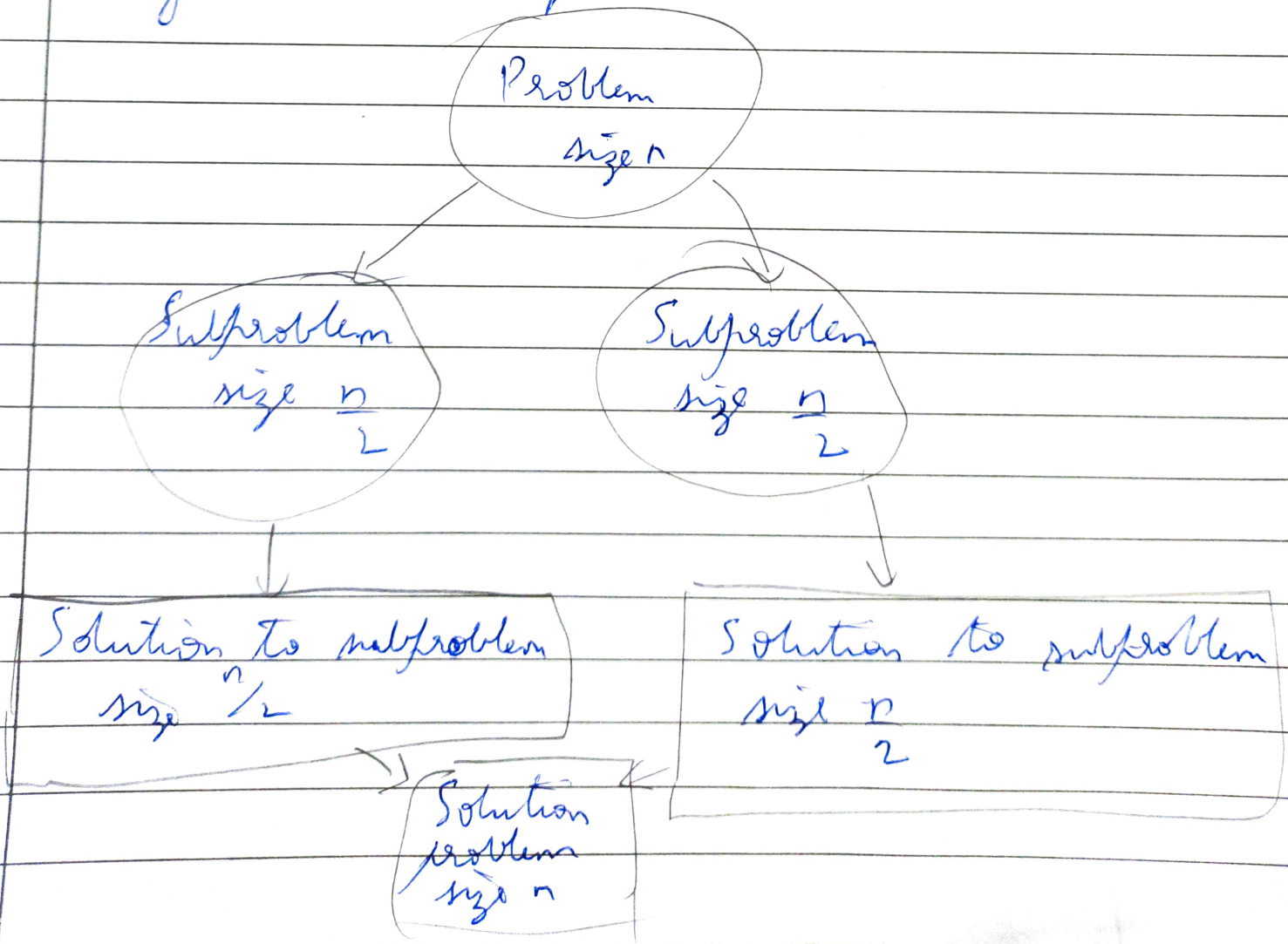
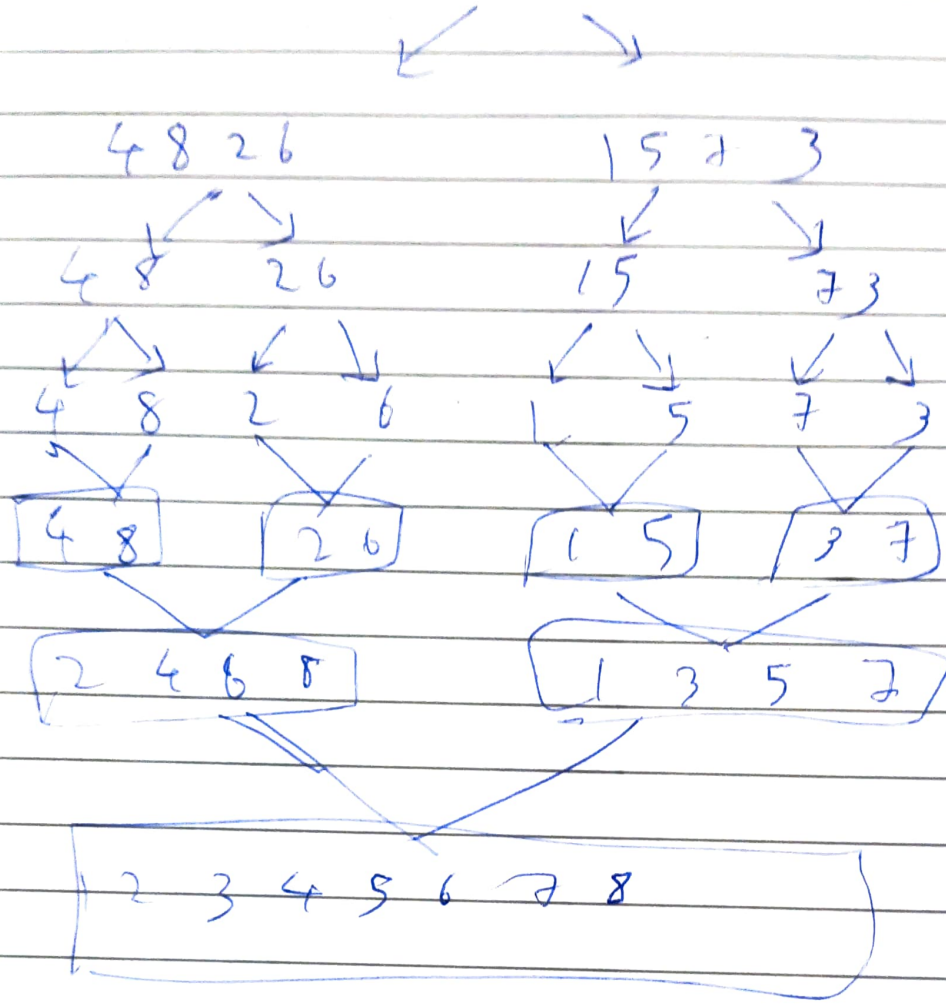
17) Merge Sort

