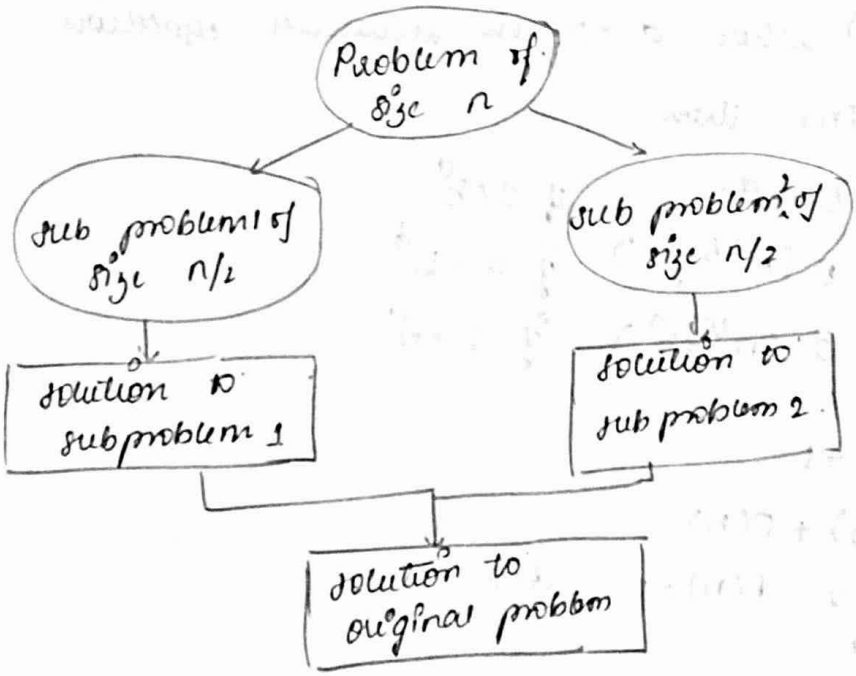
06.08.24
Monday.

MODULE-2

DIVIDE AND CONQUER

* Divide and conquer method of designing the algorithm is one of the best known method of solving a method it is a top down technique for designing algorithms. DAC method divides the problem into smaller sub problems considering that the solutions of the sub problems are easier to find.

Solution of all smaller problems are then combined to get a solution for the original problem.



* As an example consider the problem of computing the sum of n numbers $a_0 \dots a_{n-1}$ if $n > 1$ we can divide the problem into instance of the same problem to compute the sum of first $n/2$ numbers & to compute the sum of the remaining $n/2$ numbers. Once each of these sum is computed we can add their values to get the sum.

$$a_0 \dots a_{n-1} = (a_0 \dots a_{n/2-1}) + (a_{n/2} \dots a_{n-1})$$

A problem of instance of size n is divided into a instances of size $n/2$
An instance of size n can be divided into several instance of n/b with 'a' of them need to be solved.

$$T(n) + aT(n/b) + f(n)$$

— Time req. to div & conquer

$$T(n) = \begin{cases} 0 & \text{if } n=1 \\ T(n/2) + T(n/2) + 1. & \text{otherwise.} \end{cases}$$

General DAC recurrence → $T(n) = a\,T(n/b) + F(n)$

$F(n)$ is a function that accounts for the time spent on dividing the problem into smaller ones and to combine their results.

* MASTER'S THEOREM.

If $T(n) \in \Theta(n^d)$ where $d \geq 0$ in recurrence equation

$T(n) = aT(n/b) + F(n)$ then

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

Ex1. $T(n) = 2T(n/2) + 1.$

$= aT(n/b) + F(n)$

$a = 2 \quad b = 2 \quad F(n) = 1. \quad d = 0.$

$a \quad b^a$

$2 > 2^0.$

$T(n^{\log_2 2}) = T(n^1) \approx T(n).$

11.05.24
Saturday

* Brute Force

is a straight forward approach to solve a problem usually directly based on the problem statement and definition of the concepts involved.

* Selection sort

Start sorting by scanning the entire given list to find its smallest element and exchange it with the first element putting the element in its final position in the sorted list, then scan the

list from second element to find smallest in n-1 elements and exchange it with second element and putting the element in second smallest element after n-1 pass the list is sorted.

ALGORITHM Selection sort (A[0...n-1])

// Sorts a given array by selection sort

// Input an array A[0...n-1] of orderable elements.

// Output array A[0...n-1] sorted in ascending order.

n.	Local	Time Taken

for i ← 0 to n-2 do

    min ← i

    for j ← i+1 to n-1 do

        if A[j] < A[min].

            min ← j

    swap A[i] & A[min]

Ex.

0	1	2	3	4	5	6
89	45	68	90	29	34	17.
17 |	45	68	90	29	34.	89.
17	29 |	68	90	45	34	89.
17	29	34	90	45	68	89.
17	29	34 |	45	90	68	89.
17	29	34	45 |	68	90	89.
17	29	34	45	68 |	89	90.

n = 7.

n - 1 = 6.

n	Time taken
5	—
10	—
50.	—
100	—

Learn selection sort

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1.$$

$$= \sum_{i=0}^{n-2} [n-1-(i+1)+1] = \sum_{i=0}^{n-2} [n-1-i-1+1]$$

$$= \sum_{i=0}^{n-2} (n-1-i) = \frac{n(n-1)}{2} \approx \theta(n^2).$$

Bubble sort

Sort by comparing adjacent elements of the list and exchange them if they are out of order.

