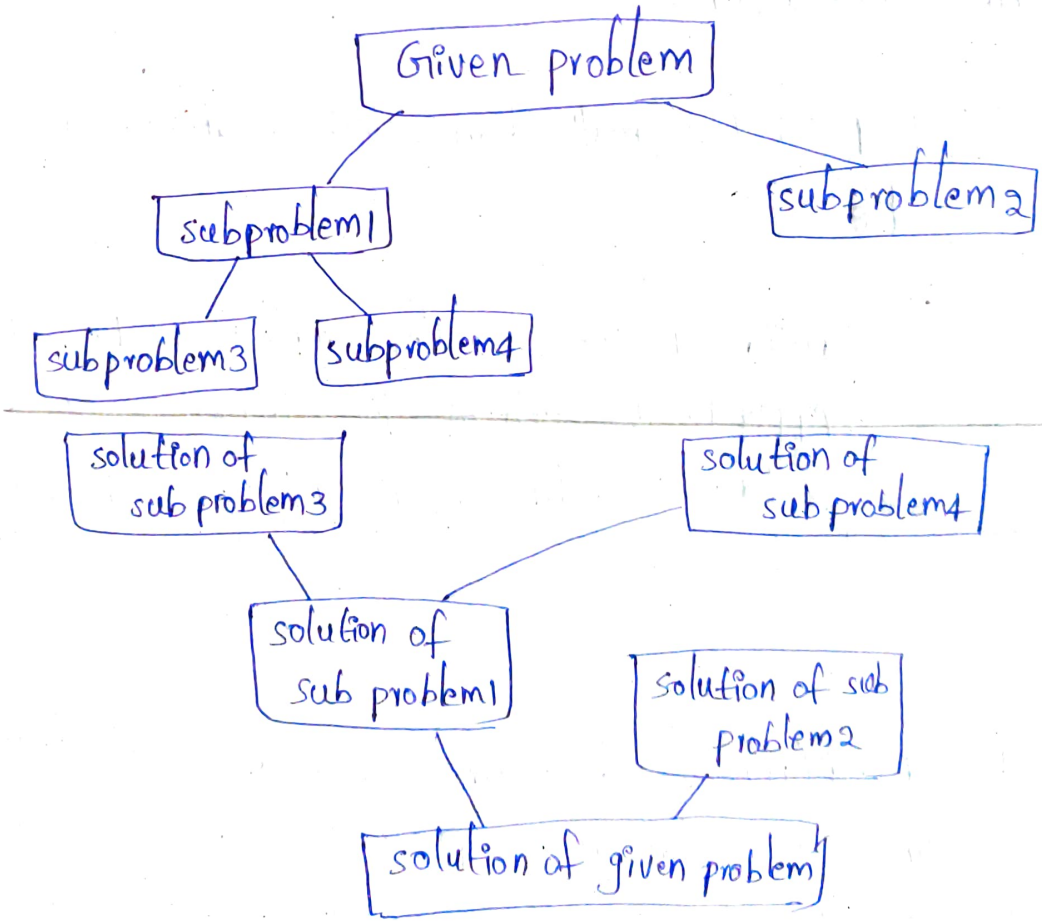
UNIT-2

divide & conquer

* If the problem is given we divide the problem into 'K' no.of sub problems, each size is 'K'. If we get the solution of each sub problem then we start at that point otherwise we still divide the problem into no.of smaller problems we solve all individual sub problems and combine all the solutions. That is the required solution of given problem.

Ex:- quick sort, merge sort.



* divide & conquer strategy involves three steps at each level of recursion.

(1) divide: Given problem is divided into no. of sub problems.

(2) conquer: Conquer the sub problems by solving them recursively.

(3) combine: combine the solutions to sub problems with the solution of subproblem solve the solution of given problem.

General method (control abstraction):-

```
Algorithm DC(P)
{
    if P is too small than return solution of P;
    else
    {
        divide (P) into smaller problems P1, P2 --- Pk ≥ 1
        Apply DC to each of the subproblems;
        return combine ( DC(P1), DC(P2), ---DC(Pk));
    }
}
```

* Let T(n) be the total time required to find the solution for the given problem.
* Let g(n) be the time required to solve the problem if 'n' is small.

