om Sa

Quick sort:-

One element as a pivot element

element < pivot are placed on left sub array

element > pivot are placed on right sub array.

Algorithm Quicksort (A[low ... high]) → (A, loco, high)

{

if (low < hight)

{

m ← partition (A [low ... high])

Quicksort (A, loco, m-1) → befor pivot-means m-1

Quicksort (A, m+1, high) → after pivot element

}

Algorithm partition (A [loco ... high])

{ pivot = a[low]

Assume i = loco, j = high

while (i ≤ j)

{

while (a[i] ≤ pivot)

i++; ↳ This condition failed check j value

while (a[j] ≥ pivot)

j--;

if (i ≤ j) → i, ≤ j

Then swap(a[i], a[j]) index index

}

if (i > j)

swap (a[j], pivot]

return j;

}

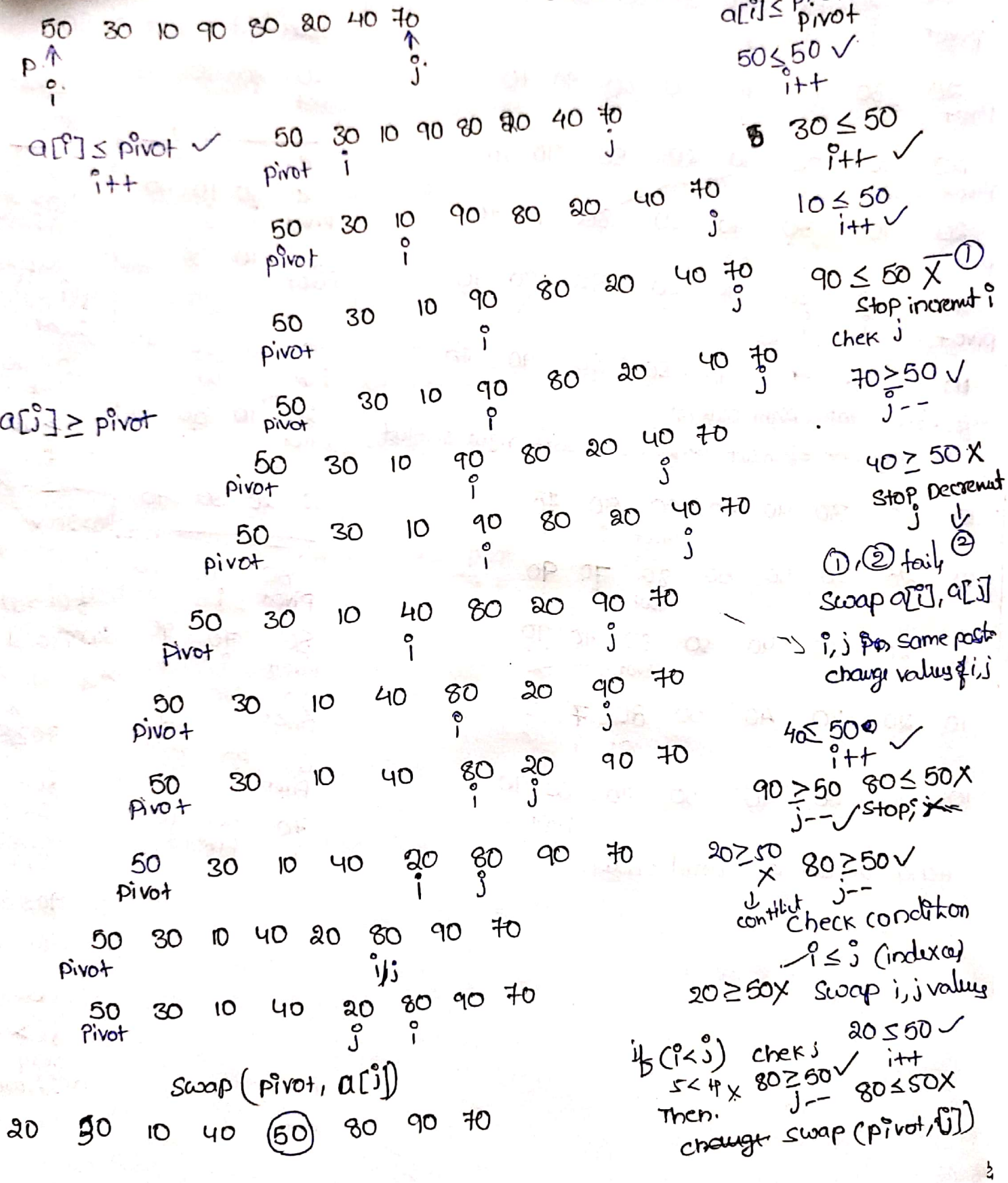
Selecting greater than.

