Syllabus of ADA

Module 1:- Introductory Concepts

- The notation of algorithm
- fundamentals of algorithmic problem Solving
- analyzing algorithms

→ Review of fundamentals data structure
- Array, Stacks
- Queue
- Linked list

2Q

Fundamentals of analysis of algorithms efficiency
- Asymptotic notation and standard efficiency classes
- mathematical analyses of recursive and non-recursive algorithms.

Divide and Conquer
- General method
- Merge Sort
- Quick Sort
- selection Sort

Sorting in Linear time
- Couting sort
- Radix sort and
- Bucket Sort

- Search
- Linear Search
- Binary Search

Module 2: Graps

1Q - Graph
1Q - Tree

→ Review of Graphs
→ Representation of Graphs
- BFS✓
- DFS✓

→ Topological sort
→ strongly connected components.

2Q

- Trees
- Review of Trees
- Minimum spanning tree
- kruskal and Prim's algorithm
- Single source shortest paths
- Bellman - Ford algo
- Single - Source shortest path in DAC (Direct Acyclic graphs)
- Dijkstra's algo

.All pair shortest path
- → Shortest paths and matrix multiplication
- → Floyd-Warshall algo
- → Johnson's algo

Module 3 : Dynamic Programming

- Introduction
- • Elements of dynamic programming
- Matrix chain multiplication — 10
- Longest common subsequence
- Optimal binary search tree
- Knapsack Problem
- Travelling Sales person problem
- Greedy Method
  - • An activity selection Problem
  - • Elements of Greedy Programming
  - • Huffman Codes
  - • A task scheduling Problem

→ Backtracking and Branch and Bound — 20
- • The 8 Queens Problem — 10
- • Graph coloring
- • Hamiltonian cycles
- • Least cost search (LC)
- • The 15 puzzle.
- Bounding
  - • FIFO branch & bound
  - • LC branch & bound

Module 4 → Maximum flow

- Flow networks
- The Ford-Fulkerson method
- → Maximum bipartite matching — 20

<u>Bubble Sort</u> :→ The basic idea of bubble sort is to compare two adjoining values and exchange them if they are not in proper order.

Intuition of bubble sort

<u>Algo</u> :- 1) A(n) ← Input
2) Compare (A(x), A(x+1))
if (A(x+1) < A(x))
Swap (A(x+1), A(x))
Repeat this (n-1) times.

3) Repeat step 2 (n-1) times.

<u>Algorithm</u>

```
1. For I = L to U                        for i = A.length
2.    { for J = L to [(U-1) -1]          for j = 1 to i-1
                                          if (A
3.    { if AR[J] > AR[J+1] then
      {
4.    temp = AR[J]
5.      AR[J] = AR[J+1]
6.      AR[J+1] = temp
7.    } } }
7.    End
```

```
# include <iostream.h>
Void main ()
{
int a [50], i, n, j, tmp, ctr;
Cout << " How many elements do u want to
         create array with (max.50)";
Cin >> n;
Cout << "Enter Array elements";
for (i= 0; i <n; i++)
```

