Divide and Conquer Algorithm

Binary Search

Quick Sort

Merge Sort

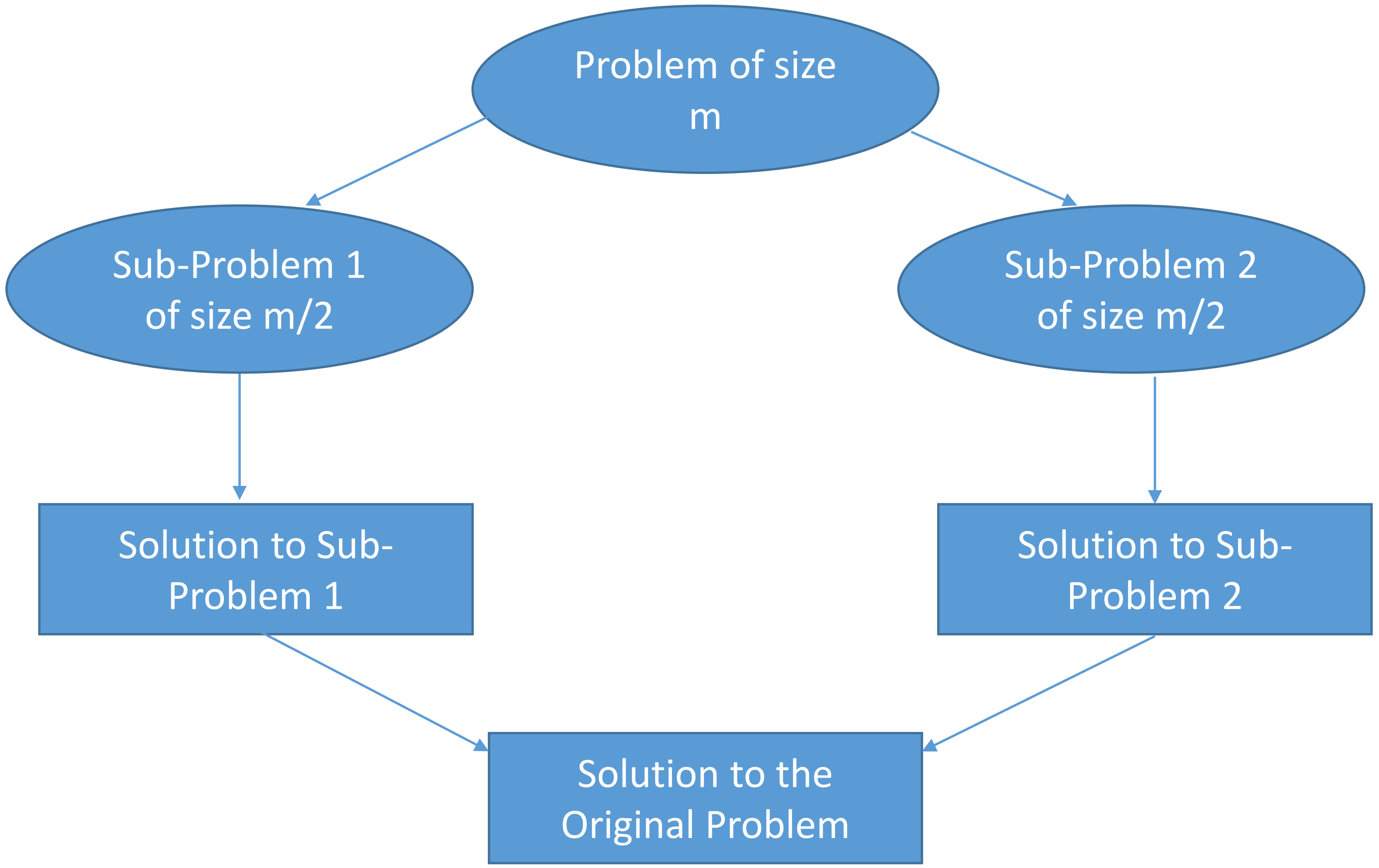
Strassen’s Matrix

Approach

- **Divide the problem** into a number of sub-problems

  Sub-problems must be of same type

  Sub-problems do not need to overlap
- **Conquer the sub-problems** by solving them recursively. If they are small enough, solve them in brute force fashion.
- **Combine the solutions** to the sub-problems into the solution for the original problem.


