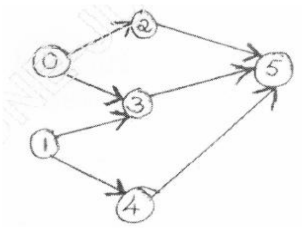
Internal Assessment Test 2 – June 2021

| Sub: | Design & Analysis of Algorithms | Sub Code: | 18CS42 | Branch: | ISE |
|---|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| 1 | (a) Discuss divide and conquer technique with its control abstraction and recurrence relation.<br>Controlabstraction-2,recurrence relation-2<br>(b) State and explain master theorem to solve the recurrence equation.<br>Theorem with 3 cases-3<br>(c) Solve the following recurrence relation using master theorem.<br>i) T(n) = 2 T(n/2)+n ii) T(n) = T(n/2) + 1.<br>1.5 for each problem | [4+3+3] | CO1 | L2 |
| 2 | (a) Write an algorithm for merge sort.<br>Algorithm with explanation- 4<br>(b) Sort the following elements using Merge Sort, 70, 20, 30, 40, 10, 50, 60.Write the recursion tree.<br>Steps with three-4<br>(c) Derive the best case, worst case, average case time efficiency of merge sort algorithm<br>Analysis-3 | [3+4+3] | CO1, C02 | L3 |
| 3 | (a) Consider the numbers given below.<br>106, 117, 128, 134, 141, 91, 84, 63, 42.Trace the partitioning algorithm of quick sort and Place 106 in its correct position. Show all the steps clearly.<br>Quicksort partition steps-5<br>(c) Write an algorithm for quick sort and derive the worst, best, and average case complexity for the same.<br>Algorithm 2.5<br>Analysis-2.5 | [5+5] | CO1, C02 | L3 |
| 4 | (a) Apply Strassen's matrix multiplication to multiply following matrices. Compare its performance with direct matrix multiplication method.<br>$\begin{bmatrix} 4 & 3 \\ 2 & 1 \end{bmatrix} \times \begin{bmatrix} 2 & 5 \\ 1 & 6 \end{bmatrix}$<br>Calculation-2<br>Equations-2<br>Comparison-1<br>(b) What are the three major variations of decrease and conquer technique? Explain with an example for each case.<br>3 cases+eg=5 | [5+5] | CO1 | L2 |

| | | | | |
|---|---|---|---|---|
| 5 | (a) Apply greedy method to obtain an optimal solution to the knapsack problem given knapsack size M = 60, its weights (w1, w2, w3, w4, w5) = (5, 10, 20, 30, 40), its Profits (p1, p2, p3, p4, p5) = (30, 20, 100, 90, 160). Find the total profit earned.<br><br>(b) Using dynamic programming, solve the following knapsack instance: 3 objects, its weights [w1,w2,w3]= [1, 2,2] , its profits [p1,P2,P3] =[18, 16,6] and knapsack size M=4 | [5+5] | CO3 | L3 |
| 6 | (a) What is topological sorting? Apply the same to the below graph<br><br>Definition-1+problem-4<br>(b) Why Merge Sort is preferred over Quick Sort for Linked Lists? | [5+5] | CO1 | L3 |

**solution**

Q1)a) Divide and conquer technique.

→ It is a best known general algorithm design according to:

(i) given a function to compute on 'n' inputs the divide & conquer suggests splitting the inputs into 'k' distinct subsets, $1 < k <= n$, resulting into 'k' sub problems.

(ii) These sub problems must be solved, and then a method must be found to combine sub solutions into a solution of the whole.

Control Abstraction:

```
Algorithm DAandC(P)
{
  if small(P) then return S(P);
  else
  {
    divide P into smaller instances P1, P2 . . . Pk, k >= 1;
    Apply DAndC to each of these instances;
    return Combine (DAndC(P1), DAndC(P2), . . ., DAndC(Pk));
  }
```

→ Initially we call DAndC(P), where 'P' is the problem to be solved.

→ Small (P) is for small problem, problem the spiliting of problem is not required and the function 'S' is invoked.