* Let $f(n)$ be the time required for dividing the problem 'p' into several sub problems.
* Then the computing time is required as a recurence relation.

$$T(n) = \begin{cases} g(n) - \text{If } n \text{ is small} \\ T(n_1)+T(n_2)---+T(n_k)+f(n), \text{otherwise} \end{cases}$$

$$T(n) = \begin{cases} T(1) & n=1 \\ aT(n/b)+f(n), & n>1 \end{cases}$$

* Many divide & conquer strategy have the time complexity of.
* where a, b are known constants. we assume that $T(1)$ is known and 'n' is power of b.

$$n = b^k$$

① $$T(n) = \begin{cases} 2 & n=1 \\ 2T(n/2)+n & n>1 \end{cases}$$ find the solution of following recurence relation?

Sol:-

$$T(n) = 2T(n/2)+n \longrightarrow ①$$

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

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

sub eq ② in eq ①

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

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

$T(n) = 4T(n/4) + 2n \longrightarrow (3)$

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

$T(n/4) = 2T(n/8) + n/4 \longrightarrow (4)$

sub eq (4) in eq (3)

$T(n) = 4(2T(n/8) + n/4) + 2n$

$T(n) = 8T(n/8) + n + 2n$

$T(n) = 8T(n/8) + 3n$

$\vdots$

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

$= 2^k T(n/2^k) + kn \longrightarrow$ $n = 2^k$, $k = \log_2 n$

$= n\,2 + \log_2 n \cdot n$

$= 2n + n\log_2 n$

T.C $= O(n \log_2 n)$

(2) $T(n) = 2T(n/2) + 2$, $T(1) = 0$, $T(2) = 1$?

Sol: $T(n) = 2T(n/2) + 2 \longrightarrow (1)$

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

$T(n/2) = 2T(n/4) + 2 \longrightarrow (2)$

sub (2) in (1)

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

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

$T(n) \Rightarrow 4T(n/4) + 6 \longrightarrow (3)$

$T(n/4) = 2T(n/8) + 2 \longrightarrow (4)$

sub (4) in (3)

$T(n) = 4(2T(n/8) + 2) + 6$

$T(n) \Rightarrow 8T(n/8) + 8 + 6$

$T(n) = 8T(n/8) + 14$

$T(n) = 2^3 T(n/2^3) + 2^3 + 2^2 + 2^1$

$T(n) = 2^k T(n/2^k) + 2^k + 6$

$n = 2^k$

$k = \log_2 n$

$T(n) = nT(n/n) + n + 6$

$T(n) = n(0) + n + 6$

$T(n) = O(n)$

(or)

$\Rightarrow 2^{k-1} T(n/2^{k-1}) + 2^{k-1} + \cdots + 2^2 + 2 + 1 - 1$

$\Rightarrow \frac{2^k}{2^1} T\left(\frac{2 \cdot n}{2^k}\right) + \frac{2^{k-1}}{2-1} - 1$

$2^k = n$

$\Rightarrow \frac{n}{2} T(2) + n - 2$

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

$\Rightarrow \frac{3n}{2} \Rightarrow O(n)$,

(3) $T(n) = \begin{cases} k & n=1 \\ 3T(n/2)+kn & n>1,\ n \text{ is a power of } 2 \end{cases}$

Sol:-

$T(n) = 3T(n/2) + kn \longrightarrow (1)$

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

$T(n/2) = 3T(n/4) + kn/2 \longrightarrow (2)$

sub (2) in (1)

$T(n) = 3\left(3T(n/4) + kn/2\right) + kn$

$T(n) = 9T(n/4) + 3kn/2 + kn$

$T(n) = 9T(n/4) + \frac{5kn}{2} \longrightarrow (3)$

$T(n/4) = \left(3T(n/4)/2\right) + kn/4$

$T(n/4) = 3T(n/8) + \frac{kn}{4} \longrightarrow (4)$

sub (4) in (3)

$T(n) = 9\left(3T(n/8) + \frac{kn}{4}\right) + \frac{5kn}{2}$

$T(n) = 27T(n/8) + \frac{9kn}{4} + \frac{5kn}{2}$

$T(n) = 3^3T(n/2^3) + \frac{3^2kn}{2^2} + \frac{3^1kn}{2^1} + kn$

applications:-

① binary search:

* Let A[1:n] be the array of elements in the non-decreasing order. consider the problem determining whether a given element x is present in the array. If x is present in the array. position is determine. If it is not present in the array. value 0 is returned.