Problem of size m
Sub-Problem 1 of size m/2
Sub-Problem 2 of size m/2
Solution to Sub-Problem 1
Solution to Sub-Problem 2
Solution to the Original Problem

Advantage of Divide and Conquer

- Divide and conquer approach supports parallelism as sub-problems are independent. Hence, an algorithm, which is designed using this technique, can run on the multiprocessor system or in different machines simultaneously.

Disadvantage of Divide and Conquer

- In this approach, most of the algorithms are designed using recursion, hence memory management is very high. For recursive function stack is used, where function state needs to be stored.

Application of Divide & Conquer

- Finding the maximum and minimum of a sequence of numbers
- Merge sort
- Binary search
- Quick Sort
- Strassen’s matrix multiplication

Divide & Conquer Algorithm

```
 D&C(P)
 {
     if (small (P))
      {
      Solve(P)
      }
    else
       {
          Divide P into P1, P2, P3 ……,Pk                 ----------- > f1(n)
          Apply DAC(P1), DAC(P2),….., DAC(Pk)     ----------- > T(n/2)
          Combine (DAC(P1), DAC(P2), …., DAC(Pk)) -------- > f2(n)
       }
}
```

Recurrence Relation of DAC algorithm will be

$T(n)= f1(n)+aT(n/2)+f2(n)$

where, n is the size of input

a is number of the sub problems we break the problem into

n/2 is the size of the sub problems (in terms of n)

Binary Search

Introduction

- Binary search can be performed only on a sorted array and reduce the time complexity.
- So, the elements must be arranged in-

  -> Either ascending order if the elements are numbers.

  -> Dictionary order if the elements are strings.
- Approach:-

  (i) Compare **x** with the **middle** element.

  (ii) If x matches with middle element, we return the **mid index**.

  (iii) Else If x is greater than the mid element, then x can only lie in
  right half subarray after the mid element. So we **recur** for **right half**.

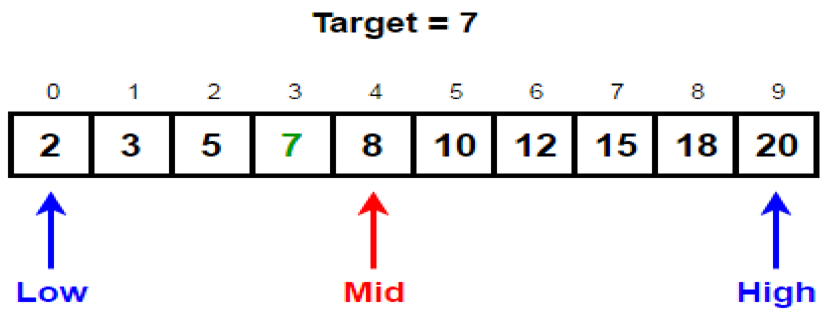
  (iv) Else (x is smaller) **recur** for the **left half.**

Binary Search Algorithm

```
int binarySearch(int arr[], int l, int r, int x)
{       if (r >= l)
        {
              int mid =  l + (r - l) / 2; // Floor Function
              if (arr[mid] == x)
              {
                 return mid; // Element found at middle
               }
              if (arr[mid] > x)
             {
                  return binarySearch(arr, l, mid - 1, x); // Element found at Left subarray
              }
             return binarySearch(arr, mid + 1, r, x); // Element found at Right subarray
        }
      return -1; // Element not found
}
```

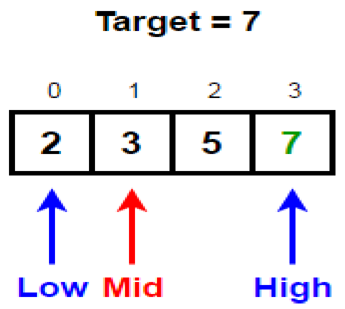
Using Binary searching search the Target element 7