Otherwise, the problem P is divided into sub problem. Then the sub problems are solved.

→ Combine is a function that determines the solution to P using the solutions of the 'k' sub problems.

Recurrence Relation:

$$T(n) = \begin{cases} g(n) & n \text{ small} \\ T(n_1) + T(n_2) + \cdots T(n_k) + f(n) & \text{otherwise} \end{cases}$$

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

→ Substitution Method: It is used for solving the recurrence relation. This method repeatedly makes substitution for each occurrence of the function T in the right hand side until all such occurrences disappear.

b) Master's theorem

→ The efficiency analysis of many divide and conquer algorithms is greatly simplified by this theorem.

It states, in recurrence equation $T(n) = aT(n/b) + f(n)$,

If $f(n) \in \Theta(n^d)$ where $d \geq 0$.

then,

$$T(n) = \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log_b n) & \text{if } a = b^d \\ \Theta(n^{\log_b a}) & \text{if } a > b^d. \end{cases}$$

example:- $A(n) = 2A(n/2) + 1$

here $a = 2$ $b = 2$ $f(n) = 1 = n^d$

$\underline{\underline{d = 0}}$

$\therefore$ $2 > 2^0$, $a > b^d$.

$\therefore A(n) = \Theta(n^{\log_b a}) = \Theta(n^{\log_2 2}) = \underline{\underline{\Theta(n)}}$.

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

here $a = 2$, $b = 2$, $f(n) = n^1 = n^d$

$2 = 2^1$ $a = b^d$. $\underline{\underline{d = 1}}$

$\therefore T(n) = \Theta(n^d \log_b n) = \Theta(n \log_2 n)$ //

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

here $a = 1$, $b = 2$, $f(n) = 1^0 = n^d$

$1 = 2^0$, $a = b^d$. $\underline{\underline{d = 0}}$

$\therefore T(n) = \Theta(n^d \log_b n) = \Theta(n^0 \log_2 n)$

$= \Theta(\log_2 n)$ //.

Q 2)a) <u>Algorithm.</u>

Mergesort (A[0.....n-1]).

// Input : an array

// output :- sorted array in ascending order.

if $n > 1$

copy A[0.....⌊n/2⌋ - 1] to B[0...⌊n/2⌋ - 1]

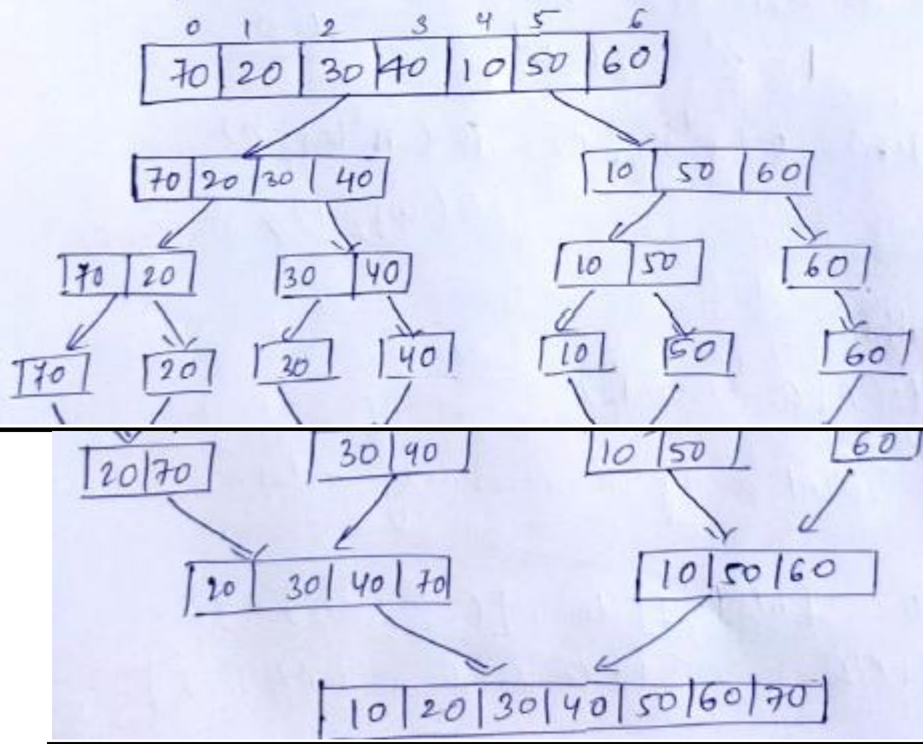
copy A[⌊n/2⌋ ..... n-1] to C[0...⌊n/2⌋ - 1]

```
Mergesort (B[0.....⌊n/2⌋-1])
Merge Sort (C[0.....⌊n/2⌋-1])
Merge (B,C, A).
```