* Let P = [n, A1 ------ An, x]

denotes an arbitory sequence where n, is the no. of elements in an array A1, --- An are array elements and 'x' is the element to be searched.

* divide & conquer strategy is used to solve the problem. If the problem 'p' is small means that the array consists of only one element. The it will take one unit of time to determine 'x' is in the array (or) not. If the problem is not small. divide the array into two equal sub arrays.

* indexes are [i, j], then mid value $= \frac{[i+j]}{2}$. (2)

* Their are three posibilities:

(1) if A[q] is x [A[q]=x], problem is solved & return to q.

(2) if x < A[q], then the element has to be in b/w A[1, q-1], therefore the problem size is reduced.

(3) if x > A[q], then the element has to be in b/w A[q+1, j], ∴ the problem size is reduced.

And the process is repeated untill the element is found.

Algorithm:-

```
Binarysearch_iterative(A[1.....n], n, x)
{
    low = l
    high = h
    while (low <= high)
    {
        mid = (low + high) / 2;
        if (A[mid] == x)
            return mid;
        else if (x < A[mid])
            high = mid - 1;
        else
            low = mid + 1;
    }
    return -1;
}
```

Example:-

1 2 3 4 5 6 7 8 9 10 11 12 13 14

(*) 1, 2, 5, 6, 7, 10, 24, 56, 84, 100, 115, 120, 131, 150

(1) search element $x = 150$, $n = 14$?

sol:-

low	high	mid	A[mid]	x ? A[mid]
1	14	7	24	150 > 24
8	14	11	115	150 > 115
12	14	13	131	150 > 131
14	14	14	150	150 = 150

(2) search element $x = 100$, $n = 14$?

low	high	mid	A[mid]	x ? A[mid]
1	14	7	24	100 > 24
8	14	11	115	100 < 115
8	10	9	84	100 > 84
10	10	10	(*) 100	100 = 100

(3) search element $x = 9$, $n = 14$?

low	high	mid	A[mid]	x ? A[mid]
1	14	7	24	9 < 24
1	6	3	5	9 > 5
4	6	5	7	9 > 7
6	6	6	10	9 < 10

element not found

(4) search element $x = 84$, $n = 14$?

low	high	mid	A[mid]	x & A[mid]
1	14	7	24	$84 > 24$
8	14	11	115	$84 < 115$
8	10	9	84	$84 = 84$
8				

* The time complexity of a binary search algorithm is depends upon the height of the binary tree. If the no. of elements are 'n' which is greater than 2^k then the time computation is maximum and is function of logn (f(logn). In sucsesful searches the computing time for binary search tree is best case : o(1)

Average case : o(logn)

Worset case : O(logn)

In unsucessful searches the computing time is o(logn).

* every time we will make one comparison and divide the given numbers n into n/2. The recurense relation is $T(n) = T(n/2) + 1$

Time complexity:

$T(n) = T(n/2) + 1 \longrightarrow (1)$

$T(n/2) = T(n/4) + 1 \longrightarrow (2)$

sub (2) in eq (1)

$T(n) = [T(n/4) + 1] + 1$

$T(n) \Rightarrow T(n/4) + 2 \longrightarrow (4)$

$T(n/4) = T(n/8) + 1 \longrightarrow (3)$

sub (3) in (4)

$T(n) = T(n/8) + 1 + 2$

$T(n) = T(n/8) + 3$

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

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

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

$\Rightarrow T(1) + \log_2 n$

$n = 2^k$

$k = \log_2 n$

Time complexity $\Rightarrow O(\log n)$

	Best-case	AVG-case	worst-case
successful - search	$O(1)$	$O(\log n)$	$O(\log n)$
un-successful - search	$O(\log n)$	$O(\log n)$	$O(\log n)$

② Consider the list of elements are:

1 2 3 4 5 6 7 8

-40, 11, 33, 37, 42, 45, 99, 100

① search element x = 45, n = 8 ?