Ex: 89 45 68 90 29 34 17

45 89 68 90 29 34 17

45 68 89 90 29 34 17

45 68 89 90 29 34 17

45 68 89 29 90 34 17

45 68 89 29 34 90 17

45 68 89 29 34 17 | 90 → first pass ends

45 68 89 29 34 17 90

45 68 89 29 34 17 90

45 68 29 89 34 17 90

45 68 29 34 89 17 90

45 68 29 34 17 | 89 90 → second pass ends

45 68 29 34 17 89 90

45 29 68 34 17 89 90

45 29 34 68 17 89 90

45 29 34 17 | 68 89 90 → third pass ends

29 45 34 17 68 89 90

29 34 45 17 68 89 90

29 34 17 | 45 68 89 90 → fourth pass ends

29 34 17 45 68 89 90

29 17 | 34 45 68 89 90 → fifth pass ends

17 | 29 34 45 68 89 90 → sixth pass ends

ALGORITHM Bubblesort (A[0..n-1])

// Sorts a given array by bubble sort

// Input an array A[0...n-1] of orderable elements

// Output array A[0..n-1] sorted in ascending order

for i ← 0 to n-2 do

    for j ← 0 to n-2-i do

        if A[j+1] < A[j]

            swap A[j] & A[j+1].

bubble search

selection "

linear "

binary "

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=0}^{n-2-i} 1.$$

$$= \sum_{i=0}^{n-2} [(n-2-i) - 0 + 1].$$

$$= \sum_{i=0}^{n-2} [n-1-i].$$

$$= \frac{n(n-1)}{2} \approx \theta(n^2)$$

→ Selection Sort.

```
import java.util.Scanner;
public class SelectionSort {
    public static void SelectionSort(int[] arr)
    int n = arr.length;
    for (int i = 0; i < n-1; i++) {
        int minIndex = i;
        for (int j = i+1; j < n; j++) {
            if (arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }
    }
```

```
            // swap arr[i] & arr[minIndex]
            int temp = arr[i];
            arr[i] = arr[minIndex];
            arr[minIndex] = temp;
        }
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the size of the array: ");
        int n = sc.nextInt();
        int[] arr = new int[n];
        System.out.println("Enter the elements of the array:");
        for (int i=0; i<n; i++) {
            arr[i] = sc.nextInt();
        }
        long startTime = System.nanoTime();
        SelectionSort(arr);
        long endTime = System.nanoTime();

        System.out.println("Sorted array:");
        for (int i : arr) {
            System.out.print(i + " ");
        }
        System.out.println();

        long timeTaken = endTime - startTime;
        System.out.println("Time taken: " + timeTaken + " nanoseconds");
    }
}
```

→ Bubble Sort

```
import java.util.Scanner;

public class BubbleSort {
    public static void bubbleSort(int[] arr) {
        int n = arr.length;
        for (int i=0; i<n-1; i++) {
            for (int j=0; j<n-i-1; j++) {
                if (arr[j] > arr[j+1]) {
                    int temp = arr[j];
                    arr[j] = arr[j+1];
                    arr[j+1] = temp;
                }
            }
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the size of the array: ");
        int n = sc.nextInt();
        int[] arr = new int[n];

        System.out.println("Enter the elements of the array:");
        for (int i=0; i<n; i++) {
            arr[i] = sc.nextInt();
        }

        long startTime = System.nanoTime();
        bubbleSort(arr);
        long endTime = System.nanoTime();
```

```
        System.out.println("Sorted array:");
        for (int i : arr) {
            System.out.print(i + " ");
        }
        System.out.println();

        long timeTaken = endTime - startTime;
        System.out.println("Time taken: " + timeTaken + " nanoseconds.");

    }
}
```

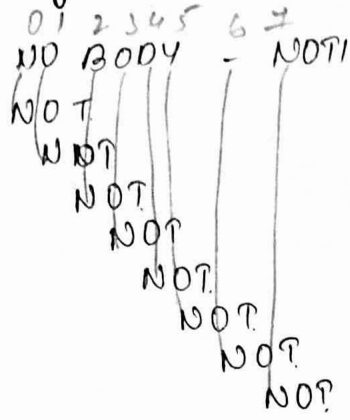
O/p for Selection sort

Number of elements (n)	Time taken nanoseconds
5	12100
10	13100
15	25200
25	48200

O/p for Bubble sort

Number of elements (n)	Time Taken nanoseconds
5	10800
10	12800
15	17000
25	26100

Monday. Brute Force Sequential Search & Brute Force String Matching.



ALGORITHM BruteforceStringmatch (T[0..n-1], P[0..m-1])

// implements brute force string matching
// input an array T(0...n-1) of n characters representing a text and an array P(0...m-1) of m characters representing a pattern.
// Output the index of the first character in the text thats starts a matching substring or -1 if the search is unsuccessful.

```
for i ← 0 to n-m do
    j ← 0
    while j < m and P[j] = T[i+j] do
        j ← j+1
    if j = m return i
return -1
```

$$\sum_{0}^{n-1} \sum$$

ANALYSIS.

Time complexity : $\theta(mn)$

→ Linear search.

```
import java.util.Scanner;

public class Linearsearch {
    public static int linearsearch(int[] arr, int x) {
        for (int i=0; i<arr.length; i++) {
            if (arr[i] == x) {
                return i;
            }
        }
        return -1;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the size of the array: ");
        int n = sc.nextInt();
        int[] arr = new int[n];
        System.out.println("Enter the elements of the array:");
        for (int i=0; i<n; i++) {
            arr[i] = sc.nextInt();
        }
        System.out.print("Enter the element to search: ");
        int x = sc.nextInt();
        long start = System.nanoTime();
        int index = linearsearch(arr, x);
        long end = System.nanoTime();
        if (index != -1) {
            System.out.println("Element found at index: "+
                                index);
```

```
    } else {
        System.out.println("Element not found");
    }
    long timeTaken = end - start;
    System.out.println("Time taken: " + timeTaken + " nanoseconds");
    }
}
```

Output:

```
Enter the size of the array: 5
Enter the elements of the array:
76
34
39
76
33
Enter the element to search: 76
Element found at index: 0
Time taken: 11600 nanoseconds
```

Number of elements (n).	Time Taken (nanoseconds)
5	8900
10	9400
15	10200
20	12000
25	13000

13/5/24

12.05.24
Wednesday.

BINARY SEARCH

$A[0] \dots A[m-1]$ — search here for $k \leq A[m]$

$A[m]$ — mid.

$A[m+1] \dots A[n-1]$ — search here for $k >$ mid

ALGORITHM. BinarySearch $[A[0 \dots n-1], k]$

// Implements non recursive binary search.

// Input: an array $A[0 \dots n-1]$ sorted in ascending order and a search key k.

// Output: An index of array's element that is equal to k or -1 if there no such element.

$l \leftarrow 0 \quad r \leftarrow n-1$

while $(l \leq r)$ do

~~m ← [(l+r)/2] (l+r)/2~~

$m \leftarrow (\lfloor (l+r)/2 \rfloor)$

if $k = A[m] \quad r \leftarrow m-1$

else.

$l \leftarrow m+1$.

return. -1

Time complexity

i) $k =$ mid.

ii) $k <$ mid.

iii) $k >$ mid

i) Best case:

$T(n) = \Omega(1)$.