→ Merge Sort is based on divide & conquer

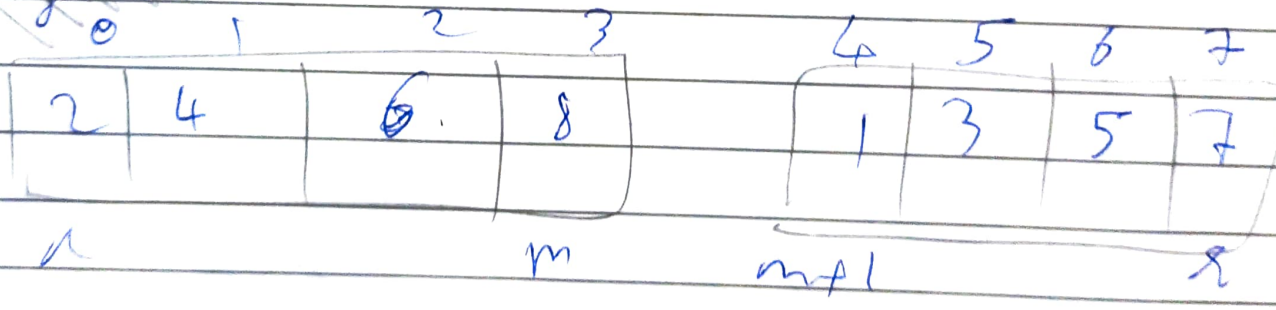
→ Diagramatic representation of solving a problem using divide & conquer