0	1	2	3	4	5	6	7	8	9
2	3	5	7	8	10	12	15	18	20



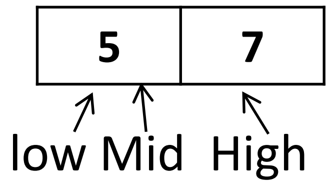
Since 8 (Mid) > 7 (target),
we discard the right half and go LEFT

New High = Mid - 1



Since 3 (Mid) < 7 (target),
We discard the left half and go RIGHT

New low = mid + 1



Since 5 (Mid)< 7 (Target),

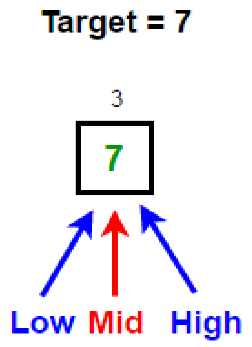
We discard the left half and go to right

Time Complexity

1^{st} Iteration ------- n
2^{nd} Iteration ------- $n/2$
3^{rd} Iteration ------- $n/2^2$
4^{th} Iteration ------- $n/2^3$

k Iteration -------- $n/2^k$
$n/2^k = 1$
$k=\log_2 n$

Time Complexity is $\mathbf{O(\log_2 n)}$



Now our search space consists of only one element 7. Since 7 (Mid) = 7 (target), we return index of 7 i.e. 3 and terminate our search

Quick Sort

Introduction

- Quick sort is an algorithm based on divide and conquer approach in which the array is split into sub arrays and these sub-arrays are recursively called to sort the elements.

  (I) Divide
  The array is divided into subparts taking pivot as the partitioning point. The elements smaller than the pivot are placed to the left of the pivot and the elements greater than the pivot are placed to the right.

  (II) Conquer
  The left and the right subparts are again partitioned using the by selecting pivot elements for them. This can be achieved by recursively passing the subparts into the algorithm.

  (III) Combine
  This step does not play a significant role in quick sort. The array is already sorted at the end of the conquer step.

```
QuickSort(a, low, upper)
{
 if(low<upper)
 {
   Pivot=Partition(a, low, upper, pivot); // Comparison & locate pivot at proper location
   QuickSort(a, low, Pivot-1);
   QuickSort(a, Pivot+1, upper);
 }
}
```

Time Complexity :-

n time taken by Partition() as it comparing elements and swapping them

$\log_2 n$ by recursive QuickSort().

$O(n\log_2 n)$ because every time Partition() will be called by QuickSort().

Note:- If we have sorted list and apply QuickSort() then time complexity will be **$O(n^2)$**

0	1	2	3	4	5	6	7
29	**23**	**17**	**57**	**34**	**89**	**65**	**27**

low=0 Pivot=low Upper=7

If pivot located at low position then Right to Left scanning & search smaller value than pivot. If found then Swap.
If pivot located at Upper then Left to Right scanning & search bigger value than pivot. If found then Swap.

0	1	2	3	4	5	6	7
27	**23**	**17**	**57**	**34**	**89**	**65**	**29**

low=0 Pivot=Upper Upper=7

0	1	2	3	4	5	6	7
27	**23**	**17**	**29**	**34**	**89**	**65**	**57**

low=3 Pivot=low Upper=7

0	1	2
27	**23**	**17**

3
29

4	5	6	7
34	**89**	**65**	**57**

return Pivot=3 and Call QS(a, low, pivot -1)

0	1	2
27	**23**	**17**

low =0 Pivot=low Upper=2

0	1	2
17	**23**	**27**

low =0 Pivot=Upper Upper=2

0	1
17	**23**

2
27

return Pivot=2 and Call QS(a, low, pivot -1)

17	**23**

0	1	2	3	4	5	6	7
17	**23**	**27**	**29**	**34**	**89**	**65**	**57**

low=4, Pivot=low, Upper=7

4	5	6	7
34	**89**	**65**	**57**

return Pivot=4 and Call QS(a, low, pivot-1)