Less that pivot

i ≤ → ← ≥ j

j value upgraded into m.

Quicksort:-


50 30 10 90 80 80 40 70
p i j
a[i] ≤ pivot ✓
i++
50 30 10 90 80 80 40 70
pivot i j
50 30 10 90 80 20 40 70
pivot i j
50 30 10 90 80 20 40 70
pivot i j
a[j] ≥ pivot
50 30 10 90 80 20 40 70
pivot i j
50 30 10 90 80 20 40 70
pivot i j
50 30 10 40 80 20 90 70
Pivot i j
50 30 10 40 80 20 90 70
Pivot i j
50 30 10 40 20 80 90 70
Pivot i/j
50 30 10 40 20 80 90 70
Pivot j i
Swap (pivot, a[j])
20 30 10 40 50 80 90 70
consida pivot = 50
a[i] ≤ pivot
50 ≤ 50 ✓
i++
30 ≤ 50
i++ ✓
10 ≤ 50
i++ ✓
90 ≤ 50 X ①
Stop increment i
Chek j
70 ≥ 50 ✓
j--
40 ≥ 50 X
Stop Decrement
j ↓ ②
①, ② fail
Swap a[i], a[j]
i, j same post
change value of i,j
40 ≤ 50
i++ ✓
90 ≥ 50 80 ≤ 50 X
j-- ✓ stop;
20 ≥ 50 80 ≥ 50 ✓
X j--
↓ conflict
Check condition
i ≤ j (index)
20 ≥ 50 X Swap i, j values
20 ≤ 50 ✓
if (i < j) chek j
5 < 4 X 80 ≥ 50 ✓ i++
Then. j-- 80 ≤ 50 X
change swap (pivot, j)

20 30 10 40 50 80 90 70
pivot i j

20 30 10 40 50 80 90 70
pivot i j --j

20 30 10 40 50 80 90 70
pivot i j

20 10 30 40 50 80 90 70
pivot i j

20 10 30 40 50 80 90 70
pivot i,j

20 10 30 40 50 80 90 70
pivot j i

10 20 30 40 50 80 90 70

Left sublist pivot Right sub list

↳ only one element here we will sort right sublist.

10 20 30 40 50 80 90 70
pivot i j

10 20 30 40 50 80 70 90
pivot i j

10 20 30 40 50 80 70 90
pivot i,j

10 20 30 40 50 80 70 90
pivot j i

10 20 30 40 50 70 80 90
pivot

Here the list is sorted order.

we have two lists

① below 50 called left sublist

② above 50 calle right sublist

take left sublist of 50

20 30 10 40
pivot i j

~~20 30 10 40~~
pivot i j

20 30 10 40
pivot i j

20 10 30 40
pivot i,j

10 20 30 40

a[i] ≤ pivot
30 ≤ 20 ✗
i++
10
40 ≥ 20 ✓
j--
10 ≥ 20 ✗
if(i ≤ j) ✓
swap
30 ≤ 20 ✓
i++
30 ≥ 10
j-- ~~✗~~
30 ≤ 20 ✗
10 ≥ 20 ✗
check j crossed i

bus
80 90 70
pivot i j

80 70 90
pivot i j

80 70 90
pivot i,j

80 70 90
pivot j i

70 80 90
i pivot j

if(i > j)
swap(a[j], pivot)
80 90 ≤ 80 ✗
70 ≥ 80 ✗
if(i ≤ j)
swap
70 ≤ 80 ✓
i++
90 ≤ 80 ✗
90 ≥ 80 ✓
j--
j crossed i
if(i ≥ j)
swap
a[j], pivot

Passen's Anta.

