# Divide and Conquer.

## The Control Abstraction

→ Divide and Conquer

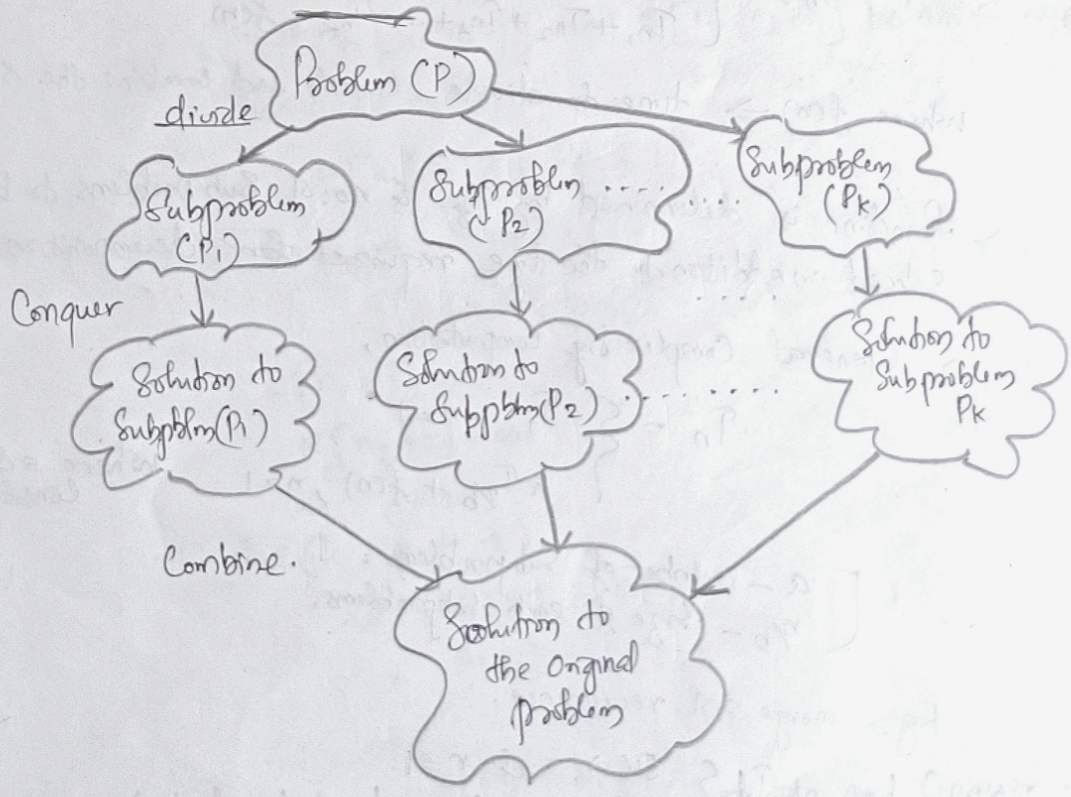
→ It is a top-down approach for designing algorithms that consists of dividing the problem into smaller subproblems hoping that the solution of subproblems are easier to find. Then composing the partial solutions into the solution of the original problem.

→ 3 parts.

1) Divide :- Dividing the problem into no: of subproblems that are smaller instances of same problem.

2) Conquer :- Conquer the subproblems by solving them recursively. If they are small enough solve the subproblems as base cases.

3) Combine :- Combining the solutions of subproblems into solution of the original problem.



Control abstraction for recursive divide and conquer method with problem instance P.

```
divide_and_conquer(P)
{
    if Small(P) // P is very small, soln is trivial.
        return solution(S);
    else
        divide the problem P into k instances P1, P2, ... Pk
        return (Combine (divide_and_Conquer(P1),
                         divide_and_conquer(P2),
                         :
                         divide_and_conquer(Pk)))
}
```

The solution of the above problem is described by recurrence, assuming size of P denoted by n.

$$T_n = \begin{cases} g(n) \\ T_{n_1} + T_{n_2} + T_{n_3} + \dots T_{n_k} + f(n) \end{cases}$$

where $f(n) \Rightarrow$ time to divide elements and combine the solutions.