Eg:- 4 8 2 6 1 5 7 3



~~Mergesort~~



A [l ... r]

$m = \frac{l+r}{2}$

{ First Array A[l ... m]
Second Array A[m+1 ... r] } merge them together

$i \leftarrow 1$

$j \leftarrow m+1$

Let B be a new array

$k \leftarrow 1$

Compare $A[i]$ with $A[j]$

If $A[i]$ is less copy $B[k]$

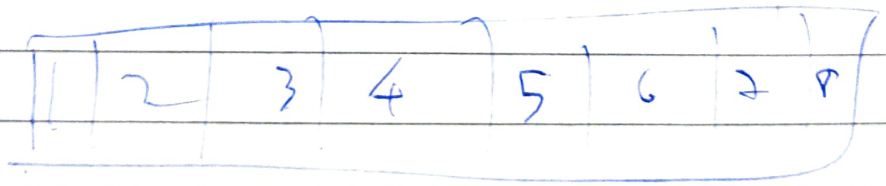
increment i & k

else copy $A[j]$ to $B[k]$

increment j & k

If there are no elements to be compared on any side, copy the remaining of the other side to the final array.

→ Copy all elements of array B back to the array

Array B → 

In-place → Bubble sort, Selection sort

Mergesort is not an inplace sorting

Algorithm Mergesort(A, l, r)

// Recursively sorts the array element using mergesort
// Input: An array A
// Output: A sorted array A

```
if l < r
    m = (l + r) / 2
    Mergesort (A, l, m)
    Mergesort (A, m+1, r)
    Merge (A, l, m, r)
```

Algorithm Merge (A, l, m, r)

// Merges 2 sorted array together
// Input: 2 sorted array A[l ... m] & A[m+1 ... r]
// Output: A single sorted Array A[l ... r]

```
i ← l
j ← m+1
k ← l
while (i <= m & j <= r)
    if A[i] < A[j]
        B[k++] ← A[i++]
    else
        B[k++] ← A[j++]
while i <= m
        B[k++] ← A[i++]
while j <= r
        B[k++] ← A[j++]
```

Copy all elements of B to A

Input size = n

Basic operation = Comparison

Let C(n) denotes the no. of comparison made if n=1 No. of comparison, i.e, C(1) = 0

$$C(n) = 2C\left(\frac{n}{2}\right) + f(n)$$

f(n) → function which denotes no. of comparisons made to merge 2 sorted arrays of size n/2

↓

Apply Masters theorem

$a = 2 \quad b = 2$

$n^{\log_b a} = n^{\log_2 2} = n$

$\Rightarrow$ Compare $n^{\log_b a}$ with $f(n)$

$\therefore n \; n \Rightarrow$ Both are same

↓

Case (2)