A (left sublist)

20	30	40	70

i

A (right sublist)

10	50	60

j

```
if (A[i] <= A[j])
{ temp[k] ← A[i]
  i ← i+1
  k ← k+1
}
else
{ temp[k] ← A[j]
  j ← j+1
  k ← k+1
}
```

40 <= 50 ✓

temp

10	20	30	40	

k

Note that i remains there and only j is incremented

A (left sublist)

20	30	40	70

i

A (right sublist)

10	50	60

j

A (left sublist)

20	30	40	70

i

A (right sublist)

10	50	60

j

temp

10	20	30	40	50	

k

else part gets executed.

only j and k will be incremented.

```
if (A[i] <= A[j])
{ temp[k] ← A[i]
  i ← i+1;
  k ← k+1
}
else { temp[k] ← A[j]
       j ← j+1
       k ← k+1
}
```

A (left sublist)

20	30	40	70

i

A (right sublist)

10	50	60

j

A (left sublist)

20	30	40	70

i

A (right sublist)

10	50	60

j

temp

10	20	30	40	50	60	

k.

← else part executed [In previous pass only j and k will be incremented]

here i remains as it is but the right sublist is over

A (left sublist)

20	30	40	70

i

A (right sublist)

10	50	60

j

A (left sublist)

20	30	40	70

i

A (right sublist)

10	50	60

j

temp

10	20	30	40	50	60	70

← ~~else~~ If part gets executed.

finally, copy the elements of array temp to array A. array A contains sorted list

10	20	30	40	50	60	70

Strassen's Matrix Multiplication:-

Suppose we want to multiply two matrices A and B each of size N i.e.

$$C = A \times B$$

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

The multiplication gives

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

Total to accomplish 2×2 matrix multiplication there are total 8 multiplication and 4 additions.

```
Algorithm Mat_mul(A,B,C,n)
{ for i=1 to n do
    C[i,j]=0;
    for k=1 to n do
    c[i,j] = c[i,j] + A[i,j] × B[i,j]  — n×n×n = O(n^3)
}
```

Strassen showed that 2×2 matrix multiplication can be accomplished in 7 multiplication and 18 additions or subtractions.

The divide and conquer approach can be used for implementing Strassen's matrix multiplication.

Divide:- Divide matrices into sub matrices: A_0, A_1, A_2 etc

conquer:- use a group of matrix multiply equation

combine:- Recursively multiply sub-matrices and get the final result of multiplication after performing required additions & subtractions

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$S_1 = (A_{11} + A_{22})(B_{11} + B_{22})$
$S_2 = (A_{21} + A_{22}) \times B_{11}$
$S_3 = A_{11} \times (B_{12} - B_{22})$
$S_4 = A_{22} \times (B_{21} - B_{11})$
$S_5 = (A_{11} + A_{12}) \times B_{22}$
$S_6 = (A_{21} - A_{11}) \times (B_{11} + B_{12})$
$S_7 = (A_{12} - A_{22}) \times (B_{21} + B_{22})$
$C_{11} = S_1 + S_4 - S_5 + S_7$
$C_{12} = S_3 + S_5$
$C_{22} = S_1 + S_3 - S_2 + S_6$
$C_{21} = S_2 + S_4$

Note:- we will compare the actual our tradition matrix multiplication procedure with strassen's procedure.