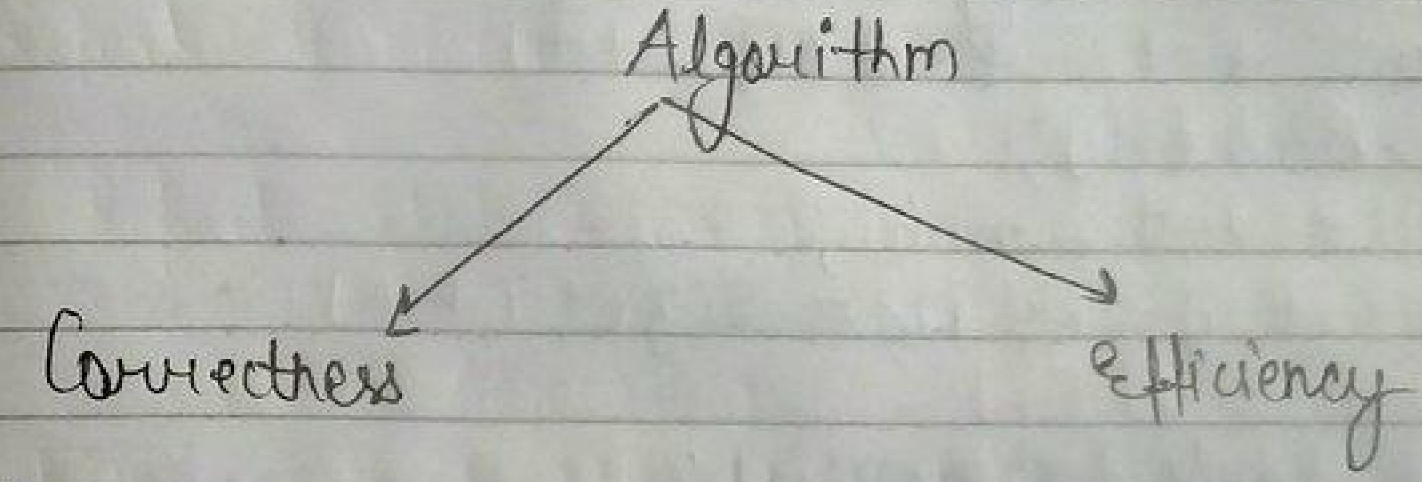
```
Cin >> n;
Cout << "Enter Array elements";
for ( i=0; i<n; i++)
  Cin >> a[i];
int( i=0; i<n; i++)
{ for( j=0; j < (n-1)--i; j++)
 { if (a[j] > a[j+1])
   {
     temp= a[j];
     a[j] = a[j+1];
      a[j+i] = temp;
   }
  }
 Cout << "Array after iteration"
   << ++ctr << " is";
 for (int i=0; i<n; i++)
   Cout << a [i] << " ";
    Cout << endl;
  }
 }
```

INSERTION SORT

1. For j ← 2 do length [A]
2. do key ← A[j]
3. Insert A [j] into the sorted array. Sequence A [1 --- j-1].
4. i ← j-1
5. While i>0 and A[i] > key
6. do A [i+1] ← A[i]
7. i= i-1
8. A[i +1] ← key.

Algorithm :→ An algorithm is any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output.

An algorithm is thus a sequence of computational steps that transform the input into the output.



Practical applications of Algorithm :-

1) Human Genome project
2) Internet
3) E-commerce.
4) Manufacturing & other commercial setting
5) Google
6

An algorithm is said to be correct if, for every input instance, it halts with the correct output.

Measure of efficiency is how long an algorithm takes to produce its result.

Insertion Sort:

```
{
int ar[50], n, index;
Cout <<" How many elements do you want to
     create array with ?";
Cin >> N;
Cout <<" Enter array elements";
for (int i=1; i<=N; i++)
  Cin >> ar[i];
ar [0] = int - min;
for (int i= 1; i<= size; i++)
  {
  int temp = U[i];
     j = i-1;
  while ( tmp < ar [j])
   {
   ar [j+1] = ar [j];
     j --;
   }
  ar [j+1] = temp;
Cout <<" Array after Pass -" << i <<" - is!";
for (int K=1; K <=size; K++)
    cout << ar [K] <<" ";
}
```

Selection sort:

```
{
int ar[50], item, n, index;
   Cout <<
```

```
for (int i=0; i<n; i++)
  Cin >> ar[i];
  for( int i=0; i< size; i++)
   {
   Small = ar[i];
   for ( int j= i+1; j< size; j ++)
    { if (ar[j] < small)
    { small = ar[j];
      pos = j;
      }
    }
   tmp = ar[i];
   ar[i] = ar[pos];
   ar[pos] = temp;
  Cout <<" Array after pass! " << i+1 <<" is";
  for (j=0; j<= size; j++)
   cout << ar[j] <<" ";
  }
  }
```

1. Bubble Sort Pseudocode
2. Insertion Sort C Program

INSERTION SORT (A) Pseudocode:-

1) for j = 2 to A. length
2) key = A[j]
3) // Insert A[j] into the Sorted sequence. A[1 ------ j-1]
4) i = j-1
5) While i > 0 and A[i] > Key
6) A[i+1] = A[i]
7) i = i-1
8) A[i+1] = Key

Bubble Sort (A) Pseudocode

```
1) For i=1 to A.length
2)   For j=1 to A.length - 1
3)     if A[j] > A[j+1]
4)       temp = A[j]
5)       A[j] = A[j+1]
6)       A[j+1] = temp
```

Bubble Sort (A) Optimised.

```
1) for i= A.length
2) for j= 1 to i-1
3)   if (A[j+1] < A(j))
4)     temp = A[j]
5)     A[j+1] = A[j]
6)     A[j] = temp
7)   i=i-1
```