→Merge (B [0... p-1], C [0.....q-1], A [0.... p+q-1])

```
i←0; j←0; k←0
while i<p and j<q do
    if B[i] ≤ C[j]
        A[k] ← B[i]; i ← i+1
    else A[k] ← C[j]; j ← j+1
    k ← k+1
if i=p
    copy C[j....q-1] to A[k....p+q-1]
else copy B[i....p-1] to A[k....p+q-1]
```

b)



c) Time efficiency.

Basic operation → Key comparism.

The efficiency is same in all three cases.

$D(n) = O(1) \Rightarrow$

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

$f(n) = n^d$

$d = 1$

$a = 2 \quad b = 2$

$a = b^d$

$$\therefore T(n) = \Theta(n^d \log_b n)$$
$$= \Theta(n \log_2 n)$$

---

Q 3) a)

| i | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 106 | 117 | 128 | 134 | 141 | 91 | 84 | 63 | 42 |

→ step 1.

pivot

| | i | | | | | | | j |
|---|---|---|---|---|---|---|---|---|
| 106 | 117 | 128 | 134 | 141 | 91 | 84 | 63 | 42 |

swap A[i] and A[j]

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 106 | 42 | 128 | 134 | 141 | 91 | 84 | 63 | 117 |
| | | i | | | | | j | |

swap A[i] and A[j]

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 106 | 42 | 63 | 134 | 141 | 91 | 84 | 128 | 117 |
| | | | i | | | j | | |

Swap A[i] & A[j]

| | | | | i | j | | | |
|---|---|---|---|---|---|---|---|---|
| p:106 | 42 | 63 | 84 | 141 | 91 | 134 | 128 | 117 |

Swap A[i] and A[j]

| | | | | j | i | | | |
|---|---|---|---|---|---|---|---|---|
| 106 | 42 | 63 | 84 | 91 | 141 | 134 | 128 | 117 |

Since i > j

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 91 | 42 | 63 | 84 | 106 | 141 | 134 | 128 | 117 |

Pivot

Q3) b) <u>Algorithm</u>

```
Quicksort(A[l..r])
if l<r
    s ← Parition(A[l..r])
    Quicksort(A[l...s-1])
    Quicksort(A[s+1...r])

Parition(A[l..r]).
p ← A[l]
i ← l; j ← r+1
repeat
    repeat i ← i+1 until A[i] ≥ p
```


```
repeat
    repeat i ← i+1 until A[i] ≥ p
```


```
    repeat j ← j-1 until A[j] ≤ p
    swap (A[i], A[j])
until i ≥ j
swap(A[i], A[j])
swap (A[l], A[j])
return j
```

3)b) Complexity

average:

$$C_{avg}(n) \approx 2n\ln n \approx 1.39 n\log_2 n$$

Worst

$$C_{worst}(n) = n+1+n+n-1+\dots\dots+3$$

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

$1+2+\dots n$

$n\frac{(n+1)}{2}$.

$\therefore \Theta(n^2)$.