In strassen's multiplication

$C_{11} = S_1 + S_4 - S_5 + S_7$

$= (A_{11} + A_{22})(B_{11} + B_{22}) + A_{22} \times (B_{21} - B_{11}) - (A_{11} + A_{12}) \times B_{22} + (A_{12} - A_{22}) \times (B_{21} + B_{22})$

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

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

Algorithm

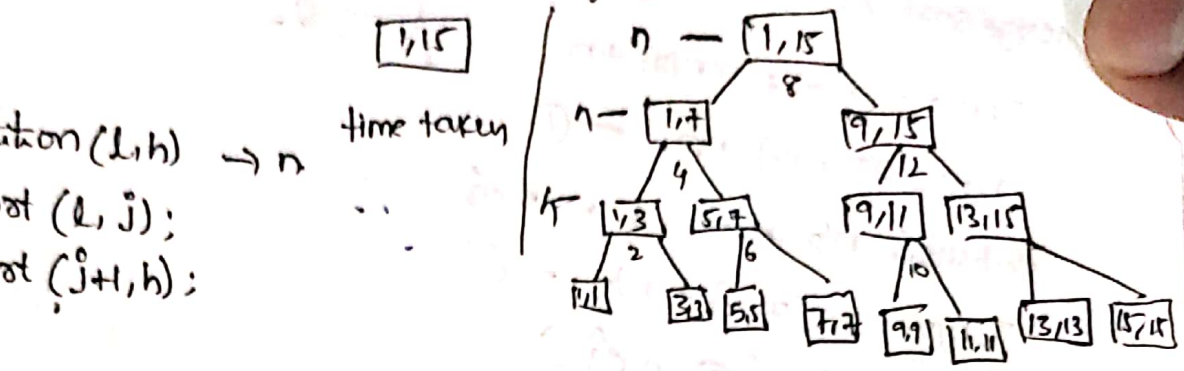
```
Algorithm St_mul(int*A, int*B,
                 int*C, int n)
{
  if (n==1) then
  {
    (*C) = (*C) + (*A)*(*B)
  }
  else
  { St_mul(A,B,C, n/4);
    St_mul(A,B+(n/4), C+(n/4), n/4)
    St_mul(A+2*(n/4), B, C+2*(n/4), n/4);
    St_mul(A+2*(n/4), B+(n/4), C+3*(n/4), n/4);
    St_mul(A+(n/4), B+2*(n/4), C, n/4);
    St_mul(A+(n/4), B+3*(n/4)+C+(n/4), n/4)
    St_mul(A+3*(n/4), B+2*(n/4), C+2*(n/4), n/4)
    St_mul(A+3*(n/4), B+3*(n/4), C+3*(n/4), n/4);
  }
}
```

Analysis

$T(n) = 1$
$T(n) = 7\,T(n/2)$
$T(n) = 7^k T(n/2)$
$T(n) = 7^{\log n}$
$T(n) = n^{\log 7}$
$= n^{2.81}$

```
Quicksort(l,h)
{
  if(l<h)
  {
    j = partition(l,h)  → n
    Quicksort(l, j);
    Quicksort(j+1, h);
  }
}
```

time taken



$$\frac{n}{2} = 1$$

$$\frac{n}{2^k} = 1 \Rightarrow 2^k = n \Rightarrow k = \log n.$$

How many times partition of the algorithm (n log n)

Quicksort → Best case (O nlogn)