Pseudocode conventions:-

1) Indentation indicates block structure.
2) Looping constructs while, for and repeat-until and the if-else conditional constructs have interpretations similar to those in C, C++, Java, C#.
3) "//" (double slash) indicates that the remainder of the line is a comment.
4.) A multiple assignment of the form i=j=e is possible. It assignes both variables i, j with value of e.
5) Variables such as i, j, key are local to the procedure.
6) We access the array elements by specifying its name followed by the index in square brackets and here index will be starting from 1.
7) "...." are used to indicate continuation
8) We typically organize compound data into objects, which are composed of attributes (A.length). The convention for accessing such attribute of an object is object name followed by (.) with attribute name.
9) We pass the parameters to a procedure by value, The called procedure receive its own copy of parameters, if it assigns value to the parameter the change is not seen by the calling group.
10) A return statement immediately transfer control back to the point of call in the calling procedure. Most written statements also take a value to pass back to the caller. Our Pseudocode differs from many programming languages in that we allow multiple values to be returned in a single return statement.
11) The boolean operators "and" and "or" are short circuited i.e. when we evaluate the expression "x" and "y" we first evaluate x. If x evaluates to FALSE, then the entire expression cannot evaluate to TRUE, and so do not evaluate y.
12) The keyword "error" indicates that an error occured because condition were wrong for the procedure to have been called. The calling procedure is responsible for handling the error, and so we do not specify what action to take.

INSERTION SORT – DESCENDING (A).

Ques Using insertion sort, sort the given list in descending order from high to low.

1) for $j=2$ to A. length
2) $Key = A[j]$.
3) // Insert $A[j]$ into the sorted sequence $A[1 \text{------} j-1]$.
4) $i = j-1$.
5) while $j > 0$ and $A[i] < Key$
6) $A[i+1] = A[i]$
7) $i = i-1$.
8) $A[i+1] = Key$

Q Write a Pseudocode for (finding or searching a number in an sequence of no. by comparing item to be searched with all elements sequentially & if number does not exist it should return NULL

SEARCHING – SORT (A)

1) for $j = 1$ to A. length
2) $Key = A[j]$
3) if $Key == Item$.
4) Print Key
5) else
6) Print Not found.

☆ Merge Sort algorithm

Merge (A, p, q, r)

1) $n_1 = q-p+1$
2) $n_2 = r-q$
3) Let $L = [1 \text{----} n_1+1]$ and $R = [1 \text{----} n_2+1]$. be new arrays.
4) for $i = 1$ to n_1
5) $L[i] = A[p+i-1]$
6) for $j = 1$ to n_2.
7) $R[j] = A[q+j]$
8) $L[n_1+1] = \infty$.
9) $R[n_2+1] = \infty$.
10) $i = 1$
11) $j = 1$
12) for $K = p$ to r.
13) if $L[i] \leq R[j]$
14) $A[K] = L[i]$
15) $i = i+1$
16) else $A[K] = R[j]$
17) $j = j+1$.

Implementation for algo :-

Let $p=1$, $q=5$, $r=8$. (total no. of elements)

1	2	4	6	9	~~x~~	3	7	8

1) $n_1 = q-p+1 \Rightarrow 5-1+1 = 4+1 = 5$.
2) $n_2 = r-q = 8-5 = 3$.
3) let $L = [1 \text{----} 6]$, $R = [1 \text{----} 4]$
4) for $i = 1$ to 5
5) $L[1] = A[p+1-1] = A[1+1-1] = A[1] = 1$
$L[2] = A[1+2-1] = A[1+1] = A[2] = 2$
$L[3] = A[3] = 4$
$L[4] = A[4] = 6$, $L[5] = A[5] = 9$

$L \rightarrow$

1	2	4	6	9	∞

6) for $j = 1$ to $n_2 = 1$ to 3, $q = 5$.

7) $R[j] = A[q+j]$

$R[1] = A[5+1]$

$R[1] = A[6]$

$R[1] = 3$.

$R[2] = A[5+2]$

$R[2] = A[7]$

$R[2] = 7$.

$R[3] = A[5+3]$

$R[3] = A[8]$

$R[3] = 8$.

R → | 3 | 7 | 8 | ∞ |

8) $L[n_1+1] = L[5+1] = L[6] = \infty$.