→ Runtime is determined by size & no: of subproblems to be solved in addition to the time required for decomposition.

→ General Complexity computation,

$$T_n = \begin{cases} T_1 , & n=1 \\ a T_{n/b} + f(n) , & n>1 \end{cases}$$

where a & b are constants.

[ a - number of subproblems.
  n/b - size of each subproblems.

Eg:- merge sort recurrence.

$$T_n = \begin{cases} \Theta(1), & \text{if } n=1 \\ 2T_{n/2} + \Theta(n), & \text{if } n>1 \end{cases}$$

Solution for merge sort recurrence $= \Theta(n \log n)$.

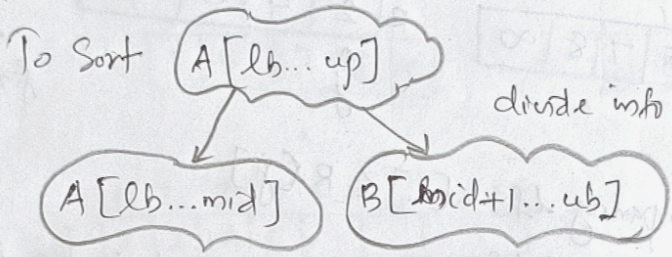
# MERGE SORT.



→ Uses divide and conquer strategy

→ The basic idea is to divide the list into sublists, sort each sublist and finally merge them to get a single sorted list.

The recursive implementation of 2-way merge sort divides the array into 2 sorts of sublists then merges them to get a single sorted list.




$mid = \lfloor \frac{lb+ub}{2} \rfloor$

## Algorithm for merging.

```
MERGE (A, P, q, r)                    P = lb
{ n1 = q - P + 1                      q = mid
  n2 = r - q                          r = ub.
  Let L[1... n1+1] and R[1... n2+1] be new arrays.
  for (i = 1 to n1)
      L[i] = A[P + i - 1]
  for (j = 1 to n2)
      R[j] = A[q + j]
  L[n1 + 1] = ∞
  R[n2 + 1] = ∞
  i = 1, j = 1
  for (k = p do r)
      if (L[i] ≤ R[j])
          A[k] = L[i];
          i = i + 1.
      else
          A[k] = R[j]
          j = j + 1.
}.
```

Eg:- A

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| 1 | 5 | 7 | 8 | 2 | 4 | 6 | 9 |

Here left half of the array is sorted and the right half of the array is also sorted. ie, the merging algorithm works on 2 sorted arrays.

→ Firstly, it copies the two sorted halves into arrays L & R. (adding ∞ at the end)

L

| 1 | 5 | 7 | 8 | ∞ |
|---|---|---|---|---|

↑ i

R

| 2 | 4 | 6 | 9 | ∞ |
|---|---|---|---|---|

↑ j

i & j pointing to first elements of L & R respectively.

→ Comparing if $L[i] \leq R[j]$

ie, $1 < 2$.

if so $L[i]$, ie, 1 is copied to the initial array A

A

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| 1 | | | | | | | |

then, increment i, ie, $i = i+1$

L

| 1 | 5 | 7 | 8 | ∞ |
|---|---|---|---|---|

↑ i

R

| 2 | 4 | 6 | 9 | ∞ |
|---|---|---|---|---|

↑ j

→ (if $L[i] \leq R[j]$ is false.

R[j] is copied to the initial array and j is incremented.

→ Compare $5 \leq 2$ → False.

∴ $j = j+1$ and 2 is copied to A.

L

| 1 | 5 | 7 | 8 | ∞ |
|---|---|---|---|---|

↑ i

R

| 2 | 4 | 6 | 9 | ∞ |
|---|---|---|---|---|

↑ j

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| 1 | 2 | | | | | | |

→ Is $5 \leq 4$ ; false ∴ 4 is copied & j++.