median

1 2 3 (4) 5 6 7

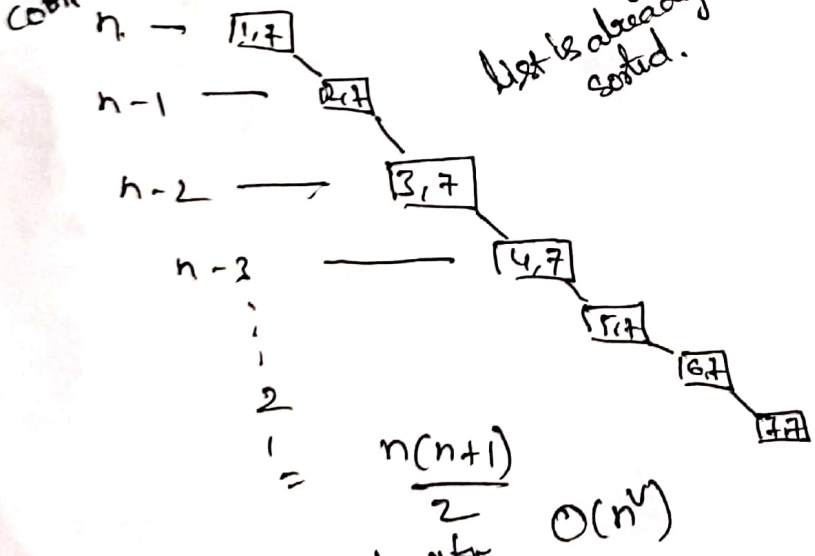
↑ middle (or) median

Pivot

(2) 4 8 10 16 18 17

i j

comparison partition done at beginning list is already sorted.



$= \frac{n(n+1)}{2}$ $O(n^2)$

worst case

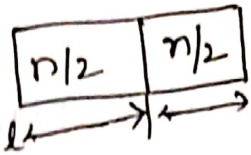
2 4 8 (10) 16 18 17

↓ select pivot as middle elm.

① select middle element as pivot

② select Random element as pivot

Merge sort Analysis:-



$T(n) = T(n/2) + T(n/2) + n$

$T(n) = 2T(n/2) + n$ — ①

Substitute $n/2$ in place of n in eq①

$T(n/2) = 2T(n/4) + n/2$ — ②

Substitute eq② in eq①

$T(n) = 2\{2T(n/4) + n/2\} + n$

$T(n) = 2^2 T(n/2^2) + 2n$ — ③

Substitute $n/4$ in place of n in eq③

$T(n/4) = 2T(n/8) + n/4$ — ④

$T(n) = 2^2\{2T(n/8) + n/4\} + 2n$

$T(n) = 2^3 T(n/2^3) + 3n$

$\vdots$

$T(n) = 2^4 T(n/2^4) + 4n$

$T(n) = 2^i T(n/2^i) + in$

$T(1) = 1$

let $\frac{n}{2^i} = 1 \Rightarrow n = 2^i$

$\log_2 n = \log_2 2^i = i \log_2 2$

$\log_2 n = i$

$T(n) = nT(1) + n\log_2 n$

$T(n) = n + n\log_2 n$

$= O(n^{\log_2 n})$

```
Algorithm Mergesort (int A[0...n-1], low, high)
// problem Description : this Algorithm is for sorting the elements using Merge sort.
// Input : Array A of unsorted elements, low as beginning.
// pointer of Array A and high as end pointer of array A.
// output : sorted array A[0....n-1]
  if (low < high) then
{
  mid ← (low+high)/2   // split the list at mid
  Mergesort (A, low, mid)  // first sublist
  Mergesort (A, mid+1, high) // second sublist
  Combine (A, low, mid, high) // merging of two sublist

Algorithm combine (A[0...n-1], low, mid, high)

    k ← low;   // k as index for array temp.
    i ← low;   // i as index for left sublist of array A.
    j ← mid+1  // j as index for right sublist of array A.
  while (i<=mid and j<= high) do
  {
    if (A[i] <= A[j]) then
   // if smaller element is present in Left sublist
    { // copy that smaller element to temp array [temp[k] ← A[i]
      temp[k] ← A[i]
       i ← i+1
       k ← k+1
    }
   else // smaller element is present in right sublist
    { // copy that smaller element to temp array.
      temp[k] ← A[j]
       j ← j+1
       k ← k+1
    }
} // co
```

// copy remaining elements of left sublist to temp

```
while (i<=mid) do
{
    temp[k] ← A[i]
    i ← i+1
    k ← k+1
}
// copy remaining elements of right sublist to temp
while (j<=high) do
{
    temp[k] ← A[j]
    j ← j+1
    k ← k+1
}
```

Analysis:- In merge sort algorithm the two recursive calls are made. Each recursive call focuses on n/2 elements of the list. After recursive call one call is made to combine two sublists i.e. to merge all n elements. Hence.