~~T(2^{k-1}) + 1.~~

~~T(1)~~

ii> Worst case.

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

$n = 2^k$

$k = \log_2 n$.

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

$= T(2^k - 1) + 1$

$= T(2^k - 2) + 1 + 1$

$= T(2^k - 2) + 2$.

$= T(2^k - 3) + 3$.

$\vdots$

$= T(2^k - k) + k$.

$= T(1) + \log_2 n$. $\Rightarrow \{ n = 2^k \Rightarrow k = \log_2 n$

$\approx O(\log_2 n)$

iii> Average case.

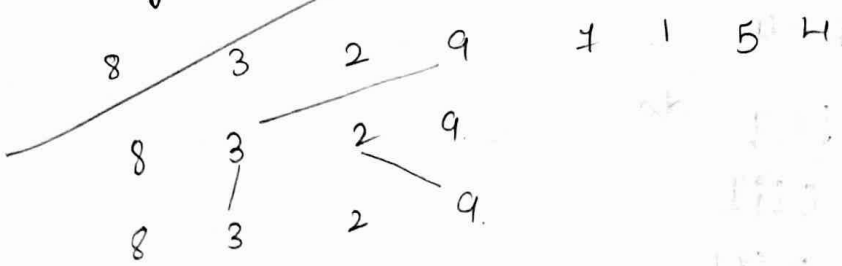
$T(n) \approx \theta(\log_2 n)$

Merge sort:

MERGE SORT

Sort given array $A[0 \ldots n-1]$ by dividing it into two half one is $A[0 \ldots n/2 - 1]$ and $A[n/2 \ldots n-1]$.

Sorting each of them recursively and then merging the two smaller sorted array into a single sorted one.



ALGORITHM mergesort (A [0... n-1])

// Sorts array A[0... n-1] by recursive merge sort.
// Input an array A [0... n-1] of orderable elements
// Output an array A [0... n-1] sorted in non-decreasing.

if n>1

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

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

mergesort (B [0... ⌊n/2⌋ -1)

mergesort (C [0 ... ⌊n/2⌋ -1)

merge (B, C, A)

ALGORITHM merge (B [0... p-1], C [0.. q-1], A [0... p+q-1])

// Merge two sorted arrays into one sorted array.
// Input arrays B [0... p-1] and C [0... q-1] both sorted
// Output sorted array A [0.. p+q-1] of the elements B and C

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 $C[j \ldots q-1]$ to $A[k \ldots p+q-1]$

else.

copy $B[i \ldots p-1]$ to $A[k \ldots p+q-1]$

ANLYSIS.

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

$n = 2^k$.

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

$= 2T(2^{k-1}) + n$.

$= 2[2T(2^{k-2}) + n] + n$

$= 2^2 T(2^{k-2}) + 2n$.

$= 2^k T(2^{k-3}) + 3n$

$\vdots$

$= 2^k T(2^{k-k}) + kn$.

$= 2^k \cdot T(2^0) + kn$.

$= 2^k T(1) + kn$

$= n T(1) + kn$.

$= n + n \log_2 n$

$\approx n \log_2 n$

But case, worst case, average case.

Recurrence Equation for D and C [divide & conquer]

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

→ $T(n)$ time for D and C on any input size n.

$g(n)$ time to compute the answer directly for small input.

$f(n)$ is the time for dividing problem & combining the solution to sub problems.

Time complexity for D & C problem is given by recurrence of the form

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

n - total no of inputs
b - dividing the total by b time
a - no of instances being solved

Here a & b are known constant

$T(1)$ is known and n is a power of b, $(n = b^k)$

$h(n)$	$u(n)$
$O(n^a)$ $a<0$	$O(1)$
$\Theta(\log n)^i$ $i \geq 0$	$\Theta\left((\log)^{i+1}/(i+1)\right)$
$\Omega(n^a)$ $a>0$	$\Theta(h(n))$

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

$a=1 \quad b=2 \quad f(n)=c$

$\log_b^a = \log_1^2 = 0$

$f(n)/n^{\log_b^a} = c/1 = c$

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

2) $T(n) = n^{\log_b^a}\,[f(n) + \Theta(\log n)] = \Theta(\log n)$

$a=2 \quad b=2 \quad f(n)=c$

$\log_b^a = \log_2^2 = 1$

$f(n)/n^{\log_b^a} = cn/n = c$

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

$T(n) = n\,[c + O(\log n)] = \Theta(\log n)$

3> $T(n) = 7\,T(n/2) + 18n^2$

$a = 7 \quad b = 2 \quad f(n) = 18n^2$

$\log_b^a = \log_2^7 = 2.81$

$f(n)/n^{\log_b^a} = 18n^2/n^{2.81}$

$= 18n^{2-2.81}$

$= 18n^{-0.81}$

$= O(1)$

compare with $h(n)$

$k < 0$

$h(n) = O(n^{-0.81})$

$u(n) = O(1)$

$T(n) = n^{2.81}\,[18n^2 + O(1)]$

4> $T(n) = 9\,T(n/3) + 4n^6$

$a = 9 \quad b = 3$

$\log_b^a = \log_3^9 = 2$

$f(n)/n^{\log_b^a} = 4n^6/n^2$

$= 4n^4$

$\Omega(n^4)$

$u(n) = \Theta(h(n)) = \Theta(n^4)$

$T(n) = n^2\,[4n^6 + \Theta(n^4)]$

14.05.24. Finding the maximum & minimum
Guiding.

ALGORITHM Straight max min (a, n, min, max

// Set max to the maximum and min to the minimum of a[1...n]

```
{
  max = min = a[0]
  for i = 1 to n-1 do
  {
     if (a[i] > max) then max = a[i]
    else if (a[i] < min then min = a[i]
  }
}
```