$\therefore C(n) = \theta(n^{\log_b a} \cdot \log n)$

$= \theta(n \log n)$

```
#include<stdio.h>
#include<stdlib.h>
#define SIZE 100
int main()
{
    int n,i,A[SIZE];
    printf("\n Read the no. of elements :");
    scanf("%d",&n);
    printf("\nRead elements \n");
    for(i=0;i<n;i++)
        scanf("%d",&A[i]);
    mergesort(A,0,n-1);
    printf("\n Sorted elements are \n");
    for(i=0;i<n;i++)
        printf("%d\n",A[i]);
    return 0;
}

void mergesort(int A[SIZE],int left,int right)
{
    int mid;
    if(left<right)
    {
        mid=(left+right)/2;
        mergesort(A,left,mid);
        mergesort(A,mid+1,right);
        merge(A,left,mid,right);
    }
```

```
void merge (int A[SIZE], int left, int mid,
int right)
{
    int i, j, k, B[SIZE];
    i = left;
    j = mid+1;
    k = left;
    while (i <= mid && j <= right)
    {
        if (A[i] < A[j])
            B[k++] = A[i++];
        else
            B[k++] = A[j++];
    }
    while (i <= mid)
            B[k++] = A[i++];
    while (j <= right)
            B[k++] = A[j++];
    for (i = left; i <= right; i++)
            A[i] = B[i];
```

19 Quick sort

→ Quicksort works on the principle of divide & conquer

→ In Quicksort to sort n elements, a concept by name 'partition' is used

→ In partition, we generally take either the first element or the last element as 'pivot', on comparing pivot with the remaining elements, at the end of the partition, we place the pivot in its actual position in the sorted list.

→ Depending on the position of pivot after partition, we can come up with either the best case or the worst case.

input is A[ ]

left = index pointing to first element

right = index pointing to the last element

i ← left +1
j ← right

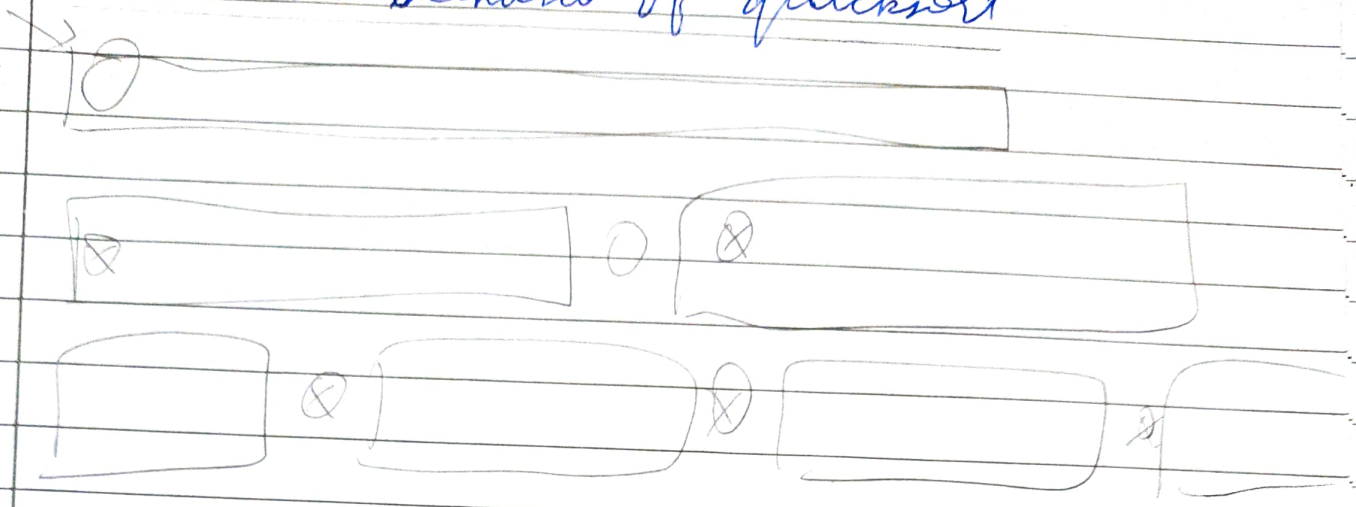
pivot = A[left]
pivot ≥ A[i] }, increment i
pivot < A[j], decrement j

if i < j
    swap(A[i], A[j]) & continue

else
    swap(A[l], A[j])
    Stop, return l

Pivot

Best case scenario of quicksort



Basic operation → Comparison

$C(n) = 2C\left(\frac{n}{2}\right) +$ No. of comparisons done to put pivot in the middle position