$T(n) = T(n/2) + T(n/2) + cn$

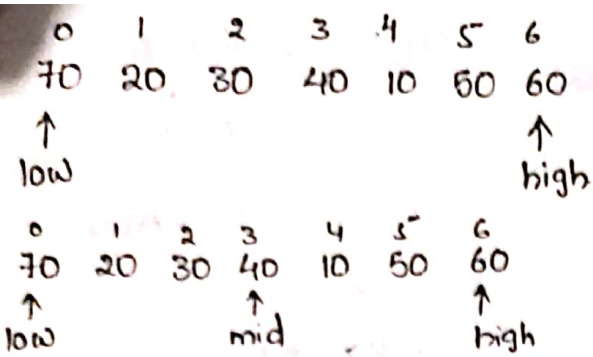
- $T(n/2)$ → Time taken by left sublist to get sorted
- $T(n/2)$ → Time taken by right sublist to get sorted.
- cn → Time taken for combining two sublist

i.e; $T(n) = 2T(n/2) + cn$

$T(1) = 0$

$T(n) = \Theta(n \log_2 n)$

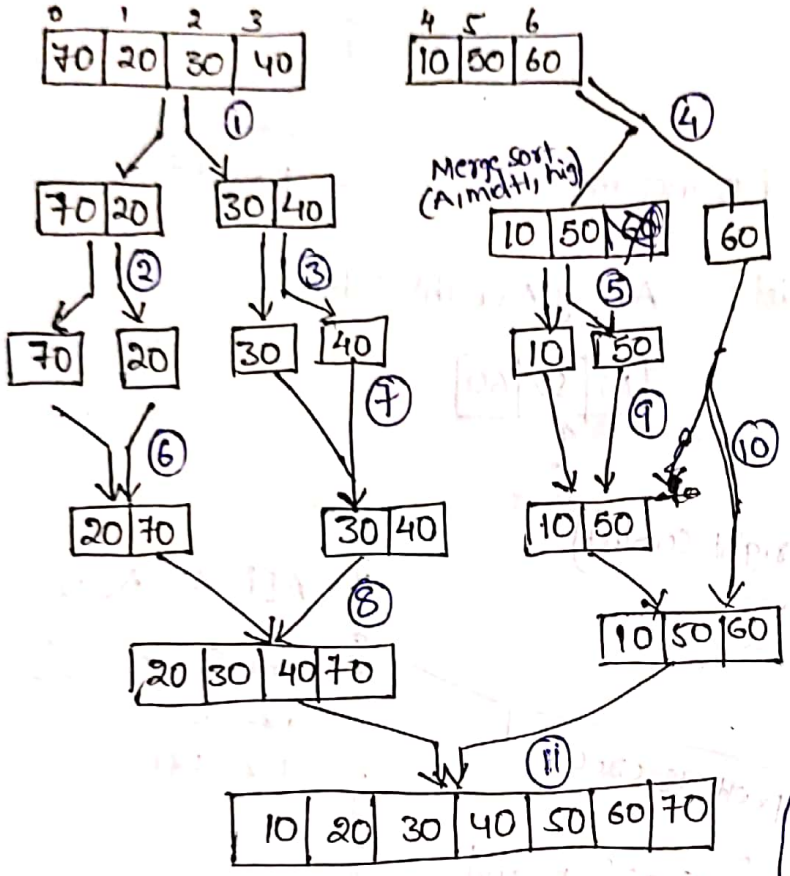
As per Masters theorem.



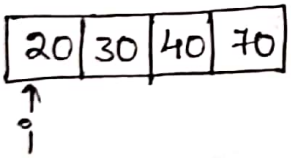
① The list be subdivided [1,2,3,4,5]

②

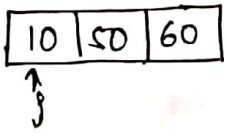
Then we will make two sublists



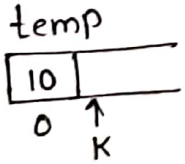
Array A (left sublist)



Array A (right Sublist)