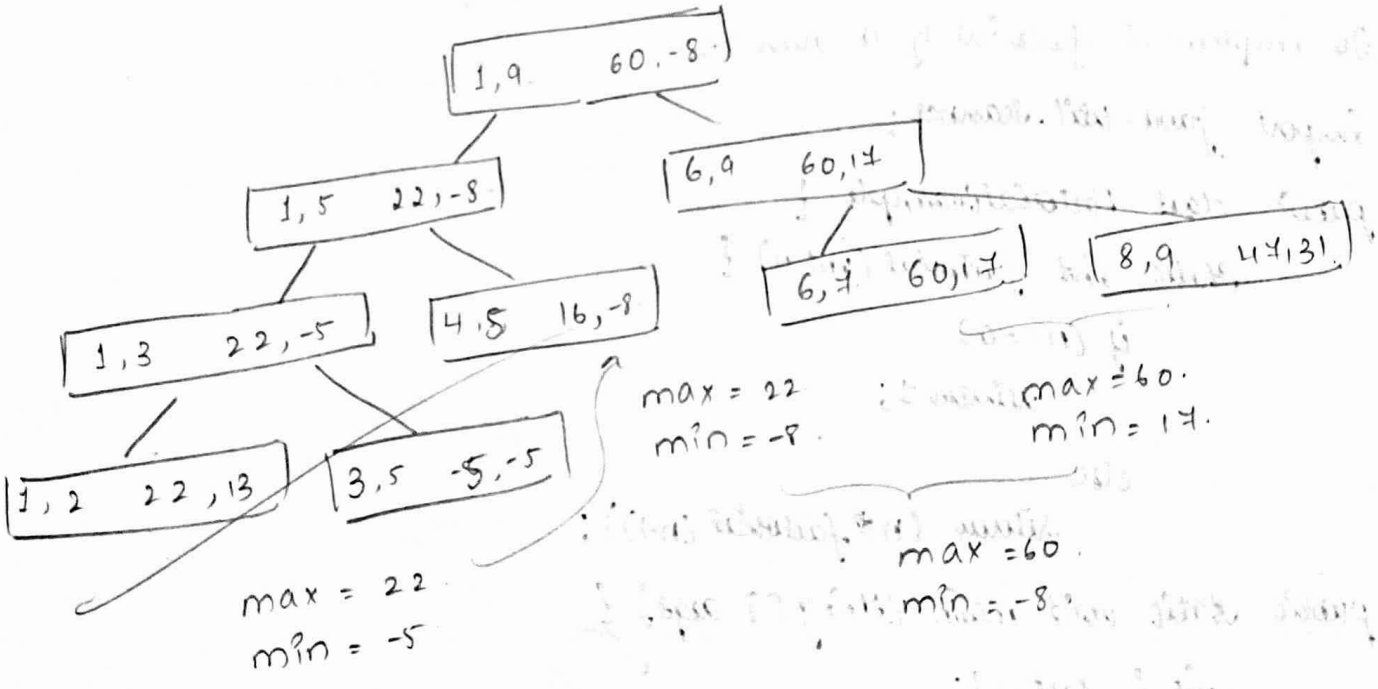
ALGORITHM MaxMin (i, j, max, min)

// A[1...n] is a global array. Parameters i and j are integers $1 \le i \le j \le n$
The effect is to set max & min to the largest & smallest values in A[i:to j] / A[i:j] respectively.

```
{
  if (i=j) the max = min = a[i] // small problem only 1 element
      elseif (i = j-1) then
        {
          if (a[i] < a[j]) then
            {
              max = a[j];
              min = a[i];
            }
          else
            {
              max = a[i];
              min = a[j];
            }
        }
```

```
else
{
    // if P (problem) is not small divide p into sub problems
    // Find where to split the set
        mid = ⌊(i+j)/2⌋
    // solve the sub problems
        MaxMin(i, mid, max, min)
        MaxMin(Mid+1, j, max1, min1)
    // combine the solutions
        if (max < max1) then max = max1
        if (min > min1) then min = min1
}
}
```

	1	2	3	4	5	6	7	8	9
a:	22	13	-5	-8	15	60	17	31	47



Time complexity.

$$T(n) = \begin{cases} 0 & n = 1 \\ 1 & n = 2 \\ T(n/2) + T(n/2) + 2 & \text{otherwise.} \end{cases}$$

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

$a = 2 \quad b = 2 \quad f(n) = 2$

$\log_b^a = \log_2^2 = 1.$

$f(n) / n^{\log_b^a} = 2/n^1 = 2n^{-1} \quad k < 0.$

$h(n) = O(n^{-1})$

$u(n) = O(1)$

$$T(n) = n^{\log_b^a} [f(n) + u(n)].$$

$$= n^1 [2 + O(1)]$$

30.05.24
Thursday

1> To implement factorial of a number

```
import java.util.Scanner;
public class FactorialExample {
    static int factorial(int n) {
        if (n==0)
            return 1;
        else
            return (n*factorial(n-1));
    public static void main(String[] args) {
        int i, fact = 1;
        int number;
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter number");
        number = sc.nextInt();
```

```
        fact = factorial(number);
        System.out.print("Factorial of "+number+" is: "+fact);
    }
}
```

Output: Enter number
5
Factorial of 5 is: 120

2. Implement fibonacci for a given number.

```
import java.util.Scanner;
public class Fibonacci {
    public static void main(String[] args) {
        int n1=0, n2=1, n3, i, count;
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter number ");
        count = sc.nextInt();
        System.out.print("Fibonacci series: "+n1+" "+n2);
        for(i=2; i<count; i++) {
            n3 = n1+n2;
            System.out.print(" "+n3);
            n1 = n2;
            n2 = n3;
        }
    }
}
```

Output: Enter number 10
Fibonacci series: 0 1 1 2 3 5 8 13 21 24

3) Sort a given set of n integer elements using the merge sort method and compute its time complexity. Run the program for varied values of n > 5000, and record the time taken to sort. Plot a graph of the time taken versus non-graph sheet. The elements can be read from a file or can be generated using the random number generator. Demonstrate using Java how the divide and conquer method works along with its time complexity analysis: worst case, avg case and best case.

```
import java.util.Random;
import java.util.Scanner;
public class Mergesort {
    public static void main(String[] args) {
        int a[] = new int[10000];
        Scanner in = new Scanner(System.in);
        long start, end;
        System.out.println("MERGE SORT PROGRAM");
        System.out.println("Enter number of elements to be sorted as");
        for (int i=0; i<n; i++)
            System.out.print(a[i]+" ");
        start = System.nanoTime();
        mergesort(a,0,n-1);
        end = System.nanoTime();
        System.out.println("\nThe sorted elements are");
        for (int i=0; i<n; i++)
            System.out.print(a[i]+" ");
        System.out.println("\n The time taken to sort is "+
            (end-start)+"ns");
}
```

```
static void mergesort (int a[], int low, int high) {
    int mid;
    if (low < high) {
        mid = (low + high) / 2;
        mergesort(a, low, mid);
        mergesort(a, mid+1, high);
        merge(a, low, mid, high);
    }
}

static void merge (int a[], int low, int mid, int high) {
    int i, j, h, k, b[] = new int[100000];
    h = low;
    i = low;
    j = mid+1;

    while ((h <= mid) && (j <= high)) {
        if (a[h] < a[j]) {
            b[i] = a[h];
            h = h+1;
        }
        else {
            b[i] = a[j];
            j = j+1;
        }
        i = i+1;
    }
    if (h > mid) {
        for (k = j; k <= high; k++)
        {
            b[i] = a[k];
            i = i+1;
        }
    }
```

```
        else {
            for (k=h; k<=mid; k++) {
                b[i] = a[k];
                i = i+1;
        } }
        for (k=low; k<=high; k++)
            a[k] = b[k];
    }
}
```