5	6	7
89	**65**	**57**

low=5 Pivot=low Upper=7   R -> L

5	6	7
57	**65**	**89**

low=5 Pivot=Upper Upper=7   L -> R

return Pivot=7 and
Call QS(a, low, Pivot-1)

5	6	7
57	**65**	**89**

low=5 Pivot=low Upper=6   R -> L

5	6
57	**65**

57	**65**

We get Sorted list :-

17	**23**	**27**	**29**	**34**	**57**	**65**	**89**

Merge Sort

- **Merge sort** is a sorting technique based on divide and conquer technique.
- Idea :-

  -> Divide: Split Array of size n down the middle into two subsequences,

  each of size n/2 each subsist consists of a single element.

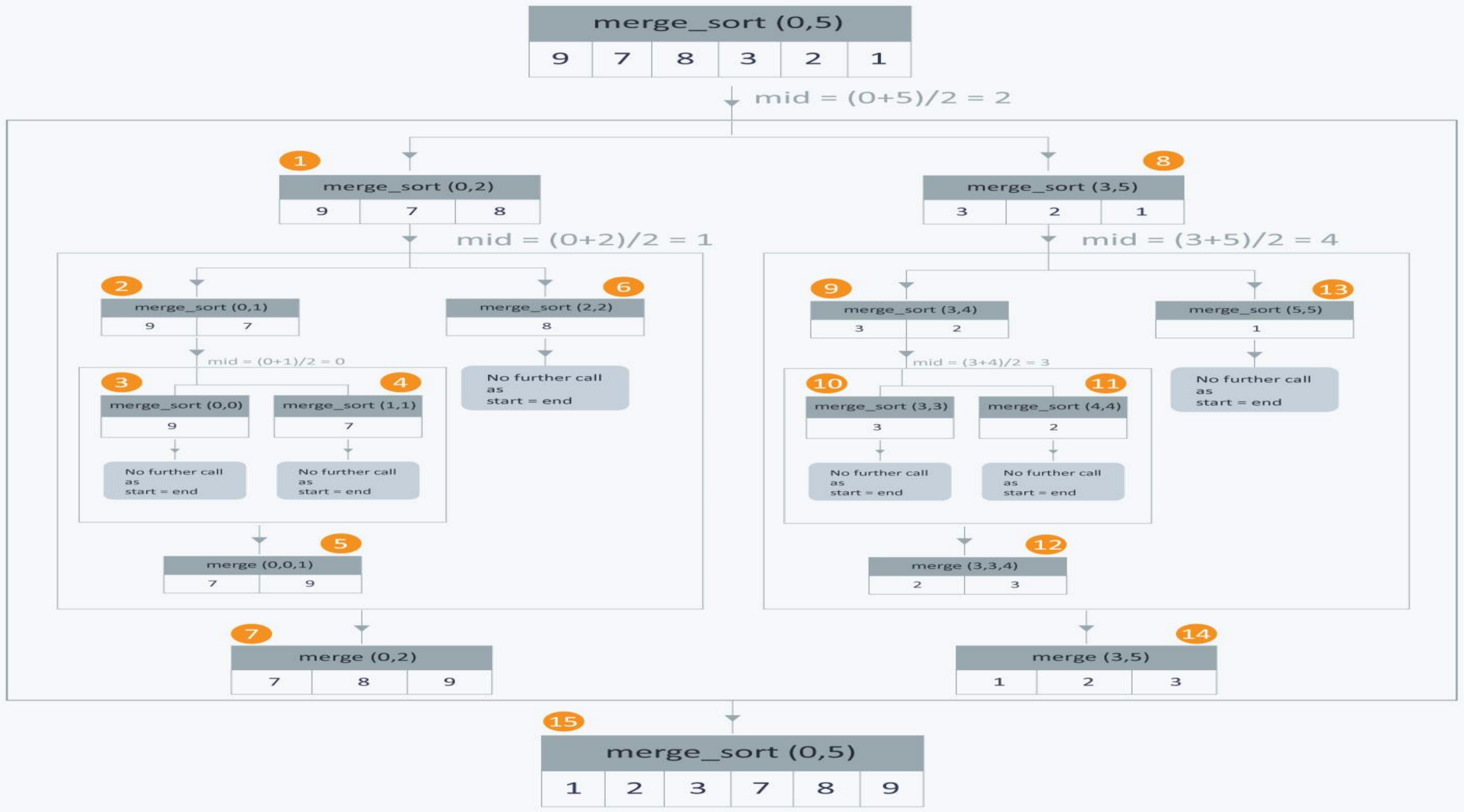
  -> Conquer: Sort each subsequence (by calling Merge Sort recursively on each).

