        j ← n;
        loop
            call Partition (m, j, pivotposition);
            case:
                k = pivotposition:
                    return  A(k);
                k < pivotposition:
                    j ← pivotposition – 1;
                else:
                    m ← pivotposition + 1;
        end loop;
    end;
```



3) By choosing pivot more carefully, we can obtain a selection algorithm with Worst-Case complexity $O(n)$

   How: Make sure pivot $v$ is chosen s.t. at least some fraction of the elements will be smaller than $v$ and at least some other fraction of elements will be greater than $v$.



3) By choosing pivot more carefully, we can obtain a selection algorithm with Worst-Case complexity $O(n)$

Use *mm* (median of medians) rule to choose pivot

procedure Select2 ($A$, $n$, $k$)
begin
if $n \le r$, then sort $A$ and return the $k^{th}$ element;

divide $A$ into $\left\lfloor \frac{n}{r} \right\rfloor$ subset of size $r$ each, ignore excess elements, and let $M = \left\{ m_1, m_2, \cdots, m_{\left\lfloor \frac{n}{r} \right\rfloor} \right\}$ be the set of medians of the $\left\lfloor \frac{n}{r} \right\rfloor$ subsets;



$v \leftarrow$ Select2 $\left(M, \left\lfloor \frac{n}{r} \right\rfloor, \left\lceil \left\lfloor \frac{n}{r} \right\rfloor \Big/ 2 \right\rceil\right)$;

use "Partition" to partition $A$ using $v$ as the pivot;
// Assume $v$ is at *pivotposition*

case:
$k$ = *pivotposition*: return ($v$);

$k$ < *pivotposition*: let $S$ be the set of elements $A$ (1,…, *pivotposition* –1), return Select2 ($S$, *pivotposition* –1, $k$);

else: let $R$ be the set of element $A$ (*pivotposition*+1,…, $n$), return Select2 ($R$, $n$- *pivotposition*, $k$-*pivotposition*);
end case;
end;

*Analysis:*

- How many $M_i$'s $\leq$ or $\geq$ *mm*?

  At least $\left\lceil \left\lfloor \frac{n}{r} \right\rfloor / 2 \right\rceil$

- How many elements $\leq$ or $\geq$ *mm*?

  At least $\left\lceil \frac{r}{2} \right\rceil \cdot \left\lceil \left\lfloor \frac{n}{r} \right\rfloor / 2 \right\rceil$

- How many elements in $|R| > mm$ or $|S| < mm$ ?

  At most $n - (\left\lceil \frac{r}{2} \right\rceil \cdot \left\lceil \left\lfloor \frac{n}{r} \right\rfloor / 2 \right\rceil)$

  Assume $r = 5$ $\quad \therefore = n - (3 \cdot \left\lceil \left\lfloor \frac{n}{5} \right\rfloor / 2 \right\rceil) \leq n - 1.5 \left\lfloor \frac{n}{5} \right\rfloor$

$$\leq n - 1.5 \cdot \frac{n-4}{5} = 0.7n + 1.2$$

$$\leq 0.75n \qquad (n \geq 24)$$



- Therefore, procedure Select2 the Worst-Case complexity is:

$$T(n) = T(\frac{n}{5}) + T(\frac{3}{4}n) + cn$$

where $c$ is chosen sufficiently large, such that $T(n) \leq cn$ for $n < 24$.

**Proof**:

Use induction to show $T(n) \leq 20 \bullet cn \quad (n \geq 24)$

**IB** (induction base):

$$n = 24, \qquad T(n) = T(\frac{24}{5}) + T(\frac{3}{4} \cdot 24) + c \cdot 24$$

$$\leq cn + cn + 24c \leq 20 \cdot cn$$

**IH** (induction hypothesis):

Suppose $T(n) \le 20 \cdot cn \quad \forall 24 \le n < m$

**IS** (induction solution):

When $n = m$;

$$T(m) = T(\frac{m}{5}) + T(\frac{3}{4}m) + cm$$
$$\le 20 \cdot c \cdot \frac{m}{5} + 20 \cdot c \cdot \frac{3}{4} \cdot m + cm$$
$$\le 20 \cdot cn$$

$\therefore$ $T(n) = O(n)$ complexity of Select2 of n elements



## 6. Matrix multiplication

*The problem*:

Multiply two matrices $A$ and $B$, each of size $[n \times n]$

$$\begin{bmatrix} & & \\ & A & \\ & & \end{bmatrix}_{n\times n} \cdot \begin{bmatrix} & & \\ & B & \\ & & \end{bmatrix}_{n\times n} = \begin{bmatrix} & & \\ & C & \\ & & \end{bmatrix}_{n\times n}$$

*The traditional way*:

$$C_{ij} = \sum_{k=1}^{n} A_{ik} \times B_{kj}$$

use three for-loop

$$\therefore T(n) = O(n^3)$$



*The Divide-and-Conquer way*:

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \cdot \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$C_{11} = \underline{A_{11} \cdot B_{11}} + \underline{A_{12} \cdot B_{21}}$$
$$C_{12} = \underline{A_{11} \cdot B_{12}} + \underline{A_{12} \cdot B_{22}}$$
$$C_{21} = \underline{A_{21} \cdot B_{11}} + \underline{A_{22} \cdot B_{21}}$$
$$C_{22} = \underline{A_{21} \cdot B_{12}} + \underline{A_{22} \cdot B_{22}}$$

transform the problem of multiplying $A$ and $B$, each of size [n×n] into 8 subproblems, each of size $\left[\frac{n}{2} \times \frac{n}{2}\right]$

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

$$= O(n^3)$$

which $an^2$ is for addition

so, it is no improvement compared with the traditional way



Eg:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

use Divide-and-Conquer way to solve it as following:

$$\begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 3 & 3 & 3 & 0 \\ 3 & 3 & 3 & 0 \\ 3 & 3 & 3 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$C_{11} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 3 & 3 \\ 3 & 3 \end{bmatrix}$$

$$C_{12} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 3 & 0 \\ 3 & 0 \end{bmatrix}$$

$$C_{21} = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 3 & 3 \\ 0 & 0 \end{bmatrix}$$

$$C_{22} = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 3 & 0 \\ 0 & 0 \end{bmatrix}$$

*Strassen's matrix multiplication*:

- Discover a way to compute the $C_{ij}$'s using 7 multiplications and 18 additions or subtractions

$$P = (A_{11} + A_{22})(B_{11} + B_{22})$$
$$Q = (A_{21} + A_{22})B_{11}$$
$$R = A_{11}(B_{12} - B_{22})$$
$$S = A_{22}(B_{21} - B_{11})$$
$$T = (A_{11} + A_{12})B_{22}$$
$$U = (A_{21} - A_{11})(B_{11} + B_{12})$$
$$V = (A_{12} - A_{22})(B_{21} + B_{22})$$
$$C_{11} = P + S - T + V$$
$$C_{12} = R + T$$
$$C_{21} = Q + S$$
$$C_{22} = P + R - Q + U$$



•Algorithm :

procedure Strassen *(n, A, B, C)* // *n* is size, *A*,*B* the input matrices, *C* output matrix

begin

if $n = 2$,

$$\begin{cases} C_{11} = a_{11} \cdot b_{11} + a_{12} \cdot b_{21}; \\ C_{12} = a_{11} \cdot b_{12} + a_{12} \cdot b_{22}; \\ C_{21} = a_{12} \cdot b_{11} + a_{22} \cdot b_{21}; \\ C_{22} = a_{12} \cdot b_{12} + a_{22} \cdot b_{22}; \end{cases}$$

else

(cont.)

else

Partition $A$ into 4 submatrices: $A_{11}, A_{12}, A_{21}, A_{22}$;
Partition $B$ into 4 submatrices: $B_{11}, B_{12}, B_{21}, B_{22}$ ;
call Strassen $(\frac{n}{2}, A_{11} + A_{22}, B_{11} + B_{22}, P)$ ;

call Strassen $(\frac{n}{2}, A_{21} + A_{22}, B_{11}, Q)$;

call Strassen $(\frac{n}{2}, A_{11}, B_{12} - B_{22}, R)$;

call Strassen $(\frac{n}{2}, A_{22}, B_{21} - B_{11}, S)$;

call Strassen $(\frac{n}{2}, A_{11} + A_{12}, B_{22}, T)$;



call Strassen $(\frac{n}{2}, A_{21} - A_{11}, B_{11} + B_{12}, U)$;

call Strassen $(\frac{n}{2}, A_{21} - A_{11}, B_{11} + B_{12}, U)$;

$C_{11} = P + S - T + V$;

$C_{12} = R + T$;

$C_{21} = Q + S$;

$C_{22} = P + R - Q + U$;

end;

*Analysis*:

$$T(n) = \begin{cases} 7 \cdot T(\frac{n}{2}) + an^2 & n > 2 \\ b & n \le 2 \end{cases}$$

$$T(n) = 7 \cdot T(\frac{n}{2}) + an^2$$

$$= 7^2 \cdot T(\frac{n}{2^2}) + (\frac{7}{4})an^2 + an^2$$

$$= 7^3 \cdot T(\frac{n}{2^3}) + (\frac{7}{4})^2 \cdot an^2 + (\frac{7}{4}) \cdot an^2 + an^2$$

$$\bullet \bullet \bullet$$



Assume $n = 2^k$ for some integer $k$

$$= 7^{k-1} \cdot T(\frac{n}{2^{k-1}}) + an^2 \cdot \left[ (\frac{7}{4})^{k-2} + \cdots + 1 \right]$$

$$= 7^{k-1} \cdot b + an^2 \left[ \frac{(\frac{7}{4})^{k-1} - 1}{\frac{7}{4} - 1} \right]$$

$$\le b \cdot 7^k + c \cdot n^2 \cdot (\frac{7}{4})^k$$

$$= b \cdot 7^{Lgn} + cn^2 \cdot (\frac{7}{4})^{Lgn} = b \cdot 7^{Lgn} + cn^2 (n)^{Lg\frac{7}{4}}$$

$$= b \cdot n^{Lg7} + cn^{Lg7} = (b + c) \cdot n^{Lg7}$$

$$= O(n^{Lg7}) = O(n^{2.81})$$