Output:

1> Random number

Input	Time Taken.
5000	74095100 ns.
4000	49148400 ns.
3000	43885300 ns.
6000	595524000 ns.

2> User input.

Input.	Time Taken.
n = 5. Elements : 3 4 4 2 1 After sort : 1 2 3 4 4.	47100 ns.
n = 3. 4 2 6. 2 4 6	38300 ns.

n = 10. 10 9 8 7 6 5 4 3 2 1. 1 2 3 4 5 6 7 8 9 10.	18.6100ns.

O/p seen
2015

03.06.24
Monday.

QUICKSORT.

A[0] ... A[s-1], A[s], A[s+1] ... A[n-1]

all are <= A[s]   all are ≥ A[s]

0 1 2 3 4 5 6 7
5 3 1 9 8 2 4 7.
pivot. i j

5 3 1 4 8 2 9 7
i j

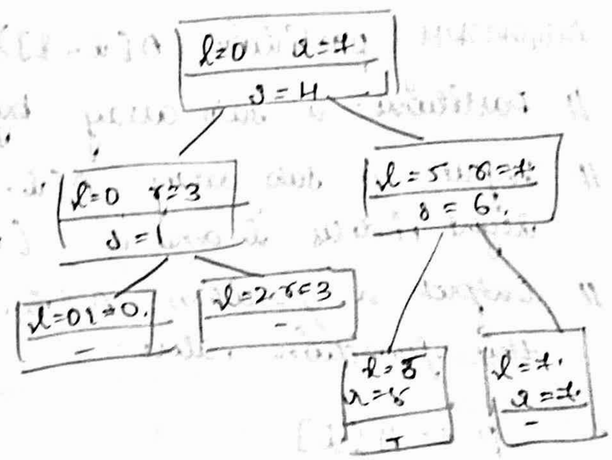
5 3 1 4 2 8 9 7
j i

2 3 1 4 (5) 8 9 7
p i j   p i j

2 1 3 4   (8) 7 9
j i   j i

1 (2) 3 4
p ij

(3) 4
j i



ALGORITHM Quicksort (A[l...r])

// Sorts a sub array by quicksort

// Input a sub array A[l-r] of A[0-n-1] defined by its left & right indices l and r

// Output sub array A[l-r] sorted in non decreasing order

```
if l < r
    s ← partition (A[l..r]) // s is a split position
    Quicksort (A[l...s-1])
    Quicksort (A[s+1...r])
```

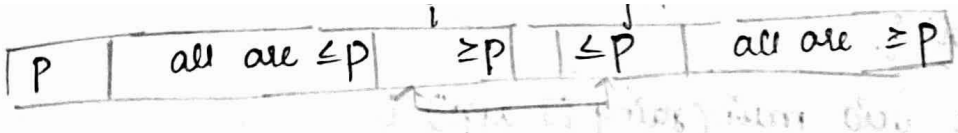
ALGORITHM partition (A[l..r])

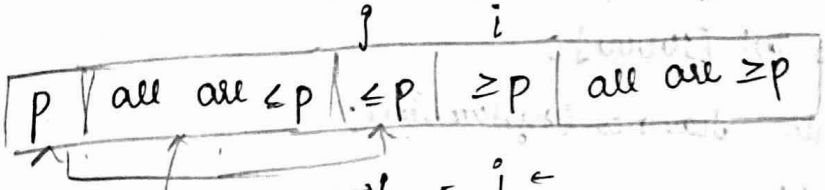
// Partitions a sub array by using its first element as pivot.

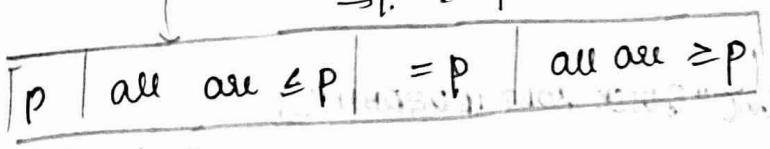
// Input a sub array A[l..r] of A[0..n-1] defined by its left & right indices l and r (l<r)

// Output a partition A[l..r] with the split position returned as this function value.

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

i) 

ii) 

iii) 

→ can combine both crossed over indices i & j by exchanging pivot with A[j] where i ≥ j

→ Time complexity

$T(1) = 0$

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

Best case: $\Theta(n \log n)$

Average case: (random array) $\approx \Theta(n \log n)$

Worst case: $O(n^2)$

↳ split will be skewed to the extreme one of the two sub arrays will be empty while the size of the other will be just one less than the size of the sub array being partitioned.