∴ L | 1 | 5 | 7 | 8 | ∞ |  R | 2 | 4 | 6 | 9 | ∞ |

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| A | 1 | 2 | 4 | | | | | |

→ is $5 \leq 6$ ; True . ∴ 5 is copied and i++

∴ L | 1 | 5 | 7 | 8 | ∞ |  R | 2 | 4 | 6 | 9 | ∞ |

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| A | 1 | 2 | 4 | 5 | | | | |

Continue these steps until the final element is compared and copied.

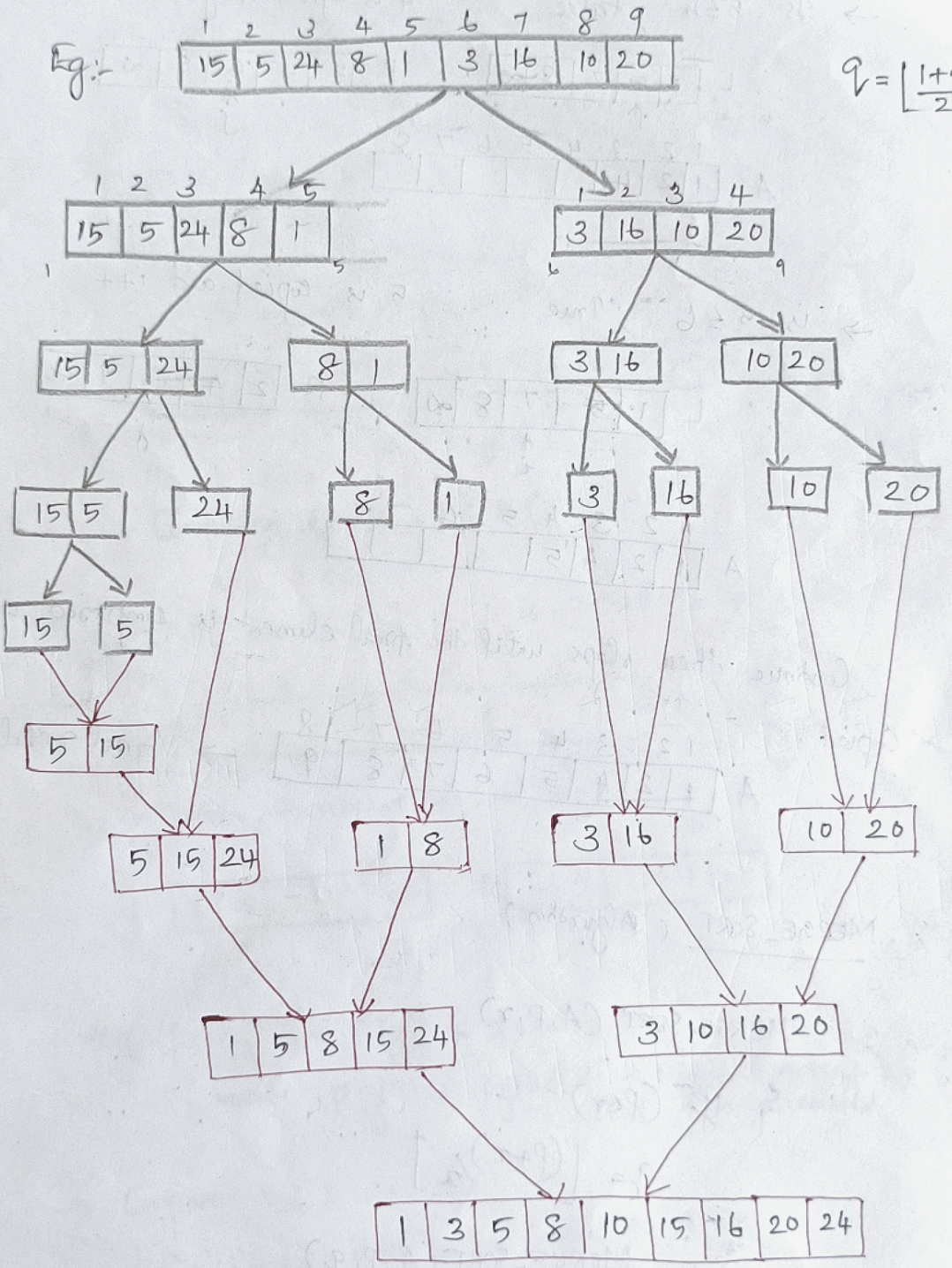
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| A | 1 | 2 | 4 | 5 | 6 | 7 | 8 | 9 |

→ Final result.

