Computer Engineering - Sem IV

NCMPC 41 : Design and Analysis of Algorithms

Module - 2 : Divide and Conquer Strategy
(6 Hours)

Instructor : Mrs. Lifna C S

Department of Computer Engineering, VESIT, Mumbai

Topics to be covered

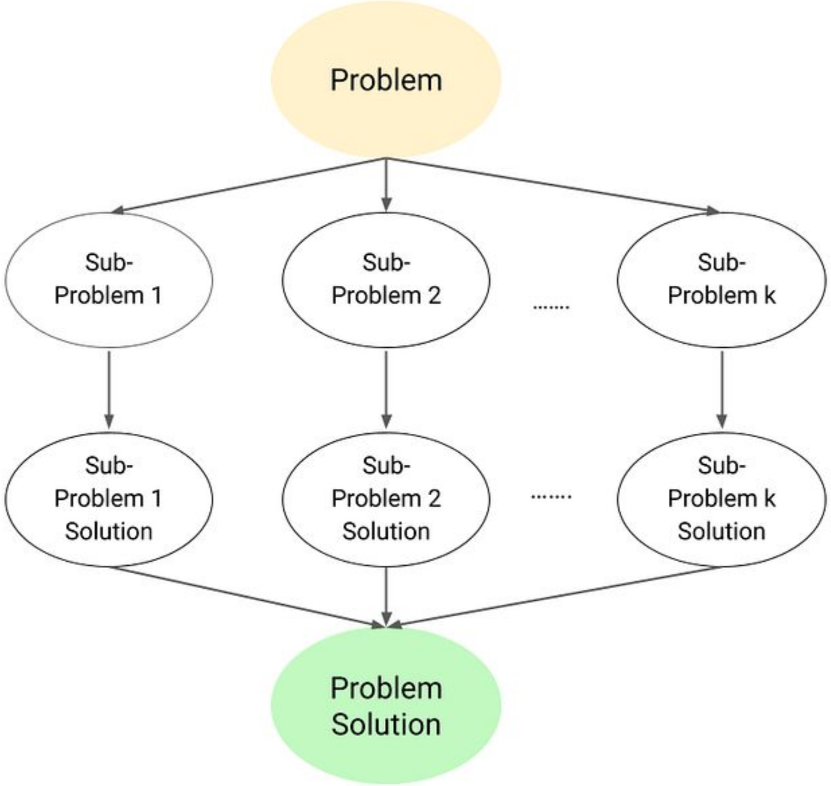
- **General method**
- Min-Max Algorithm
- Merge sort
- Quick sort
- Analysis of Binary search
- Strassen's Matrix Multiplication.

Divide and Conquer Strategy

Divide
Dividing the problem into smaller sub-problems

Conquer
Solving each sub-problems recursively

Combine
Combining sub-problem solutions to build the original problem solution



Courtesy : [Easy Algorithms]

Divide and Conquer Strategy

Sort pblm, P. $\underline{\text{divided}}$ into subpblm of Sort P_i

NB pblm is to arrange a workshop

divide the tasks as
- poster making
- invitation to speakers
- invitation to participants
- lab arrangements
- certificate etc ...

Here tasks ≠ pblm.

Divide and Conquer Strategy

```
DAC (P)
  if (small(P))
      S(P)            // solve P.
  else
    [ divide P into P1, P2, P3, ..., Pk
      apply DAC(P1) DAC(P2) DAC(P3), ... DAC(Pk)
      Combine (DAC(P1), DAC(P2), ...
```

- Binary Search.
- Find Max. & Min
- Merge Sort
- Quick Sort
- Strassen's Matrix Multi

Topics to be covered

- General method
- **Min-Max Algorithm**
- Merge sort
- Quick sort
- Analysis of Binary search
- Strassen's Matrix Multiplication.

Min-Max Algorithm

Task : Given an array X[] of size n, write a program to find the maximum and minimum element in the array.

Goal : To complete the task with minimum no. of comparisons

Examples

Input: X[] = [4, 2, 0, 8, 20, 9, 2], Output: max = 20, min = 0

Input: X[] = [-8, -3, -10, -32, -1], Output: max = -1, min = -32

Courtesy : Medium

Min-Max Algorithm - Brute Force Method


```
int[] minMax(int X[], int n)
{
    int max = X[0]
    int min = X[0]
    for (int i = 1; i < n; i = i + 1)
    {
        if (X[i] > max)
            max = X[i]
        else if (X[i] < min)
            min = X[i]
    }
    int output[2] = {max, min}
    return output
}
```

1. Initialize two variables, max = min = X[0]
2. Traverse the array to compare each element with min & max.
   a. If X[i] < min, then min = X[i].
   b. if X[i] > max, then max = X[i].
3. Return min & max values

Courtesy : Medium

Min-Max Algorithm - Divide & Conquer Method

1. Divide array by calculating mid index

   i.e. mid = l + (r — l)/2

2. Recursively find the maximum and minimum of left part by calling the same function

   i.e. leftMinMax[2] = minMax(X, l, mid)

3. Recursively find the maximum and minimum of right part by calling the same function

   i.e. rightMinMax[2] = minMax(X, mid + 1, r)

4. Finally, get the overall maximum and minimum by comparing the min and max of both halves.

```
if (leftMinMax[0] > rightMinMax[0])
   max = lminMax[0]
else
   max = rightMinMax[0]
if (leftMinMax[1] < rightMinMax[1])
   min = leftMinMax[1]
else
   min = rightMinMax[1]
```

Courtesy : Medium

Min-Max Algorithm - Analysis

Recurrence Relation :

$T(n) = 2\,T(n/2) + 2$, where $T(2) = 1$ and $T(1) = 0$.