4) Sort a given set of n integer elements using the quick sort method and compute its time complexity. Run the program for varied values of n > 5000 and record the time taken to sort. Plot a graph of the time taken versus n on graph sheet. The elements can be read from a file or can be generated using the random number generator. Demonstrate using java how the divide and method works along with its time complexity analysis: worst case, avg. case and best case.

```
import java.util.Scanner;
import java.util.Random;
```

```
public class Quicksort {
    public static void main(String [] args) {
        int a[] = new int [10000];
        Scanner in = new Scanner (System.in);
        long start, end;
        System.out.println("QUICK SORT PROGRAM");
        System.out.println("Enter the number of elements to be sorted");
        int n = in.nextInt();
        Random rand = new Random();
        for (int i=0; i<n; i++)
            a[i] = rand.nextInt(100);
        System.out.println("Array elements to be sorted are");
        for (int i=0; i<n; i++)
            System.out.print(a[i]+" ");
        a[n] = 999;
        start = System.nanoTime();
        quicksort(a, 0, n-1);
        end = System.nanoTime();
        System.out.println("\n The sorted elements are");
        for (int i=0; i<n; i++)
            System.out.print(a[i]+" ");
        System.out.println("\n The time taken to sort is "+(end-start)
                           +" ns");
    }
    static void quicksort(int a[], int p, int q) {
        int i;
```

```
        if (p<q) {
            j = partition (a, p, q);
            quicksort(a, p, j-1);
            quicksort (a, j+1, q);
        }
    }
static void partition (int a[], int m, int p) {
        int v, i, j;
        v = a[m];
        i = m;
        j = p;
        while (i<=j){
            while (a[i]<=v)
                i++;
            while (a[j]>v)
                j--;
            if (i<j)
                interchange (a, i, j);
        }
        a[m] = a[j];
        a[j] = v;
        return j;
}
static void interchange (int a[], int i, int j) {
        int p;
        p = a[i];
        a[i] = a[j];
        a[j] = p;
    }
}
```

Output:

1) Random number.

Input:	Time Taken.
5000	1620100 ns.
6000	2813900 ns.
7000	2762400 ns.
8000.	3023900 ns.

2) User input:

Input: Time Taken

n = 5

A = 4 76 75 7 72. 5900 ns.

A = 4 7 72 75 76.

n = 3.

A = 77 41 18 4500 ns.

A = 18 41 77.

n = 10.

A = 53 81 55 14 77
10 17 3 66 17.

6800 ns.

A = 3 10 14 17 53
55 66 76 77 81

```
import java.util.Random;
import java.util.Scanna.

public class GFG {.
    static class Pair {.
        int min;
        int max;
    }.
    Static Pair getMinMax (int arr[], int n) {.
        Pair minmax = new Pair ();
        int i;
        // If there is only 1 element the return it as min and max both
        if (n==1) {.
            minmax.max = arr[0];
            minmax.min = arr[1];
            return minmax;
        }
        // If there are more than one elements, then initialize min & max.
        if (arr[0] > arr[1]) {.
            minmax.max = arr[0];
            minmax.min = arr[1];
        } else {
            minmax.max = arr[1];
            minmax.min = arr[0];
        }
        for (i=2; i<n; i++) {.
            if (arr[i] > minmax.max) {.
                minmax.max = arr[i];
            } else if (arr[i] < minmax.min) {.
                minmax.min = arr[i];
            }
```

```
    }
    return minmax;
}
public static void main(String[] args) {
    int a[] = new int[10000];
    Scanner in = new Scanner(System.in);
    System.out.println("Enter the numbr of elements in an array");
    int n = in.nextInt();
    Random rand = new Random();
    for (int i = 0; i<n; i++)
        a[i] = rand.nextInt(100);
    System.out.print("Array elements are ");
    for (int i=0; i<n; i++)
        System.out.print(a[i] + " ");
    a[n] = 99;
    Pair minmax = getMinMax(a, n);
    System.out.printf("\n Minimum element is %d", minmax.min);
    System.out.printf("\n Maximum element is %d", minmax.max);
    }
}
```

Output:

1> n = 5

A = 12 86 92 73 26.

Max element is 92.

Min element is 12.

2> n = 10.

A = 95 38 72 45 84 73 74 11 69 66.

Max element 95.

Min element 11

05.06.24
Wednesday. Strassen's Matrix multiplication

A and B are two $n \times n$ matrices. $C = AB$ is also $n \times n$ matrix

$$C[i,j] = \sum A(i,k) * B(k,j). \quad i \le k \le n.$$

Time complexity: $\theta(n^3)$

$n = 2^k$ if not add zeros

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

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

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

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

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

Needs 8 multiplication and 4 addition.

$$T(n) = \begin{cases} b & n \le 2 \\ 8\,T(n/2) + cn^2 & n > 2 \end{cases}$$

cn^2 is the time required to add, divide, multiply 2 matrix

b and c are constant

(only applicable for 2x2 matrix.

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

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

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

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

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

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

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

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

$C_{12} = R + T$

$C_{21} = Q + S$

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

7 - multiplication

18 - addition, substraction.

strassen	normal
$\log_2 7$	$\log_2 8$
$2.807 < 3$	

Q.

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

$\Theta(n^3)$ is the time complexity for conventional method.

$\Theta(n^{2.807})$ is the time complexity for Strassen's method.

$A \cdot B = C$.

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 7 & 10 \\ 15 & 22 \end{bmatrix}$$

$P = (1+4)(1+4) = 25$ $\quad T = (1+2) * 4 = 12$

$Q = (3+4)+1 = 7$ $\quad U = (3-1) * (1+2) = 6$

$R = 1 * (2-4) = -2$ $\quad V = (2-4) * (3+4) = -14$

$S = 4 * (3+1) = 8$

$C_{11} = P+S \quad 25+8-12+(-14)$

$C_{12} = -2+12 = 10$.

$C_{21} = 7+8 = 15$.

$C_{22} = 25-2-7 = 22$.

Decrease And Conquer.

1) Decrease by constant
2) Decrease by constant factor.
3) Variable size decrease

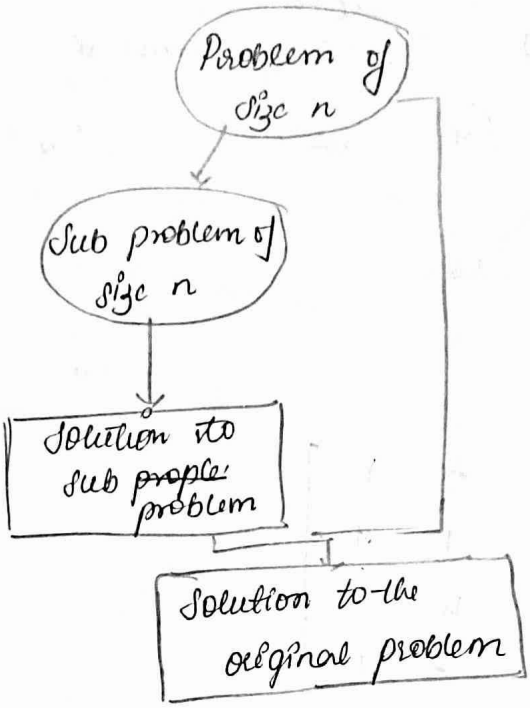
1) Decrease by constant

Size of an instance is reduced by the same constant on each iteration of the algorithm and this constant is always 1.