$C(n) = 2C\left(\frac{n}{2}\right) + n$

Apply Master Thm

$a = 2, \; b = 2$

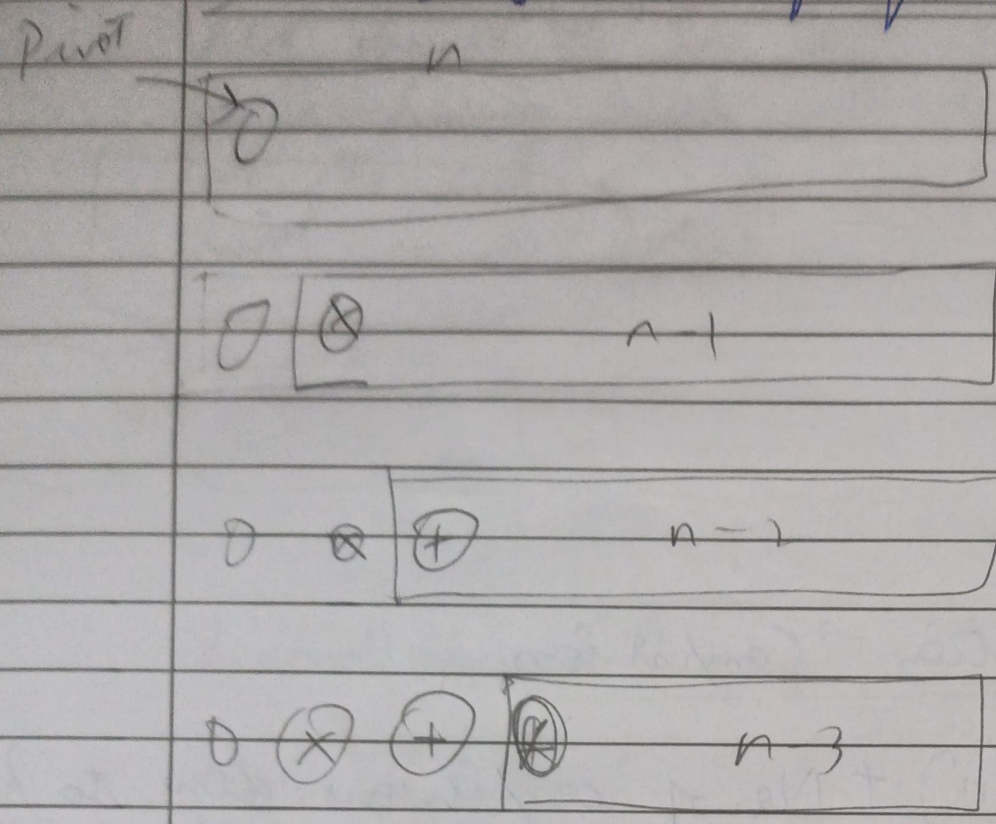
$f(n) = n$

$n^{\log_b a} = n^{\log_2 1} = n$

Case (2) of MT

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

Worst case scenario of quicksort



Eg: 1 2 3 4 5 6

1 | 2 3 4 5 6

1 2 | 3 4 5 6

$C(n) = C(n-1) +$ Comparison done

$C(n) = C(n-1) + n + 1$

$= C(n-2) + (n-1) + n + 1$

$= C(n-3) + (n-2) + (n-1) + n + 1$

$= 3 + \cdots (1+n) + n + (n-1) + (n-2) + \cdots$

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

$$= \theta(n^2)$$

Algorithm quicksort(A, left, right)

// Sorts a set of elements recursively using quicksort

// Input: An array A, indexed from left to right

// output: A sorted array A

if left < right
    s ← Partition(A, left, right)
    Quicksort(A, left, s-1)
    Quicksort(A, s+1, right)

Algorithm Partition(A, left, right)