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

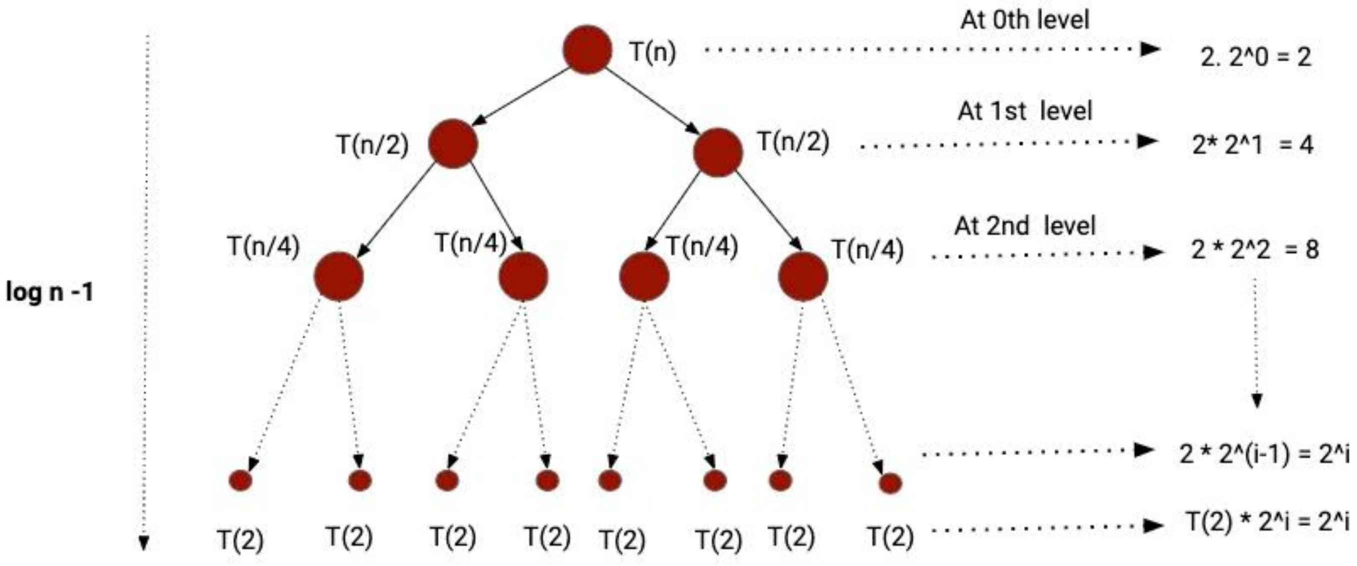
<u>Base cases</u>

1. If the array size = 1, return that single element as both max and min.
2. If the array size = 2, compare both elements and return maximum and minimum.

Auxiliary Space: **O(log n) as the stack space** will be filled for the maximum height of the tree formed during recursive calls same as a binary tree.

Courtesy : [Medium]

Min-Max Algorithm - Analysis



Courtesy : [Medium]

Min-Max Algorithm - Analysis w.r.t Comparisons

Number of Comparisons be T(n).

Strategy - 1 : Using Tournament Method:

If n is a power of 2, then we can write T(n) as: $T(n) = 2T(n/2) + 2$

$\Rightarrow T(n) = 3n/2 - 2$

$T(n) = 3n/2 - 2$ comparisons, if n is a power of 2.

$T(n) > 3n/2 - 2$ comparisons if n is not a power of 2.

Strategy - 2 : Comparing pairs

$T(n) = 3*(n-1)/2$, n is odd

$T(n) = 1 + 3*(n-2)/2 = 3n/2-2$, Otherwise

(1 comparison for initializing min and max,

3(n-2)/2 comparisons for rest of the elements

Courtesy : Medium

Topics to be covered

- General method
- Min-Max Algorithm
- **Merge sort**
- Quick sort
- Analysis of Binary search
- Strassen's Matrix Multiplication.

Merge Sort Algorithm

Algorithm 2: MergeSort(A)

```
n ← length(A);
if n ≤ 1 then
    return A;
L ← MergeSort(A[1 : n/2]);
R ← MergeSort(A[n/2 + 1 : n]);
return Merge(L, R);
```

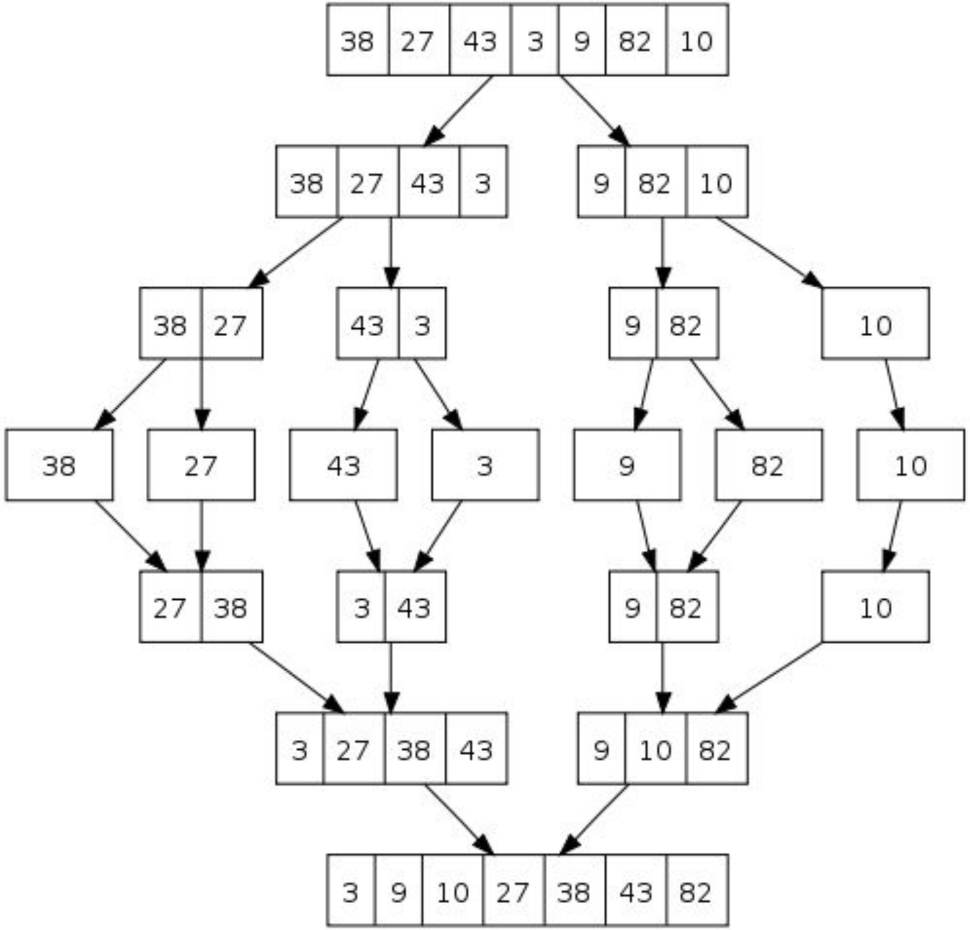
Now, we need to describe the Merge procedure, which takes two sorted arrays, L and R, and produces a sorted array containing the elements of L and R. Consider the following Merge procedure (Algorithm 3), which we will call as a subroutine in MergeSort.