low	high	mid	A[mid]	x ? A[mid]
1	8	4	37	45 > 37
5	8	6	45	45 = 45

② search element x = 99, n = 8

low	high	mid	A[mid]	x ? A[mid]
1	8	4	37	99 > 37
5	8	6	45	99 > 45
7	8	7	99	99 = 99

Merge sort:-

- Merge sort uses the divide & conquer strategy. In this method division is dynamically carried out. Here I/p array with 'n' elements consisting of 3 steps.

i) Divide:- partition the array / divide the array into 2 sub array. S1 & S2 each is having n/2 elements.

ii) conquer:-

Recursively start S1 & S2.

iii) combine:-

merge S1 & S2 into unique sorted group.

ex:1)

1	2	3	4	5	6	7	8	9	10
65	70	60	75	85	80	55	50	45	57

Soli-

* Here we divide the array into a[1 to 5] & a[6 to 10]
* Let us consider the sub array [1 to 5]
  divide the 1 to 5 sub array as

65	70	60		75	85

* again divide the sub array [1-3] as

65	70		60		75	85

* again divide the sub array [1-2] as

65		70		60

* Now, every set contains only one element.
  combine a[1] and a[2] & sort them.
* combine a[1-2] and a[3] & sort them

60	65	70

* divide a[4-5] as | 75 | | 85 |
* combine a[4] & a[5] | 75 | 85 | & sort them
* combine a[1-3] & a[4-5] & sort them

60	65	70	75	85

* divide a[6-10] as a[6-8] & a[9-10]

80	55	50		45	57

* again divide the sub array [6-8] as

80	55		50

* again divide the sub array [6-7] as [80] [55]
* Now, Every set contains only 1 element. combine a[6] & a[7] & sort them.

55	80

* combine a[6-7] and a[8] & sort them.

50	55	80

* divide a[9-10] as [45] [57]
* combine a[9] & a[10] & sort them [45|57]
* combine a[6-8] & a[9-10] & sovt them

45	50	55	57	80

* combine a[1-5] and a[6-10] then.

45	50	55	57	60	65	70	75	80	85

Merge sort Algorithm time complexity:-

* Merge sort Algorithm uses the process of merging 2 sorted lists recursively to accomplish the sorting of an unordered list.

firstly, 2 sorted arrays / lists of length m & n respectively can we merged in o(m+n) times

if m==n then

o(n+n)

o(2n)

o(n) → The time taken is o(n).

* first, each individual element of the list is considered as a one element sorted list.
* the merge sort algorithm is then used n/2 times to merge pair of elements into sorted list of length '2'.
* this we produced n/2 sorted list each of length 2. It takes n/2 comparisons to accomplish this.
* pairs of 2 elements lists are then merged to produced n/4 sorted list. Each of length 4.
* In this case there are n/4 merge operations. Each taking at most '4' comparisons. This process continous until we have 2 sorted lists of size n/2 & we merge them in o(n) times.
* As there are logn levels of merging.

∴ worst case time complexity of merge sort is o(n logn).

Q:- let T(n) be the time complexity of merge sort the recurrence relation is T(1) = 1 when n=1 for n>1 T(n) = 2 T(n/2) + cn where c is constant.

Ans:-

$$T(n) = 2T(n/2) + cn \rightarrow (1)$$

$$T(n/2) = 2T(n/4) + c(n/2) \rightarrow (2)$$

(2) in (1)

$$T(n) = 2[2T(n/4) + c(n/2)] + cn$$

$$= 4T(n/4) + cn + cn$$

$$T(n) = 4T(n/4) + 2cn \longrightarrow (3)$$

from (1) $T(n/4) = 2T(n/8) + c(n/4) \longrightarrow (4)$

from (3) sub (4) in (3) $T(n) = 4T[2T(n/8) + c(n/4)] + 2cn$

$T(n) \Rightarrow 8T(n/8) + cn + 2cn$

$T(n) \Rightarrow 8T(n/8) + 3cn$

$\Rightarrow 2^3 T(n/2^3) + 3cn$

$\Rightarrow 2^k T(n/2^k) + kcn$

put $n = 2^k$

$k = \log_2 n$

$T(n) \Rightarrow n T(n/n) + \log_2 n\, cn$

$\Rightarrow n T(1) + cn \log_2 n$

$\Rightarrow n(1) + cn \log_2 n$ (1 ↓ constant)