// Partition a subarray A indexed from left to right

Pivot ← A[left]
i ← left + 1
j ← right
while True

```
while pivot >= A[i] & i <= right
        increment i }
while pivot < A[j] decrement j
if (i < j)
        swap(A[i], A[j])
else
        swap(A[i], A[j])
if (i >= j)
        break
return j
```

2) Multiplication of 2 integers using Divide & Conquer

* Let a & b be 2 integers, the product of a &

b can be expressed as
$c = a \times b$

In brut force method if 'n' denotes the size of 2 integers (a & b), the no. of multiplications done is n^2

↓ By using divide & conquer.

we decrease the time complexity to $n^{\log_2 3}$

Let a & b be 2 integers denoted with (Let size of a & b be n)

$a = a_1 a_0$ | where a_1, a_0, b_1, b_0 are digits of size $\frac{n}{2}$

$b = b_1 b_0$

The product c is calculated as

Step 1:- $c_0 = a_0 \times b_0$ → Multiplication

Step 2:- $c_2 = a_1 \times b_1$ → Multiplication

Step 3:- $c_1 = (a_0 + a_1) \times (b_1 + b_0) - (c_2 + c_0)$ → Multiplication

Step 4:- $c = c_2 c_2 10^n + c_1 10^{n/2} + c_0$ → Shift

Eg:- $a = 26$, $b = 45$

$a_1 = 2$ | $b_1 = 4$

$a_0 = 6$ | $b_0 = 5$

$c_2 = 8$

$c_0 = 30$

$c_1 = (8 * 9) - 38$

$= 72 - 38 = 34$

$c = 800 + 340 + 30$

$= 1170$

a) $a = 12$ $b = 12$

$a_1 = 12$ b_1

$a_0 = 12$

$M(n) = 3M\left(\frac{n}{2}\right) + C$

$M(n) = 3M\left(\frac{n}{2}\right) +$ Time used for shift

c_n

Masters Theorem

$n^{\log_b a} = n^{1.59}$

Case (1) :- $M(n) = O(n^{1.59})$

22) Strapes Multiplication of Matrix

→ Brute Force, time complexity is n^3

→ Strassen's method is based on divide & conquer, in which time complexity is reduced to $n^{2.8}$

→ Let A & B be 2 matrices of order n×n

$$A = \begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix}, B = \begin{bmatrix} B_1 & B_2 \\ B_3 & B_4 \end{bmatrix}, C = \begin{bmatrix} C_1 & C_2 \\ C_3 & C_4 \end{bmatrix}$$

$D = A_1(B_2 - B_4)$ → 1

$E = A_3(B_3 - B_1)$ → 1

$F = B_1(A_3 + A_4)$ → 1

$G = B_3(A_1 + A_2)$ → 1

$H = (A_3 - A_1)(B_1 + B_2)$ → 1

$I = (A_2 - A_4)(B_3 + B_4)$ → 1

$J = (A_1 + A_4)(B_1 + B_4)$ → 1

On solving

$C_1 = E + I + J - G$

$C_2 = D + G$

$C_3 = E + F$

$C_4 = D + H + J - F$

$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, B = \begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix}$

$D = 1(1-2) = -1$

$E = 4(2-1) = 4$

$F = 1(3) = 3$

$G = 2(3) = 6$

$H = 2(3) = 6$

$I = -2(1) = -2$

$J = 15$

$C_1 = E + B + J - G$

$4 - 8 + 15 - 6 = 5$

$C_2 = B + G = -1 + 6 = 5$

$C_3 = E + F = 4 + 7 = 11$

$C_4 = B + H + J - F = -1 + 4 + 15 - 7 = 11$

$\therefore \begin{bmatrix} 5 & 5 \\ 17 & 11 \end{bmatrix}$

$M(n) = 7M\left(\frac{n}{2}\right) + cn$

Base case $M(1) = 1$

$a = 7, b = 2$

$n^{\log_b a} = n^{\log_2 7} = n^{2.81}$

$\therefore M(n) = \theta(n^{2.81})$