Algorithm 3: Merge(L, R)

```
m ← length(L) + length(R);
S ← empty array of size m;
i ← 1; j ← 1;
for k = 1 → m do
    if L(i) < R(j) then
        S(k) ← L(i);
        i ← i + 1;
    else
        S(k) ← R(j);
        j ← j + 1;
return S;
```

Courtesy : Cormen et. al

Merge Sort Example



Merge Sort Analysis

(I)

$T_p(n)$	**MergeSort(A)**	
1	$n = len(A)$	
1	$if(n \leq 1)$	
1	return A	
$T_p(n/2)$	$L = MergeSort(A[1 \cdots n/2])$	
$T_p(n/2)$	$R = MergeSort(A[n/2+1 \cdots n])$	
$\Theta(m)$	return Merge(L,R)	

Merge Sort Analysis

$T_{P_1}(m)$	Merge(L, R)
1	m = len(L) + len(R)
1	S[m]
2	i = j = 1
m+1	for(k=1, K<=m, k++)
m	m if(L[i] < R[j])
	2 S[k]=L[i]
	i++
	else
	2 S[k]=R[j]
	j++
1	return S

$$T_{P_1}(m) = 4 + 2m + 1 = 2m + 5 = \Theta(m)$$

Merge Sort Analysis

$$T_p(n) = 2T_p\left(\frac{n}{2}\right) + \Theta(n^1) + 3 \quad \text{III}$$

$$aT\left(\frac{n}{b}\right) + \Theta(n^k \log^p n)$$

$a \geq 1 \quad b > 1 \quad k \geq 0 \quad p \in \mathbb{R}$

#subpblms pblm size work done

$a = 2 \quad b = 2 \quad k = 1 \quad p = 0$

$a \quad b^k$

$2 = 2^1$

Case 1 $a > b^k$

Case 2 $a = b^k$ — $p > -1$, $p = -1$, $p < -1$

Case 3 $a < b^k$

IV

$$T(n) = \Theta(n^{\log_b a} \log^{p+1} n)$$

$$= \Theta(n^{\log_2 2} \log^1 n)$$

$$= \Theta(n \log n)$$

Merge Sort

1. ***Divide:*** Trivial.
2. ***Conquer:*** Recursively sort 2 subarrays.
3. ***Combine:*** Linear-time merge.

$$T(n) = 2\,T(n/2) + \Theta(n)$$

 # subproblems

subproblem size

work dividing and combining

Merge sort: $a = 2,\ b = 2 \ \Rightarrow\ n^{\log_b a} = n^{\log_2 2} = n$

$\Rightarrow$ CASE 2 $(k = 0)$ $\Rightarrow$ $T(n) = \Theta(n \lg n)$.

Courtesy : [Cormen et. al]()

Merge Sort - Recurrence Relation

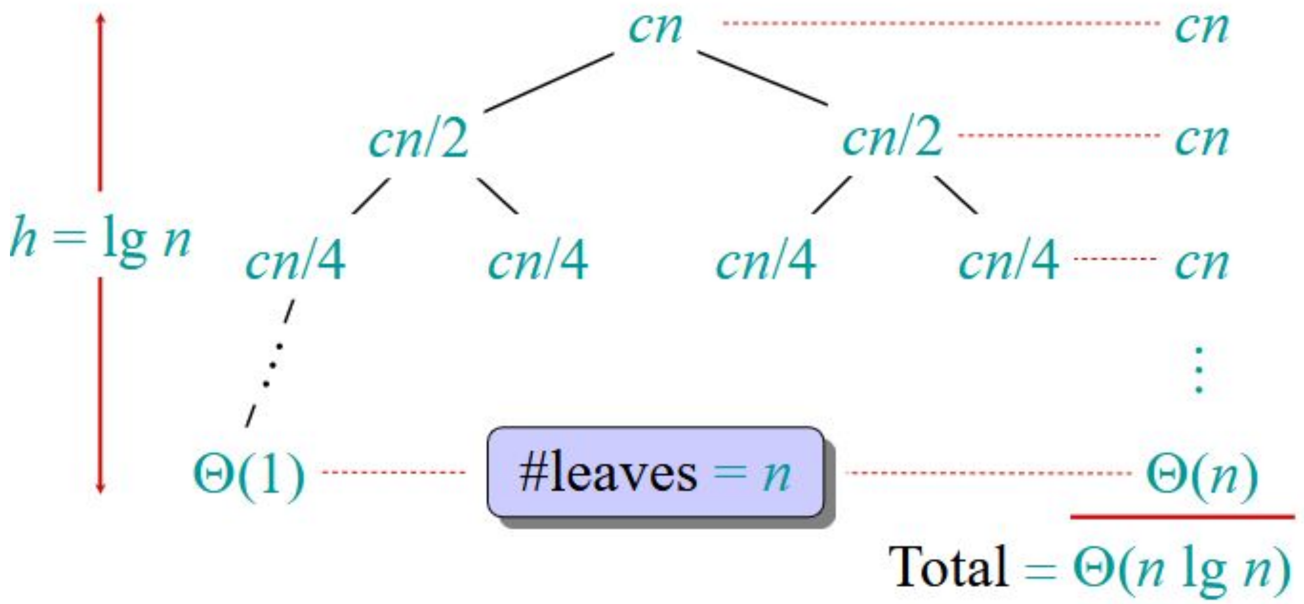
$$T(n) = \begin{cases} \Theta(1) \text{ if } n = 1; \\ 2T(n/2) + \Theta(n) \text{ if } n > 1. \end{cases}$$

- We shall usually omit stating the base case when $T(n) = \Theta(1)$ for sufficiently small n, but only when it has no effect on the asymptotic solution to the recurrence.