Best

$$C_{best}(n) = 2C_{best}(n/2)+n \qquad f(n) = n = n^d$$

$a=2 \quad b=2 \qquad d=1$

$2 = b^d = 2^1$

$\therefore$ equal

$$C_{best}(n) = \Theta(n^d \log_b n)$$

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

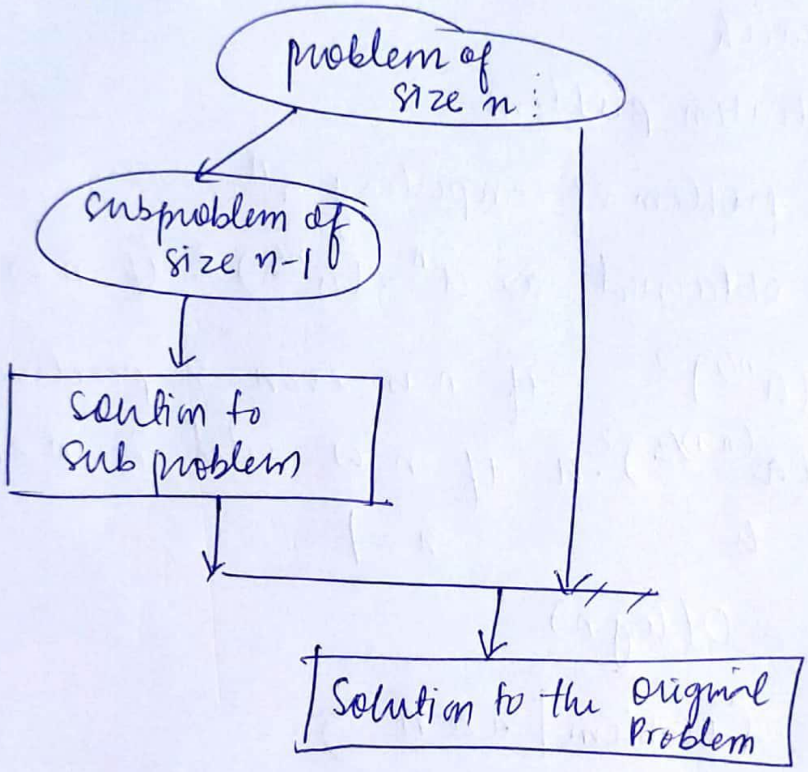
Q5) b) Three major variations of decrease & conquer technique,

(i) Decrease by a constant.

→ Here the size of problem instance is decreased by same constant on each iteration of algorithm. Typically this constant is equal to 1.

eg:- i) Insertion sort.

:- ii) Exponentiation problem.




Exponentiation:

→ Consider a problem of computing $a^n$.

Now $a^n$ can be obtained as $a^n = a^{n-1} \cdot a$.

Hence sol. for problem instance of size n can be obtained by finding sol. of problem instance of size n-1.

Consider $f(n) = a^n$ which can be computed using top-down approach. by using Recursion.

$$f(n) = \begin{cases} f(n-1) \cdot a & \text{if } n > 1 \\ a & \text{if } n = 1 \end{cases}$$

Or bottom up approach by multiplying a with itself for $n-1$ terms.

(ii) Decrease by a constant factor

→ each the size of problem instance is reduced by same constant. Factor on each iteration of algo.

Typically the const factor is 2.

Here the size of the instance is divided by 2.