  -> **Combine:** Merge the two sorted subsequences into a single sorted list.

Algorithm

```
mergeSort(arr[], l, r)
If (r > l)
1.   Find the middle point to divide the array into two halves:
             middle m = (l+r)/2
2.   Call mergeSort for first half:
             Call mergeSort(arr, l, m)
3.   Call mergeSort for second half:
             Call mergeSort(arr, m+1, r)
4.   Merge the two halves sorted in step 2 and 3:
             Call merge(arr, l, m, r)
```

Merge Sort


merge_sort (0,5)
9 7 8 3 2 1
mid = (0+5)/2 = 2
1
merge_sort (0,2)
9 7 8
mid = (0+2)/2 = 1
2
merge_sort (0,1)
9 7
mid = (0+1)/2 = 0
3
merge_sort (0,0)
9
No further call as start = end
4
merge_sort (1,1)
7
No further call as start = end
5
merge (0,0,1)
7 9
6
merge_sort (2,2)
8
No further call as start = end
7
merge (0,2)
7 8 9
8
merge_sort (3,5)
3 2 1
mid = (3+5)/2 = 4
9
merge_sort (3,4)
3 2
mid = (3+4)/2 = 3
10
merge_sort (3,3)
3
No further call as start = end
11
merge_sort (4,4)
2
No further call as start = end
12
merge (3,3,4)
2 3
13
merge_sort (5,5)
1
No further call as start = end
14
merge (3,5)
1 2 3
15
merge_sort (0,5)
1 2 3 7 8 9

The **performance** of **merge sort** is then given by this **recurrence relation**

$T(N) = 2*T(N/2) + N$ ...... (1)

$T(1) = 1$ ...... (2)

We can use the **expansion method** to find a solution:

$$
\begin{aligned}
T(N) &= 2*T(N/2) + N \\
&= 2*(2*T(N/4) + N/2) + N \\
&= 4*T(N/4) + N + N \\
&= 4*T(N/4) + 2*N
\end{aligned}
$$

$$
\begin{aligned}
T(N) &= 4*T(N/4) + 2*N \\
&= 4*(2*T(N/8) + N/4) + 2*N \\
&= 8*T(N/8) + N + 2*N \\
&= 8*T(N/8) + 3*N
\end{aligned}
$$

And so on....

In general:

$T(N) = 2^k * T(N/(2^k)) + k*N$

If $N = 2^k$ $T(N)$

$= 2*T(N/2) + N$

$= 4*T(N/4) + 2*N$

$= 8*T(N/8) + 3*N$

$= 16*T(N/16) + 4*N$

$= 32*T(N/32) + 5*N$

$= \ldots$

$=(2^k)*T(N/(2^k)) + k*N$ w.k.t $(N = 2^k$, so: $N/(2^k) = 1)$

$=(2^k)*T(1) + k*N$ $(2^k= N)$

$=N*T(1) + k*N$ $(T(1) = 1)$

$= N + k*N$

$= (k+1)*N$ $(N = 2^k \Rightarrow k = \lg(N))$

$= (\lg(N)+1)*N$

The **running time** of **merge sort** to **sort** an **array of size *N*** is **bounded by**