Initially K=0 Then K will be incremented



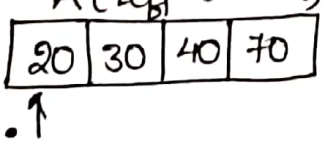
→ else part of the algorithm gets executed.

```
if (A[i] <= A[j])
{
  temp[k] ← A[i]
  i ← i+1
  k ← k+1
}
else
{
  temp[k] ← A[j]
  j ← j+1
  k ← k+1
}
```

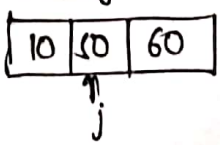
A[i] <= A[j]
20 <= 10 X
else
k → 10
k++ 0 → k.

NOTE! that i remains There and j is incremented

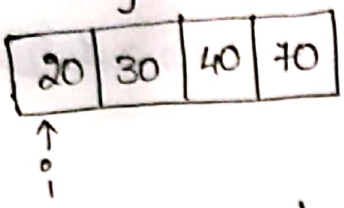
A (Left Sublists)



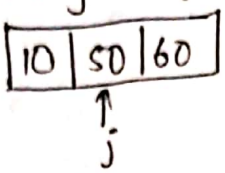
A (right sublist)



Array A (left sublist)



Array A (right sublist)



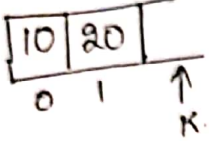
Applicable part of Algorithm

```
if (A[i] <= A[j]
{ temp[k] ← A[i]
  i ← i+1;
  k ← k+1;
}
else
{ temp[k] ← A[j]
  j ← j+1
  k ← k+1
}
```

K=1 It is advanced later on

if part algorithm executed.

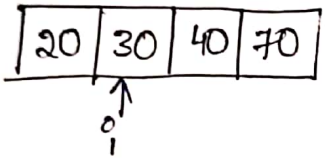
temp



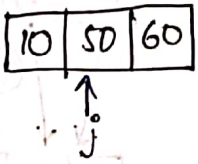
moves a head

NOTE! that j remains there and only i is incremented

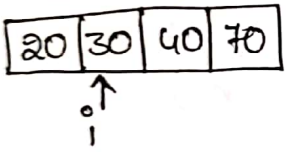
Array A (Left sublist)



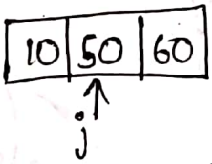
Array A (right sublist)



Array A (left sublist)



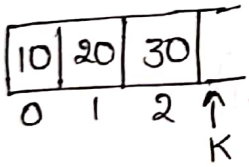
Array A (right sublist)




```
if (A[i] <= A[j]
{ temp[k] ← A[i]
  i ← i+1
  k ← k+1
}
else
{ temp[k] ← A[j]
  j ← j+1
  k ← k+1
}
```

if part is executed

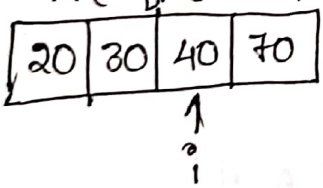
Now K=2



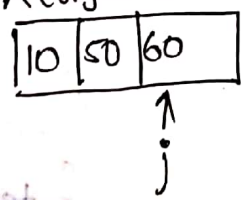
moves ahead

Note that j is as it is and only i is incremented.

A (Left Sublist)



A (right sublist)



```
Algorithm mm(A,B,n)
{
  if (n<=2)
  {
    c=4 -formula
    !
  else
  {
    mid...n/2
  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)
  }
}
```

8 multiplication

$$R.R \rightarrow T(n) = \begin{cases} 1 & n \le 2 \\ 8T(n/2) + n^2 & n > 2 \end{cases}$$ → Apply MASTER

$a=8$

$b=2$ $f(n)=n^2$ $\log_b^a = \log_2^8 = 3 \rightarrow O(n^3)$

$n^k = n^2$

$$T(n) = \begin{cases} 1 & n \le 2 \\ 7T(n/2) + n^2 & n > 2 \end{cases}$$

$\log_2^7 = 2.81$

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