eg: i) Binary search

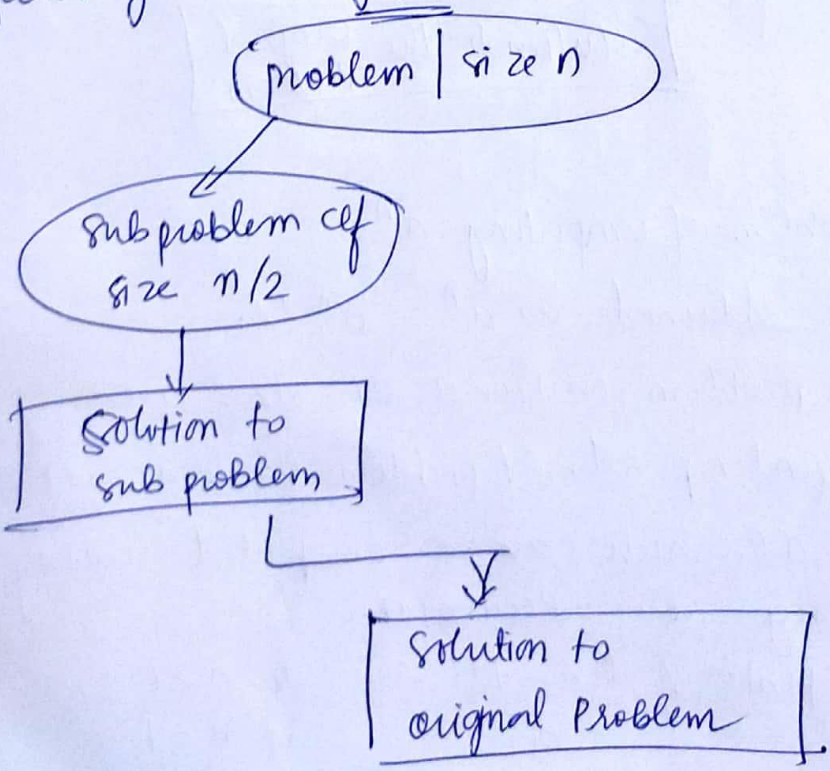
ii) Exponentiation problem.

→ Consider a problem of computing $a^n$.

Now $a^n$ can be obtained as $a^n = (a^{n/2})^2$ if $n$ is even.

$$\therefore a^n = \begin{cases} (a^{n/2})^2 & \text{if } n \text{ is even \& positive} \\ (a^{(n-1)/2})^2 \cdot a & \text{if } n \text{ is odd \& } n>1 \\ a & n=1 \end{cases}$$

→ efficiency $= O(\log n)$



(iii) Variable size decrease:

Here the reduction of problem instance size varies from one iteration of an algo. to another.

eg:- Euclid's algorithm.

→ Here it is used to find the greatest common divisor of values m and n.

Formula used:-

$$gcd(m,n) = gcd(n, m \bmod n)$$

→ when m mod n equals 0, return the last value of m as GCD of m and n.

ex:- $gcd(60,24) = gcd(24,12) = gcd(12,0) = \underline{\underline{12}}$

4)a) $\begin{bmatrix} 4 & 3 \\ 2 & 1 \end{bmatrix} \times \begin{bmatrix} 2 & 5 \\ 1 & 6 \end{bmatrix}$

$m_1 = (a_{00} + a_{11}) * (b_{00} + b_{11})$
$= (4+1) * (2+6) = 5*8 = 40$

$m_2 = (a_{10} + a_{11}) \times b_{00}$
$= (2+1) \times 2 = 3 \times 2 = 6$

$m_3 = a_{00} \times (b_{01} - b_{11}) = 4 \times (5-6) = 4 \times (-1) = -4$

$m_4 = a_{11} \times (b_{10} - b_{00}) = 1 \times (1-2) = 1 \times (-1) = -1$

$m_5 = (a_{00} + a_{01}) \times b_{11} = (4+3) \times 6 = 7 \times 6 = 42$

$m_6 = (a_{10} - a_{00}) \times (b_{00} + b_{01}) = (2-4) \times (2+5)$
$= (-2) * 7 = -14$

$m_7 = (a_{01} - a_{11}) \times (b_{10} + b_{11}) = (3-1) * (1+6)$
$= 2 * 7 = 14$

$$\begin{bmatrix} 4 & 3 \\ 2 & 1 \end{bmatrix} \times \begin{bmatrix} 2 & 5 \\ 1 & 6 \end{bmatrix} = \begin{bmatrix} m_1 + m_4 - m_5 + m_7 & m_3 + m_5 \\ m_2 + m_4 & m_1 + m_3 - m_2 + m_6 \end{bmatrix}$$

