Divide and Conquer

Stages
- 1) Divide
- 2) Solve
- 3) Combine

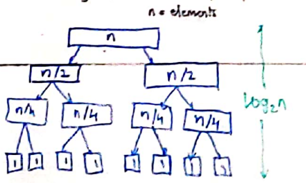
Control Abstraction
1) Divide : recursively divide the pb into smaller sub-pb
2) Solve : subpb are solved independently
3) Combine: combine solutions of sub-pb in order to derive the solution of original big pb.

Applications
1) Finding exp of the number
2) Multiplying large number
3) Multiplying matrix (Strassen's Algo.)
4) Sorting elements (Quicksort & Merge Sort)
5) Searching element from list (Binary Search)
6) Discrete Fourier Transform
7) Closest Pair Problem
8) Min-Max Pb

Sorting
- Process of arranging elements in certain order.
- Numeric data may be sorted in ascending or descending order
- Alphabet or Strings may be sorted in Lexicographical order.
- If no. of elements are small enough to sort in main memory, sorting → internal sorting
- If no. of elements are large to sort in main memory then we need secondary storage → external storage

Properties
1) In place - only require constant additional space to sort the algorithm
2) Stable - does not alter the relative position of same elements after sorting
3) Online - sort the data as it arrives.
4) Adaptive - performance of algo varies with i/p pattern
5) Incremental - build sorted sequence one no. at a time

Merge Sort
- Sorting smaller list is faster than sorting larger list
- Combining 2 sorted sublist is faster than that of 2 unsorted list
- M.S works in 3 stages:
  → divide, solve, cor
- M.S divides list element of n into 2 sublists each of size $n/2$
- This subdivision continuous until pb size = 1
- After hitting to pb size = 1, conquer phase starts



Algorithm for Merge sort

```
mergesort ( int[] a, int left, int right)
{
  if ( right > left)
  { middle = left + (right - left)/2 ;
    mergesort (a, left, middle);
    merge sort (a, middle +1, right);
    merge (a, left, middle, right); }
}
```

Time to merge two arrays each $N/2$ elements is linear, i.e. N

i] $T(1) = 1$

ii] $T(N) = 2T(N/2) + N$

iii] $T(N)/N = T(N/2) / (N/2) + 1$

Properties of Merge Sort
1) Not adaptive - running time of M.S doesn't change with i/p sequence.
2) Stable / Unstable - both implementation are possible
3) Not incremental - doesn't sort one by one element in each pass so it is not incremental
4) Not Online - need all data to be in memory at the time of sorting, so merge sort is not online
5) Not in place - it need Big oh of n extra space of size $(n/2)$.

Quicksort
- Tony Hoare
- Partition exchange sort

Best Case	Average Case	Worst average
$O(n \log n)$	$O(n \log n)$	$O(n^2)$

```
sort (A)
1. quickSort (A, 0, n-1)
end

Quicksort (A, left, right)
1. if (left < right) then
   pi = partition (A, left, right)
3. quicksort (A, left, pi-1)
4. quicksort (A, pi+1, right)
end

partition (A, left, right)
1. p = select pivot in A[left, right]
2. swap A[p] and A[right]
3. store = left
4. for i = left to right-1 do
5.    if (A[i] ≤ A[right]) then
6.       swap A[i] and A[store]
7.       store++
8. swap A[store] & A[right]
9. return store
end
```

Binary Search
- efficient than linear search
- for binary search, array must be sorted
- it is divide & conquer based search technique
- in each step, algo divides list into two halves and check if element to be searched is on upper or lower half of array

Algorithm for Binary Search

```
Algorithm BINARY_SEARCH (A, Key)
// Description : Perform Binary search on array A
// Input : Sorted array A of size n and Key to
   be searched
// Output : Success / Failure

low ← 1
high ← n-1
while low < high do
   mid ← (low + high)/2
   if A[mid] == Key then
      return mid
   else if A[mid] < Key then
      low ← mid +1
   else
      high ← mid -1
   end
end
return 0
```

Min-Max Problem
- used to find max and min element from given list.
- approach $(n-1)$ comparisons for finding max & same no. of comparisons for finding min
- it results in total $2n-1$ comparisons
- it works in 3 stages : divide, solve, combine

Algorithm for Min-Max

```
min_max (A, min, max, low, high)
{
 if (low == high) then
 {
   min = max = A[low]
 }
 else
 { if (A[low] < A[high]) then
   {
     min = A[low]
     max = A[high]
   }
   else
   {
     min = A[high]
     max = A[low]
   }
 }
 else
 {
   mid = (low + high)/2
   min_max (A, low, mid, min, max)
   min-max (A, mid+1, high, min, max)
 }
}
}
```

$\boxed{\text{Complexity} : \log_2 n}$

Strassen's matrix multiplication

```
Algorithm STRASSEN_MAT_MUL (int *A, int *B, int *C, int n)
// A and B input matrices
// C is output matrix
// All matrices are of size n x n

if n == 1 then
   *C = *C + (*A) * (*B)
else
   STRASSEN_MAT_MUL (A, B, C, n/4)
   STRASSEN_MAT_MUL (A, B+(n/4), C+(n/4), n/4)
   STRASSEN_MAT_MUL (A+2*(n/4), B, C+2*(n/4), n/4)
   STRASSEN_MAT_MUL (A+2*(n/4), B+2*(n/4), C+3*(n/4), n/4)
   STRASSEN_MAT_MUL (A+3*(n/4), B+2*(n/4), C+2*(n/4), n/4)
   STRASSEN_MAT_MUL (A+3*(n/4), B+3*(n/4), C+3*(n/4), n/4)
end
```

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

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

1 2 3 4

b a a b

b_{11} a_{11} a_{22} b_{22}

$P = (a_{11} + a_{22})(b_{11} + b_{22})$

$Q = b_{11}(a_{21} + a_{22})$

$R = a_{11}(b_{12} - b_{22})$

$S = a_{22}(b_{21} - b_{11})$

$T = b_{22}(a_{11} + a_{12})$

$U = (b_{11} + b_{12})(a_{21} - a_{11})$

$V = (b_{21} + b_{22})(a_{12} - a_{22})$

$c_{11} = P + S - T + V$

$c_{12} = R + T$

$c_{21} = Q + S$

$c_{22} = P + R - Q + U$