Courtesy : Cormen et. al

Merge Sort - Solving Recurrence Relation

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



Courtesy : [Cormen et. al]

Topics to be covered

- General method
- Min-Max Algorithm
- Merge sort
- **Quick sort**
- Analysis of Binary search
- Strassen's Matrix Multiplication.

Quick Sort

1. ***Divide:*** Partition the array into two subarrays around a ***pivot*** x such that elements in lower subarray $\leq x \leq$ elements in upper subarray.

2. ***Conquer:*** Recursively sort the two subarrays.
3. ***Combine:*** Trivial.

Key: *Linear-time partitioning subroutine.*

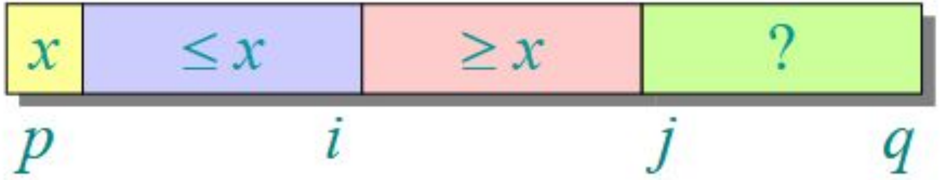
Courtesy : [Cormen et. al]

Quick Sort

PARTITION(A, p, q) ⊳ $A[p \,.\,.\, q]$
  $x \leftarrow A[p]$ ⊳ pivot $= A[p]$
  $i \leftarrow p$
  for $j \leftarrow p + 1$ **to** q
    do if $A[j] \leq x$
      then $i \leftarrow i + 1$
        exchange $A[i] \leftrightarrow A[j]$
  exchange $A[p] \leftrightarrow A[i]$
  return i

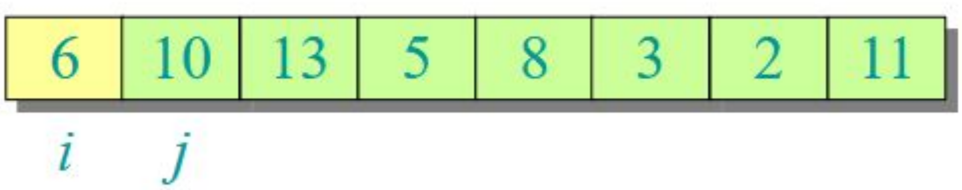
Running time $= O(n)$ for n elements.

Invariant:



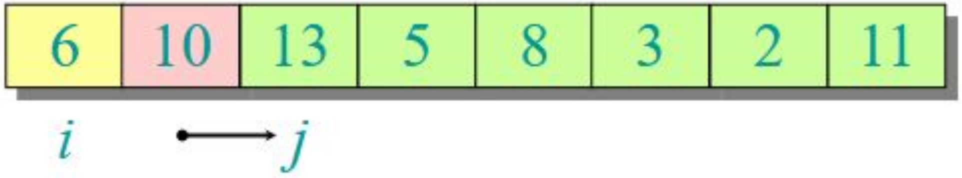
Courtesy : Cormen et. al

Quick Sort - Example of Partitioning



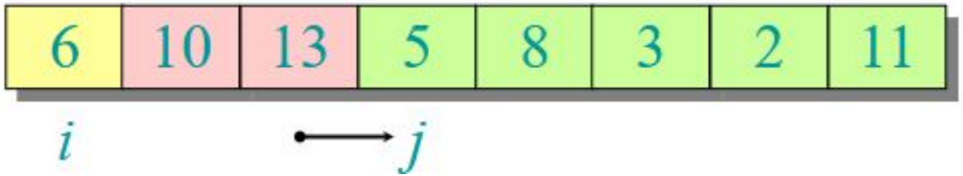
Courtesy : Cormen et. al

Quick Sort - Example of Partitioning



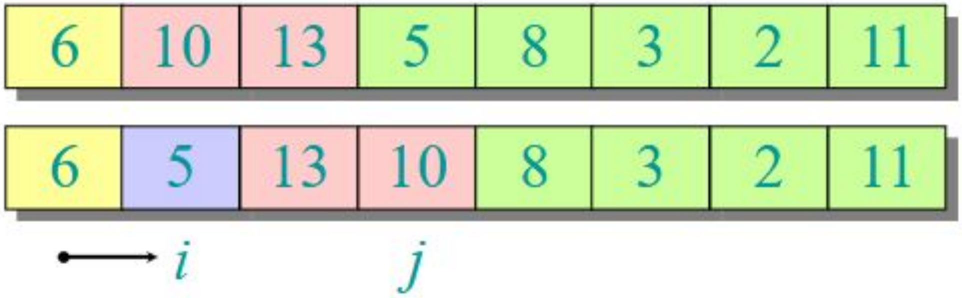
Courtesy : [Cormen et. al]()

Quick Sort - Example of Partitioning



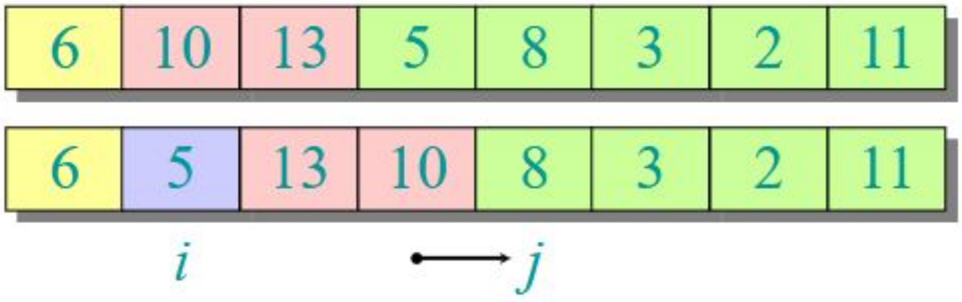
Courtesy : [Cormen et. al]