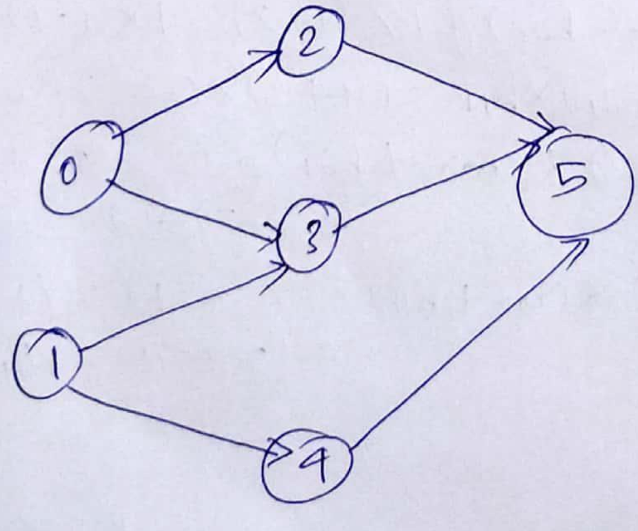
$$= \begin{bmatrix} 40 - 1 - 42 + 19 & -4 + 42 \\ 6 - 1 & 40 - 4 - 6 - 14 \end{bmatrix}$$

$$= \begin{bmatrix} 11 & 38 \\ 5 & 16 \end{bmatrix}$$

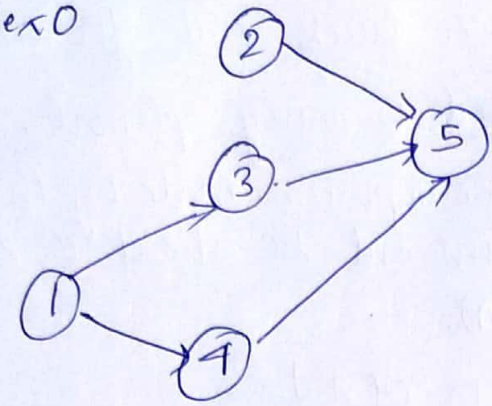
→ Here, for multiplying two 2×2 matrices, using Strassen's algorithm makes $\underline{\underline{7}}$ multiplications and $\underline{\underline{18}}$ additions/subtractions.

→ However, the direct matrix multiplication would require eight multiplications & four additions.

Q6)a) Topological sorting :-
It is a process of ordering of a directed graph is a linear ordering of its vertices such that for every directed edge uv from vertex u to vertex v, u comes before v in the ordering.



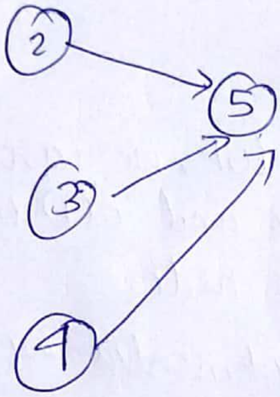
Remove vertex 0



output

0

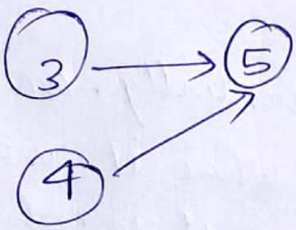
Remove 1



output

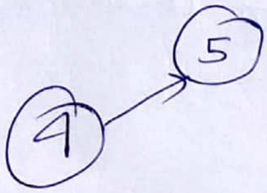
0 1

Remove 2



0, 1, 2

Remove 3



0, 1, 2, 3

Remove 4

output

0, 1, 2, 3, 4.

Finally Remove 5.

∴ The order list of elements will be

0, 1, 2, 3, 4, 5

6)b) Merge sorted preferred over Quick sort for Linked Lists:

(i) no. of comparisons performed is nearly optimal.

(ii) For large n, the no. of comparisons made by this algorithm in the average case turns out be about 0.25n less and hence is also O(n log n).

(iii) It will never degrade to $O(n^2)$.

(iv) It is stable algorithm

→ Unlike array, in linked list, we can insert items in the middle in O(1) extra space and O(1) time if we are given reference/pointer to previous node.

∴ Merge sort can be implemented without extra space for linked lists.

→ Merge sort access data sequentially, and therefore it is faster unlike quick sort where it would have to seek and read every item it wants to compare.