MERGE_SORT (Algorithm)

```
MERGE-SORT (A, P, r)
{ if (P<r)
    q = ⌊(P+r)/2⌋
      MERGE_SORT (A, P, q)
      MERGE-SORT (A, q+1, r)
      MERGE (A, P, q, r)
}.
```

Eg:-


1 2 3 4 5 6 7 8 9
15 5 24 8 1 3 16 10 20
q = ⌊(1+9)/2⌋ = 10/2 = 5
1 2 3 4 5
15 5 24 8 1
1 5
1 2 3 4
3 16 10 20
6 9
15 5 24
8 1
3 16
10 20
15 5
24
8
1
3
16
10
20
15
5
5 15
5 15 24
1 8
3 16
10 20
1 5 8 15 24
3 10 16 20
1 3 5 8 10 15 16 20 24

# QUICK SORT

→ In-place comparison, not stable, fast.
→ Use divide and conquer strategy
→ Complexity $O(n^2)$ — Worst Case
$O(n \log n)$ - Best case & average case.

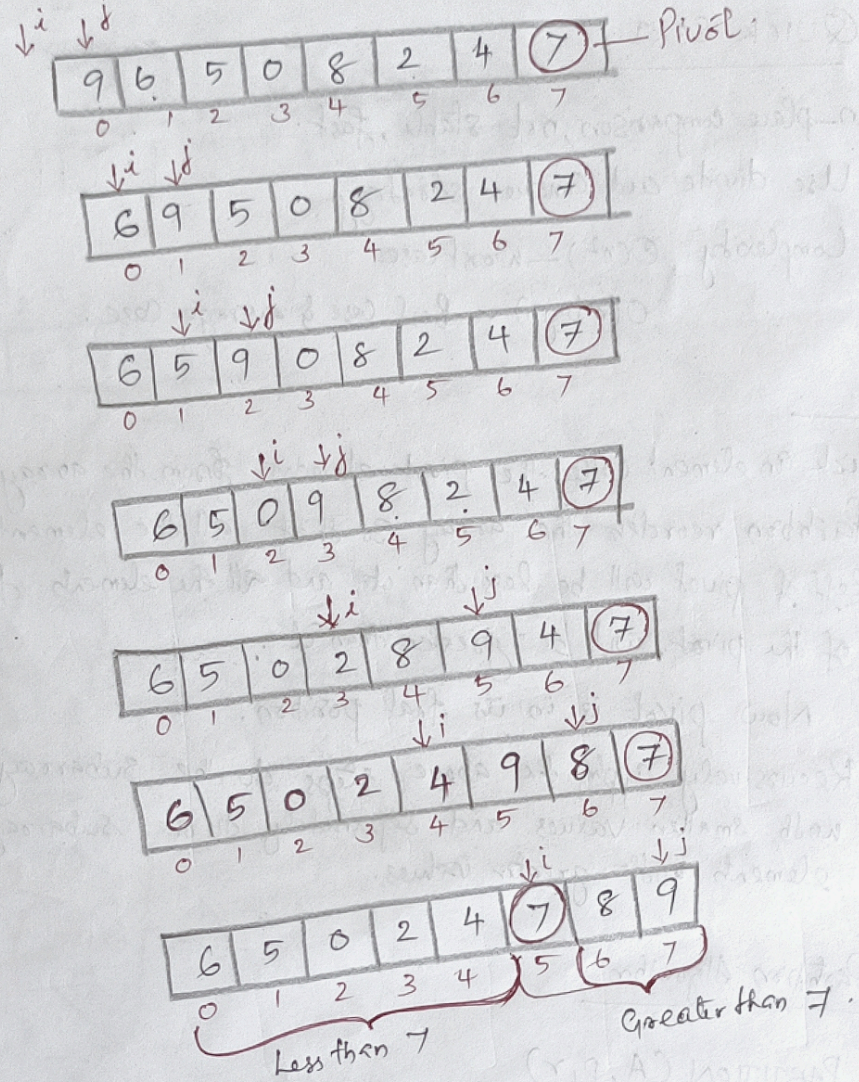
## Steps.