Quick Sort - Example of Partitioning



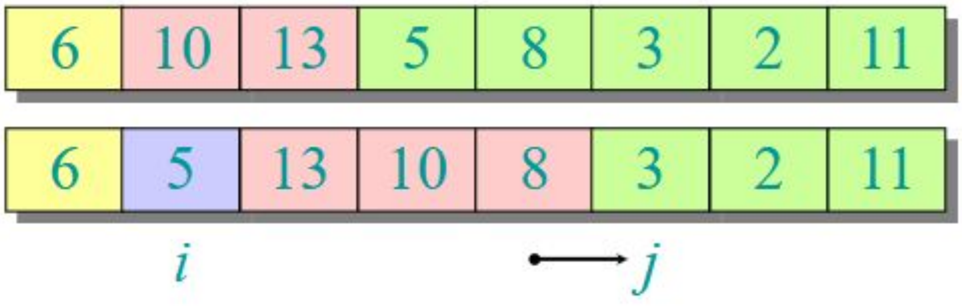
Courtesy : [Cormen et. al]()

Quick Sort - Example of Partitioning



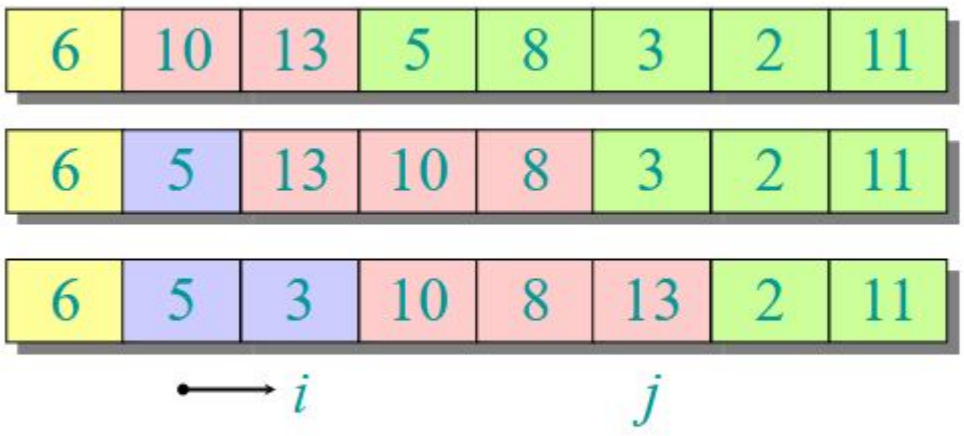
Courtesy : [Cormen et. al]

Quick Sort - Example of Partitioning



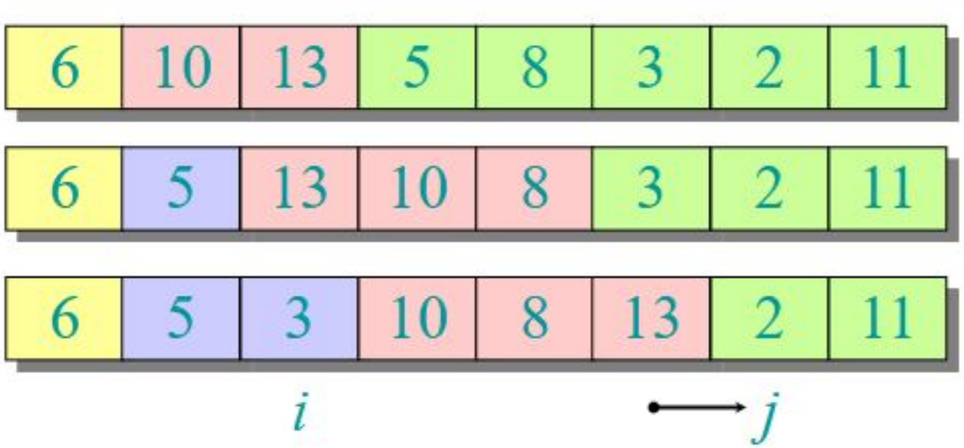
Courtesy : [Cormen et. al]()

Quick Sort - Example of Partitioning



Courtesy : [Cormen et. al]()

Quick Sort - Example of Partitioning



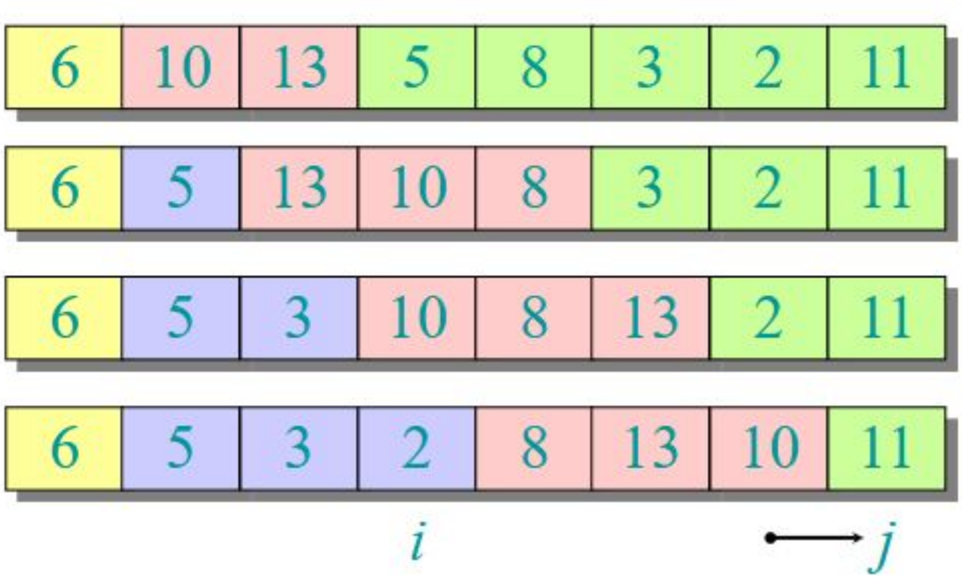
Courtesy : [Cormen et. al]()

Quick Sort - Example of Partitioning

6	10	13	5	8	3	2	11
6	5	13	10	8	3	2	11
6	5	3	10	8	13	2	11
6	5	3	2	8	13	10	11
			i			j	