$\Rightarrow n + n \log_2 n$

$\Rightarrow 1 + n \log_2 n$ (1 ↓ constant)

$\Rightarrow n \log_2 n$

$\Rightarrow O(n \log_2 n)$

Space complexity:-

In the merge sort for implementing the recursion stack space is required. Here merge sort divides the array into 2 approximately each size subsets The maximum depth of the stack is to logn.

so space complexity is $O(n \log_2 n)$

Algorithm:-

```
Algorithm Merge_sort (low, high)
{
  if (n==1) then
  return;
  else
  {
    if (low < high) then
    mid = (low + high) / 2;
  // consider the floor value of mid
      merge_srt(low, mid);
      merge_srt(mid+1, high);
      combine(low, mid, high);
  }
}
combine(low, mid, high)
{  up := high;
   i := low;
   j := mid+1;
   while(up<=mid) AND (j<=high) do
   {
      if (a[up]<=a[j]) then
      {  temp[i] = a[up],
         up = up+1,
      }
      else
      {  temp[i] = a[j],
         j = j+1,
      }
```

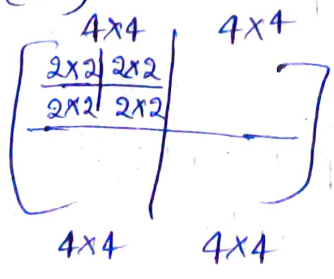
```
        i = i+1;
    }
    if (up > mid) then
        for (t := i to high) do
        {
            temp[i] = a[t]
            i = i+1;
        }
    else
        for t := up to mid do
        {
            temp[i] = a[t];
            i = i+1;
        }
        for t := low to high do
            a[t] = temp[t];
}
```

strassen's matrix multiplication:-

```
Algorithm MATMUL(A, B, C, n)
{
    for i := 1 to n do
    for j := 1 to n do
        C(i,j) := 0
    for k := 1 to n do
        C(i,j) := C[i,j] + A[i,k] * B[k,j];
}
```

The time complexity of the matrix multiplication is $O(n^3)$.

Eg:-



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

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

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

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

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

Strassen's formulas:-

$P = (A_{11} + A_{22})(B_{11} + B_{22})$

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

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

$S = A_{22}(B_{21} - B_{11})$

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

$U = (A_{21} - A_{11})(B_{11} + B_{12})$

$V = (A_{12} - A_{22})(B_{21} + B_{22})$

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

$C_{12} = R + T$

$C_{21} = Q + S$

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

eg:- ① $\begin{matrix} A_{11} & A_{12} \\ \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix} \\ A_{21} & A_{22} \end{matrix}$

$\begin{bmatrix} 0+1 & 0+1 \\ 0+1 & 0+1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$

formula:-

$P = (1+1)(0+1) = 2(1) = 2$

$Q = (1+1)0 = 0$

$R = 1(0-1) = -1$

$S = 1(1-0) = 1$

$T = (1+1)1 = 2$

$U = (1-1)(0+0) = 0$

$V = (1-1)(1+1) = 0$

$C_{11} = P+S-T+V = 2+1-2+0 = 1$

$C_{12} = R+T = -1+2 = 1$

$C_{21} = Q+S = 0+1 = 1$

$C_{22} = P+R-Q+V = 2-1-0+0 = 1$

$\Rightarrow \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$

* Here 2x2 matrix required 7 multiplication & 18 Addition (or) subtraction operations.
* where as General matrix multiplication requires 8 multiplications & 4 Addition operations.
* strassen's reduced one multiplication operation.

② $\begin{matrix} A_{11} & A_{12} \\ 1 & 1 \\ 1 & 1 \\ A_{21} & A_{22} \end{matrix} * \begin{matrix} B_{11} & B_{12} \\ 1 & 1 \\ 0 & 1 \\ B_{21} & B_{22} \end{matrix}$