***(log(N)+1)* × *N* = O(*N*×log(*N*))**

Applications of Merge Sort

1) Merge Sort for Linked Lists

https://www.youtube.com/watch?v=mM0EVkKraVw

2) Count Inversions in an array

https://www.youtube.com/watch?v=k9RQh21KrH8

3) External Sort

https://www.youtube.com/watch?v=Bp7fGofslng

Merge Sort v/s Quick Sort

- https://www.youtube.com/watch?v=Jjrk4OqZ2VY

Strassen’s Matrix

Divide and Conquer

- The Naive method to perform **Matrix Multiplication** of square matrices A and B of size n x n each

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

- We may apply the definition of block-matrix multiplication to write down a formula for the block-entries of C, i.e.

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

8 Multiplications
4 Additions

- Addition of two matrices takes $O(N^2)$ time.

  So, the time complexity can be written as

$$T(N) = 8T(N/2) + O(N^2)$$

- From Master's Theorem, time complexity of above method is $\mathbf{O(N^3)}$

Pseudocode for Matrix Multiplication

```
void multiply(int A[][N], int B[][N], int C[][N])
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            C[i][j] = 0;
            for (int k = 0; k < N; k++)
            {
                C[i][j] += A[i][k]*B[k][j];
            }
        }
    }
}
```

Time Complexity
$O(N^3)$

Strassen’s Matrix Multiplication

- Divide and Conquer technique
- The overall complexity for multiplication two matrices is reduced by decreasing the total number of multiplication performed at the expenses of a slight increase in the number of addition.
- Strassen’s Matrix multiplication can be performed only on **square matrices** where **n** is a **power of 2** and if n is not power of 2 then matrices can be padded with zeros.
- Order of both of the matrices are **n × n**.
- Break up each matrix into four N/2 x N/2 submatrices.
- Recursively multiply the parts

$$\begin{bmatrix} A11 & A12 \\ A21 & A22 \end{bmatrix} \text{X} \begin{bmatrix} B11 & B12 \\ B21 & B22 \end{bmatrix} = \begin{bmatrix} C11 & C12 \\ C21 & C22 \end{bmatrix}$$

The four sub-matrices of result are calculated using following formula:-

P1 = A11(B12 − B22)

P2 = (A11 + A12) B22

P3 = (A21 + A22) B11

P4 = A22(B21 − B11)

P5 = (A11 + A22) (B11 + B22)

P6 = (A12 − A22) (B21 + B22)

P7 = (A11 − A21) (B11 + B12)

Computing the values of the 4 quadrants of the final matrix C :-

C11 = P1 + P4 - P5 + P7
C12 = P3 + P5
C21 = P2 + P4
C22 = P1 + P3 - P2 + P6

Example: Perform Matrix Multiplication using Strassen's Matrix

$$\mathbf{A} = \begin{pmatrix} 5 & 2 & 6 & 1 \\ 0 & 6 & 2 & 0 \\ 3 & 8 & 1 & 4 \\ 1 & 8 & 5 & 6 \end{pmatrix} \qquad \mathbf{B} = \begin{pmatrix} 7 & 5 & 8 & 0 \\ 1 & 8 & 2 & 6 \\ 9 & 4 & 3 & 8 \\ 5 & 3 & 7 & 9 \end{pmatrix}$$

P1= A_{11} $(B_{12} - B_{22})$

$$= \begin{pmatrix} 5 & 2 \\ 0 & 6 \end{pmatrix} * \left(\begin{pmatrix} 8 & 0 \\ 2 & 6 \end{pmatrix} - \begin{pmatrix} 3 & 8 \\ 7 & 9 \end{pmatrix} \right) = \begin{pmatrix} 15 & -46 \\ -30 & -18 \end{pmatrix}$$

P2= $(A_{11} + A_{12})$ B_{22}

$$= \left(\begin{pmatrix} 5 & 2 \\ 0 & 6 \end{pmatrix} + \begin{pmatrix} 6 & 1 \\ 2 & 0 \end{pmatrix} \right) * \begin{pmatrix} 3 & 8 \\ 7 & 9 \end{pmatrix} = \begin{pmatrix} 54 & 115 \\ 48 & 70 \end{pmatrix}$$