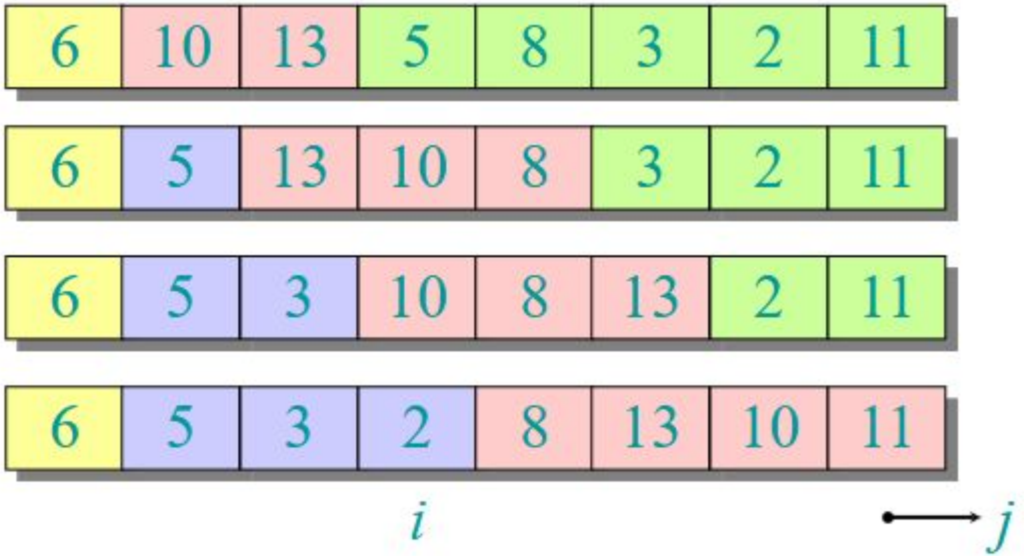
Courtesy : [Cormen et. al]

Quick Sort - Example of Partitioning



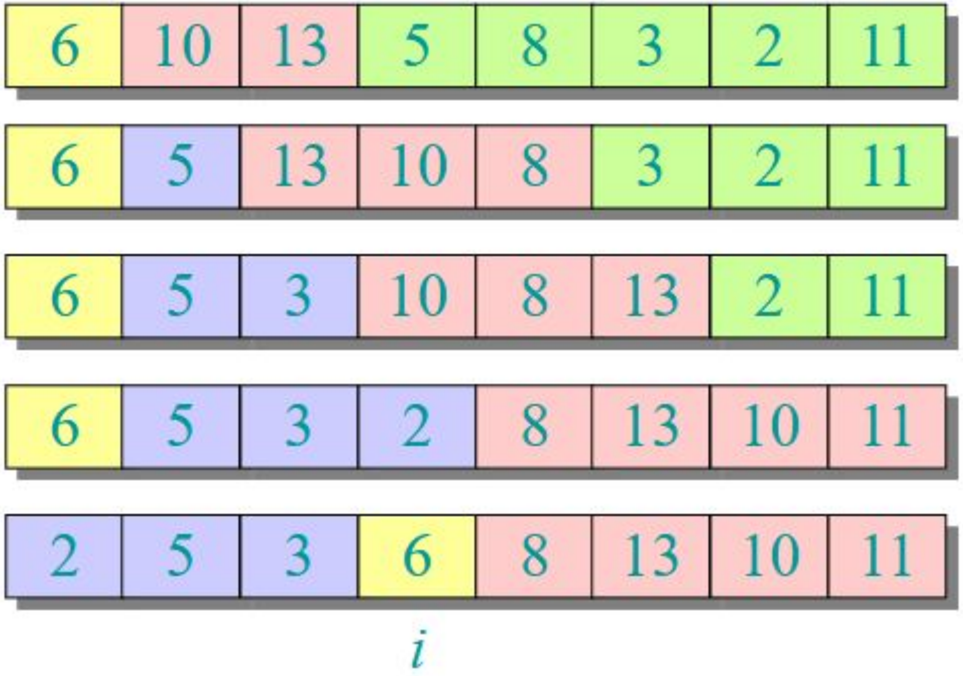
Courtesy : [Cormen et. al]()

Quick Sort - Example of Partitioning



Courtesy : [Cormen et. al]

Quick Sort - Example of Partitioning



Courtesy : [Cormen et. al]

Quick Sort - Algorithm

```
QUICKSORT(A, p, r)
    if p < r
        then q ← PARTITION(A, p, r)
            QUICKSORT(A, p, q–1)
            QUICKSORT(A, q+1, r)
```

Initial call: QUICKSORT(A, 1, n)

Courtesy : [Cormen et. al]

Quick Sort Analysis (Worst Case)

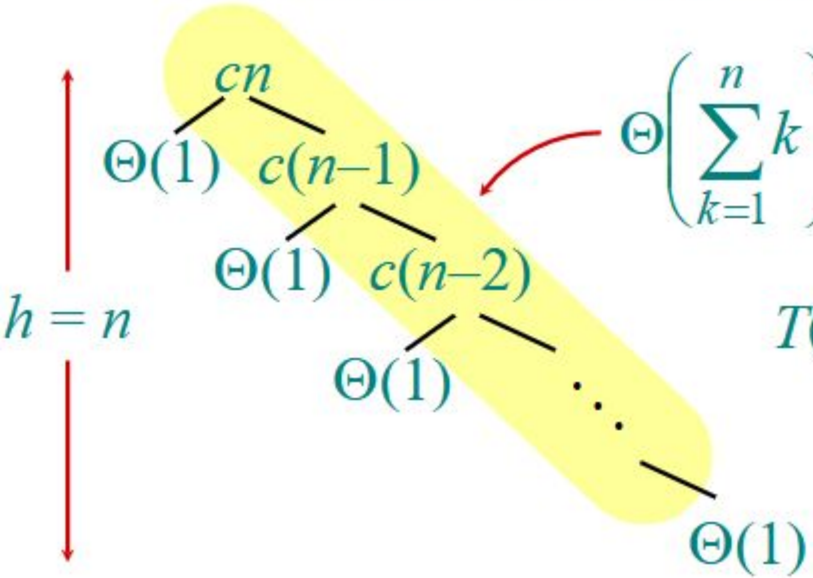
- Input sorted or reverse sorted.
- Partition around min or max element.
- One side of partition always has no elements.

$$
\begin{aligned}
T(n) &= T(0) + T(n-1) + \Theta(n) \\
&= \Theta(1) + T(n-1) + \Theta(n) \\
&= T(n-1) + \Theta(n) \\
&= \Theta(n^2) \qquad \textit{(arithmetic series)}
\end{aligned}
$$