$\Rightarrow \begin{bmatrix} 1+0 & 1+1 \\ 1+0 & 1+1 \end{bmatrix}$

$\Rightarrow \begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix}$

$P = (A_{11}+A_{22})(B_{11}+B_{22})$

$= (1+1)(1+1)$

$\Rightarrow (2)(2)$

$\Rightarrow 4$

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

$\Rightarrow (1+1)(1) \Rightarrow (2)(1)$

$\Rightarrow 2$

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

$\Rightarrow 1(1-1) \Rightarrow 1(0) \Rightarrow 0$

$S = A_{22}(B_{21}-B_{11}) = 1(0-1)$

$\Rightarrow -1$

$T = (A_{11}+A_{12})B_{22} = (1+1)(1)$

$\Rightarrow (2)(1) = 2$

$U = (1-1)(1+1) \Rightarrow 0$

$V = (1-1)(0+1) = 0$

$C_{11} = P+S-T+V \Rightarrow 4+(-1)-2+0 \Rightarrow 1$

$C_{12} = R+T \Rightarrow 0+2 \Rightarrow 2$

$C_{21} = Q+S \Rightarrow 2+(-1) \Rightarrow 1$

$C_{22} = P+R-Q+U \Rightarrow 4+0-2+0 \Rightarrow 2$

Time complexity for strassen's:-

$n = 2^k$

$k = \log_2 n$

$T(n) = 7T(n/2) + cn^2 \longrightarrow (1)$

$T(n/2) = 7T(n/4) + c(n/2)^2 \longrightarrow (2)$

sub (2) in eq (1)

$T(n) = 7[7T(n/4) + c(n/2)^2] + cn^2$

$T(n) = 49T(n/4) + 7c(\frac{n}{2})^2 + cn^2 \longrightarrow (3)$

$T(n/4) = 7T(n/8) + c(n/4)^2 \longrightarrow (4)$

sub (4) in eq (3)

$T(n) = 49\left[7T(n/8) + c(n/4)^2\right] + 7c(\frac{n}{2})^2 + cn^2$

$T(n) = 343T(n/8) + 49c(n/4)^2 + 7c(\frac{n}{2})^2 + cn^2$

$T(n) = 7^3T(n/2^3) + 7^2c(\frac{n^2}{16}) + cn^2(7/4) + cn^2$

$T(n) = 7^3T(n/2^3) + cn^2(\frac{7}{4})^2 + cn^2(\frac{7}{4}) + cn^2$

$\vdots$

$= 7^kT(n/2^k) + cn^2\left[(7/4)^{k-1} + (7/4)^{k-2} + \cdots\right.$

$\Rightarrow 7^kT(n/2^k) + cn^2(7/4)^k + (7/4) + 1\Big]$

$\Rightarrow 7^kT(n/2^k) + cn^2(\frac{7^k}{4^k})$ put $n = 2^k$

$\Rightarrow 7^{\log_2 n}T(2^k/2^k) + cn^2\left(\frac{7^{\log_2 n}}{4^{\log_2 n}}\right)$

$$\Rightarrow 7^{\log_2 n} T(1) + cn^2 \frac{7^{\log_2 n}}{4^{\log_2 n}}$$

$$\Rightarrow 7^{\log_2 n} \cdot T(1) + cn^2 \frac{n^{\log_2 7}}{n^{\log_2 4}}$$

$$\Rightarrow n^{\log_2 7} + cn^2 \frac{n^{\log_2 7}}{n^2}$$

$$\Rightarrow (1+c) n^{\log_2 7}$$

Time complexity is $O\left(n^{\log \frac{7}{2}}\right)$ or $O(n^{2.81})$

(3) $\begin{bmatrix} A_{11} & A_{12} \\ 0 & 1 \\ 1 & 0 \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ 1 & 0 \\ 0 & 1 \\ B_{21} & B_{22} \end{bmatrix}$

$\begin{bmatrix} 0+0 & 0+1 \\ 1+0 & 0+0 \end{bmatrix} \Rightarrow \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

$P = (0+0)(1+1) \Rightarrow 0(2) = 0$

$Q = (1+0)1 \Rightarrow 1$

$R = 0(0-1) \Rightarrow 0$

$S = 0(0-1) \Rightarrow 0$

$T = (0+1)1 \Rightarrow 1$

$U = (1-0)(1+0) \Rightarrow 1$

$V = (1-0)(0+1) \Rightarrow 1$

$C_{11} = 0+0-1+1$
$\Rightarrow 0$

$C_{12} = 0+1 \Rightarrow 1$

$C_{21} = Q+S = 1+0$
$\Rightarrow 1$

$C_{22} = 0+0-1+1$
$\Rightarrow 0$

④ COMPUTER SCIENCE (merge sort by alphabetical order)

sol:-

i) Divide the array into two sub array of a[1-8] and a[9-15]

ii) Now again divide the sub array a[1-8] into two sub arrays.