Eg. a^n reduced by 1.

$a^n = a^{n-1} a$

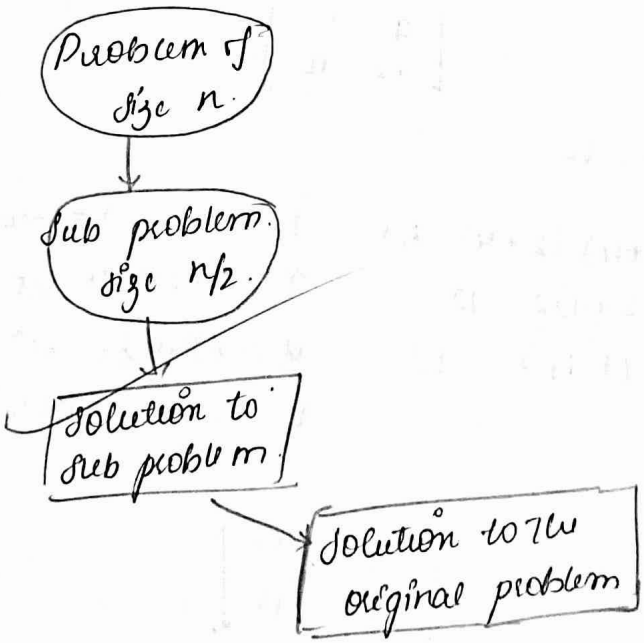
$$f(n) = \begin{cases} f(n-1)\, a & \text{if } n>1 \\ a & \text{if } n=1. \end{cases}$$



2) Decrease by constant factor

Reduce the problems instance by the same constant factor on each iteration of algorithm., this constant factor is equal to 2.

eg. $a^n = a^{n/2} \cdot a^{n/2}$.



3) Variable size decrease

Size reduction pattern varies from one iteration of the algorithm to another.

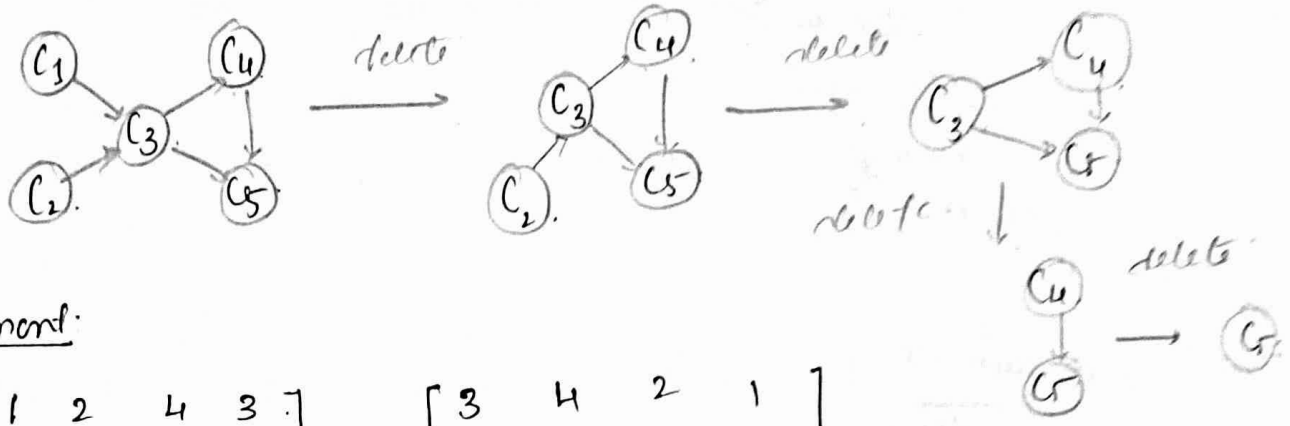
Eg. $gcd(m,n) = gcd(n, m \bmod n)$

Topic: TOPOLOGICAL SORTING:

→ Uses DAG (Direct Acyclic graph) → DFS.

→ source removal.

source removal method.



Assignment:

$$\begin{bmatrix} 1 & 2 & 4 & 3 \\ 1 & 3 & 2 & 4 \\ 4 & 1 & 2 & 3 \\ 1 & 2 & 3 & 4 \end{bmatrix} * \begin{bmatrix} 3 & 4 & 2 & 1 \\ 3 & 2 & 1 & 4 \\ 3 & 2 & 4 & 1 \\ 4 & 3 & 2 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 \\ 1 & 8 \end{bmatrix} * \begin{bmatrix} 3 & 4 \\ 3 & 2 \end{bmatrix}$$

$P = (1+3)(3+2) = 20$

$Q = (1+3)(3) = 12$

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

$S = 3(3-3) = 0$

$T = (1+2)(2) = 6$

$U = (1-1)(3+4) = 0$

$V = (2-3)(3+2) = -5$

$C_{11} = P+S-T+V = 20+0-6-5 = 9$

$C_{12} = R+T = 2+6 = 8$

$C_{21} = Q+S = 12+0 = 12$

$C_{22} = P+R-Q+U = 20+2-12+10 = 10$

$$\begin{bmatrix} 9 & 8 \\ 12 & 10 \end{bmatrix}$$

$$\begin{bmatrix} 4 & 3 \\ 2 & 4 \end{bmatrix} * \begin{bmatrix} 2 & 1 \\ 1 & 4 \end{bmatrix}$$

$P = (4+4)(2+4) = 48$

$Q = (2+4)2 = 12$

$R = 4(1-4) = -12$

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

$T = (4+3)4 = 28$

$U = (2-4)(2+1) = -6$

$V = (3-4)(1+4) = -5$

$C_{11} = P+S-T+V = 48-4-28-5 = 11$

$C_{12} = R+T = -12+20 = 16$

$C_{21} = Q+S = 12-4 = 8$

$C_{22} = P+R-Q+U = 48-12-12-6 = 18$

$$\begin{bmatrix} 11 & 16 \\ 8 & 18 \end{bmatrix}$$

$$\begin{bmatrix} 4 & 1 \\ 1 & 2 \end{bmatrix} * \begin{bmatrix} 3 & 2 \\ 4 & 3 \end{bmatrix}$$