Courtesy : [Cormen et. al]

Quick Sort Analysis (Worst Case Recursion Tree)

$$T(n) = T(0) + T(n-1) + cn$$



$$\Theta\left(\sum_{k=1}^{n} k\right) = \Theta(n^2)$$

$$T(n) = \Theta(n) + \Theta(n^2)$$
$$= \Theta(n^2)$$

Courtesy : [Cormen et. al]

Quick Sort Analysis (Best Case)

If we're lucky, PARTITION splits the array evenly:

$$\begin{aligned} T(n) &= 2T(n/2) + \Theta(n) \\ &= \Theta(n \lg n) \qquad \text{(same as merge sort)} \end{aligned}$$

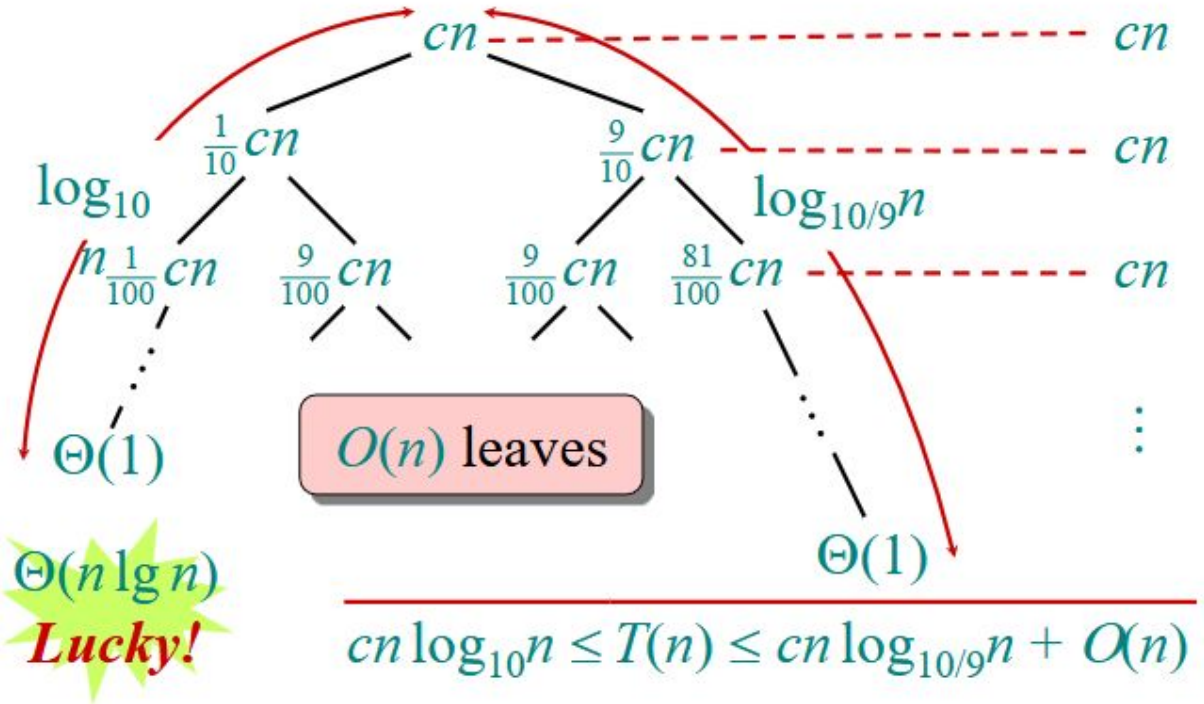
What if the split is always $\frac{1}{10} : \frac{9}{10}$?

$$T(n) = T\left(\tfrac{1}{10}n\right) + T\left(\tfrac{9}{10}n\right) + \Theta(n)$$

What is the solution to this recurrence?

Courtesy : Cormen et. al

Quick Sort Analysis (Almost Best Case)



Courtesy : [Cormen et. al]

Quick Sort Analysis - More Intuition

Suppose we alternate lucky, unlucky, lucky, unlucky, lucky, ….

$$L(n) = 2U(n/2) + \Theta(n) \quad \textit{lucky}$$

$$U(n) = L(n-1) + \Theta(n) \quad \textit{unlucky}$$

Solving:

$$\begin{aligned} L(n) &= 2(L(n/2-1) + \Theta(n/2)) + \Theta(n) \\ &= 2L(n/2-1) + \Theta(n) \\ &= \Theta(n \lg n) \quad \textit{Lucky!} \end{aligned}$$

How can we make sure we are usually lucky? → **Randomized Quick Sort !!**

Courtesy : Cormen et. al

Quick Sort - Summary

- Quicksort is a great general-purpose sorting algorithm.
- Quicksort is typically over twice as fast as merge sort.
- Quicksort can benefit substantially from *code tuning*.
- Quicksort behaves well even with caching and virtual memory.

Courtesy : Cormen et. al

Topics to be covered

- General method
- Min-Max Algorithm
- Merge sort
- Quick sort
- **Analysis of Binary search**
- Strassen's Matrix Multiplication.

Analysis of Binary Search

Find an element in a sorted array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.

Example: Find 9

3	5	7	8	9	12	15

Courtesy : Cormen et. al

Analysis of Binary Search

Find an element in a sorted array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.

Example: Find 9

3	5	7	8	9	12	15

Courtesy : Cormen et. al

Analysis of Binary Search

Find an element in a sorted array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.