P3= $(A_{21} + A_{22})$ B_{11}

$$= \left(\begin{pmatrix} 3 & 8 \\ 1 & 8 \end{pmatrix} + \begin{pmatrix} 1 & 4 \\ 5 & 6 \end{pmatrix} \right) * \begin{pmatrix} 7 & 5 \\ 1 & 8 \end{pmatrix} = \begin{pmatrix} 40 & 116 \\ 56 & 142 \end{pmatrix}$$

$P4 = A_{22}(B_{21} - B_{11}) = \begin{pmatrix} 18 & -21 \\ 34 & -35 \end{pmatrix}$

$P5 = (A_{11} + A_{22})(B_{11} + B_{22}) = \begin{pmatrix} 108 & 180 \\ 146 & 269 \end{pmatrix}$

$P6 = (A_{12} - A_{22})(B_{21} + B_{22}) = \begin{pmatrix} 24 & 24 \\ -108 & -108 \end{pmatrix}$

$P7 = (A_{12} - A_{22})(B_{21} + B_{22}) = \begin{pmatrix} 96 & 68 \\ 24 & 56 \end{pmatrix}$

$C11 = P5+P4-P2+P6 = \begin{pmatrix} 108 & 180 \\ 146 & 269 \end{pmatrix} + \begin{pmatrix} 18 & -21 \\ 34 & -35 \end{pmatrix} - \begin{pmatrix} 54 & 115 \\ 48 & 70 \end{pmatrix} + \begin{pmatrix} 24 & 24 \\ -108 & -108 \end{pmatrix} = \begin{pmatrix} 96 & 68 \\ 24 & 56 \end{pmatrix}$

$C12 = P1+P2$

$C21 = P3+P4$

$C22 = P1+P5-P3-P7$

$$\mathbf{A * B} = \begin{pmatrix} \mathbf{96} & \mathbf{68} & \mathbf{69} & \mathbf{69} \\ \mathbf{24} & \mathbf{56} & \mathbf{18} & \mathbf{52} \\ \mathbf{58} & \mathbf{95} & \mathbf{71} & \mathbf{92} \\ \mathbf{90} & \mathbf{107} & \mathbf{81} & \mathbf{142} \end{pmatrix}$$

Comparison of Naïve and Strassen’s Matrix

```
MMult(A, B, n)
 If n = 1
  Output A × B
 Else
   Compute A11, B11, . . ., A22, B22 % by computing m = n/2
   X1 ← MMult(A11, B11, n/2)
   X2 ← MMult(A12, B21, n/2)
   X3 ← MMult(A11, B12, n/2)
   X4 ← MMult(A12, B22, n/2)
   X5 ← MMult(A21, B11, n/2)
   X6 ← MMult(A22, B21, n/2)
   X7 ← MMult(A21, B12, n/2)
   X8 ← MMult(A22, B22, n/2)
   C 11 ← X1 + X2
   C 12 ← X3 + X4
   C 21 ← X5 + X6
   C 22 ← X7 + X8
   Output C
End If
```

$$\boxed{T(n) = 8T(n/2) + \Theta(n^2)}$$

```
Strassen(A, B)
If n = 1
   Output A × B 2
Else
  Compute A11, B11, . . ., A22, B22 %  by computing m = n/2
   P1 ← Strassen(A11, B12 − B22)
   P2 ← Strassen(A11 + A12, B22)
   P3 ← Strassen(A21 + A22, B11)
   P4 ← Strassen(A22, B21 − B11)
   P5 ← Strassen(A11 + A22, B11 + B22)
   P6 ← Strassen(A12 − A22, B21 + B22)
   P7 ← Strassen(A11 − A21, B11 + B12)
   C 11 ← P5 + P4 − P2 + P6
   C 12 ← P1 + P2
   C 21 ← P3 + P4
   C 22 ← P1 + P5 − P3 − P7
   Output C
 End If
```

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