$P = (4+2)(3+3) = 36$

$Q = (1+2)3 = 9$

$R = 4(2-3) = -4$

$S = 2(4-3) = 2$

$T = (4+1)3 = 15$

$U = (1-4)(3+2) = -15$

$V = (1-2)(4+3) = -7$

$C_{11} = P + S - T + V = 36 + 2 - 15 - 7 = 16.$

$C_{12} = R + T = -4 + 15 = 11$

$C_{21} = Q + S = 9 + 2 = 11$

$C_{22} = P + R - Q + U = 36 - 4 - 9 - 15 = 18.$

$$\begin{bmatrix} 16 & 11 \\ 11 & 18 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 3 \\ 3 & 4 \end{bmatrix} * \begin{bmatrix} 4 & 1 \\ 2 & 1 \end{bmatrix}$$

$P = (2+4)(4+1) = 30.$

$Q = (3+4)4 = 28.$

$R = 2(4-1) = 6.$

$S = 4(2-4) = -8.$

$T = (2+3)1 = 5.$

$U = (3-2)(4+1) = 5.$

$V = (3-4)(2+1) = -3.$

$$\begin{bmatrix} 14 & 11 \\ 20 & 13 \end{bmatrix}.$$

$C_{11} = P + S - T + V = 30 - 8 - 5 - 3 = 14.$

$C_{12} = R + T = 6 + 5 = 11$

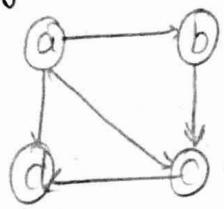
$C_{21} = Q + S = 28 - 8 = 20.$

$C_{22} = P + R - Q + U = 30 + 6 - 28 + 5 = 13.$

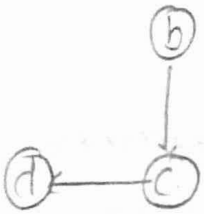
$$\therefore = \begin{bmatrix} 9 & 7 & 11 & 16 \\ 12 & 9 & 8 & 18 \\ 16 & 11 & 14 & 11 \\ 11 & 8 & 20 & 13 \end{bmatrix}$$

6.06.24
Thursday

Topological Order



remove a.



remove b.

remove c.

remove d

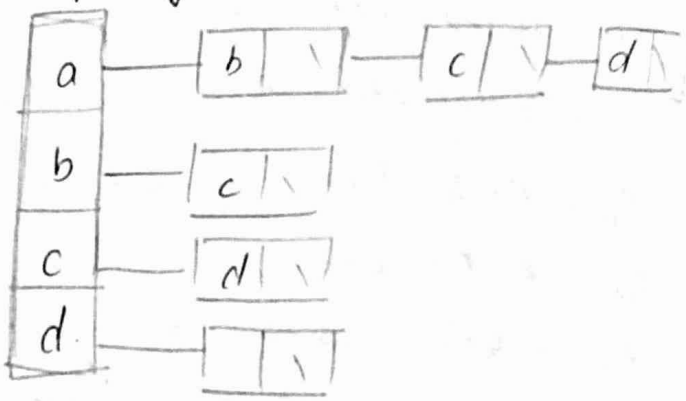
Directed graph

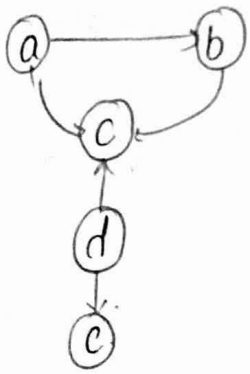
directed graph or digraph is a graph with direction specified for all its edges.

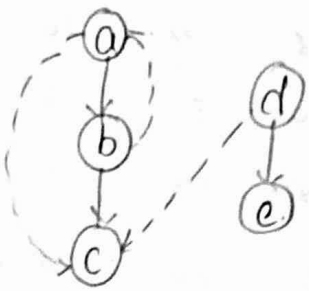
Adjacency matrix.

	a	b	c	d
a	0	1	1	1
b	0	0	1	0
c	0	0	0	1
d	0	0	0	0

Adjacency list.







* Four Types of edges possible in DFS

1) Tree edge (ab) (bc) (de)

2) Back edge (ba)

3) Forward edge (ac)

4) Cross edge (dc)

Implement Topological sort using source removal method find the time required to sort the elements.

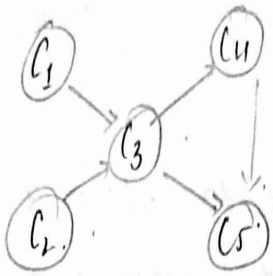
```
import java.util.*;
public class TopologicalSort {
    public static List<Integer> topologicalSort(List<List<Integer>> adj, int V) {
        int[] indegree = new int[V];
        for (List<Integer> list : adj) {
            for (int vertex : list) {
                indegree[vertex]++;
            }
        }
        // Queue to store vertices with indegree 0
        Queue<Integer> q = new LinkedList<>();
        for (int i=0; i<V; i++) {
            if (indegree[i]==0) {
                q.add(i);
            }
        }
        List<Integer> result = new ArrayList<>();
        while (!q.isEmpty()) {
            int node = q.poll();
            result.add(node);
            // decrease indegree of adjacent vertices as the current node is
               in topological order
            for (int adjacent : adj.get(node)) {
                indegree[adjacent]--;
            // if indegree becomes 0, push it to the queue
                if (indegree[adjacent]==0)
                    q.add(adjacent);
            }
        }
```

```
        if (result.size() != V) {
            System.out.println("Graph contains cycle!");
            return new ArrayList<>();
        }
        return result;
    }
    public static void main(String[] args) {
        int n = 3; //Number of nodes
        List<List<Integer>> edges = Arrays.asList(Arrays.asList(0,1),
                                    Arrays.asList(1,2), Arrays.asList(0,2));

        List<List<Integer>> adj = new ArrayList<>();
        for (int i=0; i<n; i++) {
            adj.add(new ArrayList<>());
        }
        // constructing adjacency list
        for (List<Integer> edge : edges) {
            adj.get(edge.get(0)).add(edge.get(1));
        }
        // performing topological sort
        System.out.print("Topological sorting of the graph:");
        List<Integer> result = topologicalSort(adj, n);

        // Display the result
        for(int vertex: result) {
            System.out.print(vertex + " ");
        }
    }
}
```

OUTPUT



Topological sorting of graph : 0 1 2 3 4.

Topological sorting of graph : contains cycle.