Example: Find 9

3 5 7 8 9 12 15

Courtesy : Cormen et. al

Analysis of Binary Search

$$T(n) = 1\,T(n/2) + \Theta(1)$$

subproblems

subproblem size

work dividing and combining

$$n^{\log_b a} = n^{\log_2 1} = n^0 = 1 \Rightarrow \text{CASE 2 } (k = 0)$$
$$\Rightarrow T(n) = \Theta(\lg n)\,.$$

Courtesy : Cormen et. al

Topics to be covered

- General method
- Min-Max Algorithm
- Merge sort
- Quick sort
- Analysis of Binary search
- **Strassen's Matrix Multiplication.**

Matrix Multiplication

Input: $A = [a_{ij}], B = [b_{ij}].$
Output: $C = [c_{ij}] = A \cdot B.$

$$\left.\begin{array}{l} A = [a_{ij}], B = [b_{ij}]. \\ C = [c_{ij}] = A \cdot B. \end{array}\right\} \quad i, j = 1, 2, \ldots, n.$$

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

Courtesy : Cormen et. al

Matrix Multiplication

```
SQUARE-MATRIX-MULTIPLY(A, B)
n = A.rows
let C be a new n × n matrix
for i = 1 to n
        for j = 1 to n
                c_ij = 0
                for k = 1 to n
                        c_ij = c_ij + a_ik · b_kj
return C
```

$T(n) = \Theta\ (n^3)$

Courtesy : [Cormen et. al]()

Matrix Multiplication - Divide & Conquer Method

IDEA:

$n \times n$ matrix = 2×2 matrix of $(n/2) \times (n/2)$ submatrices:

$$\left[\begin{array}{c|c} r & s \\ \hline t & u \end{array}\right] = \left[\begin{array}{c|c} a & b \\ \hline c & d \end{array}\right] \cdot \left[\begin{array}{c|c} e & f \\ \hline g & h \end{array}\right]$$

$$C \quad = \quad A \quad \cdot \quad B$$

$$\left.\begin{aligned} r &= ae + bg \\ s &= af + bh \\ t &= ce + dg \\ u &= cf + dh \end{aligned}\right\} \quad \begin{array}{l} 8 \text{ mults of } (n/2)\times(n/2) \text{ submatrices} \\ 4 \text{ adds of } (n/2)\times(n/2) \text{ submatrices} \end{array}$$

Courtesy : Cormen et. al

Matrix Multiplication - Divide & Conquer Method

SQUARE-MATRIX-MULTIPLY-RECURSIVE(A, B)

```
n = A.rows
let C be a new n × n matrix
if n == 1
    c11 = a11 · b11
else partition A, B, and C as in equations (4.9)
    C11 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A11, B11)
          + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A12, B21)
    C12 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A11, B12)
          + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A12, B22)
    C21 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A21, B11)
          + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A22, B21)
    C22 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A21, B12)
          + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A22, B22)
return C
```

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ 8T(n/2) + \Theta(n^2) & \text{if } n > 1. \end{cases}$$

$$T(n) = \Theta(n^3)$$

Courtesy : Cormen et. al

Strassen’s Matrix Multiplication (Simplified)

1. Given two square matrices A and B of size $n \times n$, divide them into four equal-sized submatrices:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

2. Compute **seven** matrix products (instead of eight in traditional multiplication):

$$M_1 = (A_{11} + A_{22})(B_{11} + B_{22})$$

$$M_2 = (A_{21} + A_{22})B_{11}$$

$$M_3 = A_{11}(B_{12} - B_{22})$$

$$M_4 = A_{22}(B_{21} - B_{11})$$

$$M_5 = (A_{11} + A_{12})B_{22}$$

$$M_6 = (A_{21} - A_{11})(B_{11} + B_{12})$$

$$M_7 = (A_{12} - A_{22})(B_{21} + B_{22})$$

Courtesy : Cormen et. al

Strassen's Matrix Multiplication (Simplified)

3. Compute the resultant submatrices:

$$C_{11} = M_1 + M_4 - M_5 + M_7$$

$$C_{12} = M_3 + M_5$$

$$C_{21} = M_2 + M_4$$

$$C_{22} = M_1 - M_2 + M_3 + M_6$$

4. Combine these submatrices to get the final result.

$$\boxed{T(n) = \begin{cases} \Theta(1) & \text{if } n = 1\,, \\ 7T(n/2) + \Theta(n^2) & \text{if } n > 1\,. \end{cases}}$$

$$\boxed{T(n) = 7T(n/2) + O(n^2)}$$

(Master's theorem)

$$\boxed{T(n) = O(n^{\log_2 7}) \approx O(n^{2.81})}$$

Courtesy : Cormen et. al

Strassen's Matrix Multiplication (Simplified)

Compute : $\begin{pmatrix} 1 & 3 \\ 7 & 5 \end{pmatrix}\begin{pmatrix} 6 & 8 \\ 4 & 2 \end{pmatrix}$

Step 1: Compute Strassen's 7 Products

1. $M_1 = (A_{11} + A_{22})(B_{11} + B_{22})$

$$M_1 = (1 + 5) \times (6 + 2) = 6 \times 8 = 48$$

2. $M_2 = (A_{21} + A_{22})B_{11}$

$$M_2 = (7 + 5) \times 6 = 12 \times 6 = 72$$

3. $M_3 = A_{11}(B_{12} - B_{22})$

$$M_3 = 1 \times (8 - 2) = 1 \times 6 = 6$$

4. $M_4 = A_{22}(B_{21} - B_{11})$

$$M_4 = 5 \times (4 - 6) = 5 \times (-2) = -10$$

5. $M_5 = (A_{11} + A_{12})B_{22}$

$$M_5 = (1 + 3) \times 2 = 4 \times 2 = 8$$

6. $M_6 = (A_{21} - A_{11})(B_{11} + B_{12})$

$$M_6 = (7 - 1) \times (6 + 8) = 6 \times 14 = 84$$

7. $M_7 = (A_{12} - A_{22})(B_{21} + B_{22})$

$$M_7 = (3 - 5) \times (4 + 2) = (-2) \times 6 = -12$$

Strassen's Matrix Multiplication (Simplified)

Compute : $\begin{pmatrix} 1 & 3 \\ 7 & 5 \end{pmatrix}\begin{pmatrix} 6 & 8 \\ 4 & 2 \end{pmatrix}$

Step 2: Compute the Resultant Matrix C

$$C_{11} = M_1 + M_4 - M_5 + M_7 = 48 + (-10) - 8 + (-12) = 18$$

$$C_{12} = M_3 + M_5 = 6 + 8 = 14$$

$$C_{21} = M_2 + M_4 = 72 + (-10) = 62$$

$$C_{22} = M_1 - M_2 + M_3 + M_6 = 48 - 72 + 6 + 84 = 66$$

Final Answer:

$$C = \begin{bmatrix} 18 & 14 \\ 62 & 66 \end{bmatrix}$$