→ Pick an element called the pivot element from the array.
→ Partition reorder the array so that all the elements to the left of pivot will be less than it and all the elements to the right of the pivot will be greater than it.
Now pivot is in its final position.
→ Recursively apply the above steps to the subarray of elements with smaller values and separately to the subarray of elements with greater values.

## Partition Algorithm

```
PARTITION (A, p, r)
{  x = A[r]
   i = p-1
   for (j=p to r-1)
     { if (A[j] ≤ x)
         { i = i+1
           exchange A[i] with A[j]
         }
     }
   exchange A[i+1] with A[r]
   return i+1.
}.
```

Eg:-




And 7 is in its correct position.

## QUICK SORT Algorithm

```
QUICK_SORT (A, P, r)
{ if (P<r)
  { q = PARTITION (A, P, r)
    QUICKSORT (A, P, q-1)
    QUICK SORT (A, q+1, r)

  }
}
```

# STRASSEN'S MATRIX MULTIPLICATION.

It is an algorithm for matrix multiplication which is faster than the standard matrix multiplication algorithm.

1) Naive matrix multiplication.

Two matrices $A\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$ $B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$

$$C = A \times B = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

$$c_{11} = a_{11} * b_{11} + a_{12} * b_{21}$$

$$c_{12} = a_{11} * b_{12} + a_{12} * b_{22}$$

$$c_{21} = a_{21} * b_{11} + a_{22} * b_{21}$$

$$c_{22} = a_{21} * b_{12} + a_{22} * b_{22}.$$

$$c_{ij} = \sum_{i=1}^{n} A_{ik} * B_{kj}$$

Algorithm

```
for(i=0; i<n; i++)
  {for(j=0; j<n; j++)
    {  c[i,j]=0
       for(k=0; k<n; k++)
       { c[i,j] += A[i,k] * B[k,j]
       }
    }
  }
```

If has 3 for loops and each will iterate n times.

∴ Time Complexity = $O(n^3)$

## By using Divide and Conquer.

Strassen's algorithm is a recursive method for matrix multiplication where we divide the matrix into 4 sub-matrices of dimensions $n/2 \times n/2$ in each recursive step.

For eg:- Consider two 4×4 matrices A and B that we need to multiply. A 4×4 can be divided into four 2×2 matrices.

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \quad B = \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix}$$

(A11, A12, A21, A22 and B11, B12, B21, B22 mark the corresponding 2×2 blocks)

ie,

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

where $A_{11}$, $B_{11}$ ... ele are not elements. They are matrices.

### Algorithm

```
MM(A, B, n)
{ if (n<=2)
  { C11 = a11 * b11 + a12 * b21
    C12 = a11 * b12 + a12 * b22
    C21 = a21 * b11 + a22 * b21
    C22 = a21 * b12 + a22 * b22.
  }
```

else

{

mid = n/2 ; // dividing matrices as 2x2

MM (A11, B11, n/2) + MM (A12, B21, n/2)

MM (A11, B12, n/2) + MM (A12, B22, n/2)

MM (A21, B11, n/2) + MM (A22, B21, n/2)

MM (A21, B12, n/2) + MM (A22, B22, n/2)

}

}.

Recurrence relation.

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

$$T(n) = aT(n/b) + f(n)$$

$a = 8,\ b = 2,\ f(n) = n^2.$

$$\log_b^a = \log_2 8 = \frac{\log 8}{\log 2} = 3.$$

$$n^{\log_b^a} = n^3.$$

$f(n) < n^{\log_b^a}$ $\quad n^2 < n^3.$

$$O(n^{\log_b^a}) = O(n^3)$$