a[1-4] to a[5-8]

Comp UTER

iii) Now again the array a[1-4] is divided into two parts.

a[1-2] to a[3-4]

CO mp

iv) Now again divide the arrays into single elements.

a[1] a[2] a[3] a[4]

C O m p

v) Now combine the arrays and sort them.

a[1-2] & a[3-4]

CO mp

vi) Now combine the arrays and sort them.

a[1-4]

CmOp

vii) Now again divide the sub array a[5-8] into two parts. a[5-6] to a[7-8]

UT ER

viii) Now again divide the sub array into two parts a[5] to a[6] } a[7] a[8]

U T } E R

ix) Now combine & sort them. a[5] & a[6]

a[5-6]

TU

x) Now combine & sort them. a[7] & a[8]

a[7-8]

ER

xi) Now combine & sort them. a[5-8]

ERTU

xii) Now combine & sort them. a[5-8] & a[1-4]

CEMOPRTU

Now let us consider the second sub array.

* divide the array into two parts

SCIE NCE

* divide the array SCIE into two parts

SC IE

* divide the array SC & IE into single elements.

S C I E

* combine the elements & sort them.

C S E I

* combine the elements & sort them.

CEIS

* combine the element & sort them

CEN

* combine and sort the elements CEIS & CEN

CCEEEINS

* Now combine and sort the elements.

CEMOPRTU & CCEEEINS

CCCEEEEImnOPRSTU

(5) 1 2 3 4 5 6 7 8 9
CEIMNOPRS binary search (search 'S'):

sol:- step 1:- we have to sort the elements.

low	high	mid	A[mid]	x ? A[mid]
1	9	5	N	S > N
6	9	7	P	S > P
8	9	8	R	S > R
9	9	9	S	S = S

low	high	mid	A[mid]	x ? A[mid]
1	9	5	N	U > N
6	9	7	P	U > P
8	9	8	R	U > R
9	9	9	S	U > S

Not found

Quick sort :-

0 1 2 3 4 5 6 7 8 9 10

① (44), 33, 11, 55, 77, 90, 40, 60, 99, 22, 88, 66.

P ←

[22], 33, 11, 55, 77, 90, 40, 60, 99, (44), 88, 66

→ ←

22, 33, 11, (44), 77, 90, 40, 60, 99, [55], 88, 66

←

22, 33, 11, [40], 77, 90, (44), 60, 99, 55, 88, 66

→

22, 33, 11, 40 | (44) | 90, [77] 60, 99, 55, 88, 66

L-I L-II ←

P (22), 33, 11, 40 ←

[11], 33, (22), 40

→ 11, (22), [33], 40

P (90), 77, 60, 99, 55, 88, 66. ←

[66], 77, 60, 99, 55, 88, (90) ←

→ 66, 77, 60, (90), 55, 88, [99] ←

66, 77, 60 [88], 55 | (90) | 99

→ L-III

P (66), 77, 60, 88, 55 ←

→ [55], 77, 60, 88, (66)

55, (66), 60, 88, [77] ←

55, [60], | (66) | 88, 77

77 88

all the elements are sorted combine them.

11, 22, 33, 40, 44, 55, 60, 66, 77, 88, 90, 99

0 1 2 3 4 5 6 7

(2) (5) 3, 8, 1, 4, 6, 2, 7
p ←

→ [2], 3, 8, 1, 4, 6, (5), 7

2, 3, (5), 1, 4, 6, [8], 7 ←

→ 2, 3, [4], 1, (5), 6, 8, 7
I_1 I_2

p (2), 3, 4, 1 ←

[1], 3, 4, (2)

→ 1, | (2) | 4, [3] ←
I_3 I_4
3, 1

p (6), 8, 7 ←
I_5

(8), 7
p ←
7, 8
I_6

all the elements are sorted combine them

1, 2, 3, 4, 5, 6, 7, 8.