9) $R[n_2+1] = R[3+1] = R[4] = \infty$

10) $i = 1$

11) $j = 1$

12) for $k = p$ to $r = 1$ to 8.

$k = 1$,

13) For $A[k] \le R[j]$, 14) $A[k] = L[i]$

$A[1] \le R[1]$. $A[1] = L[1]$

$1 \le 3$. True. $A[1] = 1$.

$k = 2$, $i = 2$

for $A[2] \le R[1]$, $A[k] = L[i]$

$2 \le 3$. True $A[2] = L[2]$.

$A[2] = 2$.

$k = 3$ $A[3] \le R[1]$

$4 \le 3$ False.

$A[3] = R[1]$

$A[3] = 3$.

15) $i = i+1$

$i = 1+1 = 2$

L → | 1 | 2 | 4 | 6 | 9 | ∞ |, R → | 3 | 7 | 8 | ∞ |

~~$k=4$. $L[2] \le R[2]$~~ $k = 1$ to 8

13) If $L[i] \le R[j]$.

14) $A[k] = L[i]$

15) $i = i+1$

else

16) $A[k] = R[j]$

17) $j = j+1$.

$k = 1$ to 8, $i = 1$, $j = 1$

$k=1$, $i=1$: $L[i] \le R[j]$

$L[1] \le R[1]$

$1 \le 3$. True.

$A[k] = L[i]$

$A[1] = L[1]$.

$A[1] = 1$.

$k=2$, $i=2$: $L[2] \le R[1]$

$2 \le 3$. True

$A[k] = L[i]$.

$A[2] = L[2]$

$A[2] = 2$.

$k=3$, $i=3$: $L[3] \le R[1]$

$4 \le 3$. False.

$A[k] = R[j]$

$A[3] = R[1]$

$A[3] = 3$.

$k=4$, $j=2$, $i=3$: $L[3] \le R[2]$

$4 \le 7$ True

$A[k] = L[i]$

$A[4] = L[3]$

$A[4] = 4$.

$k=5$, $i=4$, $j=2$: $L[4] \le R[2]$.

$6 \le 7$ True

$A[5] = L[4]$

$A[5] = 6$.

$k=6$, $i=5$, $j=2$: $L[5] \le R[2]$

$9 \le 7$ False

$A[k] = R[j]$

$A[6] = R[2]$

$A[6] = 7$.

$k=7$, $i=5$, $j=3$: $L[5] \le R[3]$

$9 \le 8$. False.

$A[k] = R[j]$

$A[7] = R[3]$.

$A[7] = 8$.

$k=8$, $i=5$, $j=4$: $L[5] \le R[4]$

$9 \le \infty$. ~~False~~ True

$A[k] = L[i]$

$A[8] = L[5]$

$A[8] = 9$.

A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
1	2	3	4	6	7	8	9

It will merge the array without depending on q (Partition). Partition can be $\frac{1}{3}$rd.

4th step:- Termination Condition

Factorial :- if (n = = 0)
return 1
else.

☆ Merge Sort (A, p, r) $\boxed{r = A.length}$

1) if p < r
2) ~~q = [L(p+r)/2]~~ , $q = \lfloor (p+r)/2 \rfloor$
3) Merge sort (A, p, q)
4) Merge-sort (A, q+1, r)
5) Merge (A, p, q, r)

mid pt = 4.

A [ ] = { 4, 8, 3, 2 }

p = 1, r = 4.

* MS (A, 1, 4) , p = 1, r = 4
1) if p < r
   1 < 4 True.
   ~~MS (A, q+1, r)~~
   ~~MS (A.~~
2) q = [~~L~~(p+r)/2]
   q = ~~L~~(1+4)/2
   q = ~~L~~(5/2)
   q = 2.
3) MS (A, p, q)
   MS (A, 1, 2).
4) MS (A, p, q, r)
   MS (A, 1, 2, 4)

3) MS (A, 1, 2) (p = 1, r = 2)
a) p < r
   1 < 2 True.
b) q = (1+2)/2
   q = 3/2 = 1

∴ MS (A, 1, 1).
a) p < r
   1 < 1 (fail).

MS (A, 2, 2)
(fail).

MS (A, 1, 1, 2)

MS (A, p, q, r)

4	8

Merge (A, 1, 1, 2).

4	8	3	2

1.) $n_1 = 1 - 1 + 1 = 1$
2) $n_2 = 2 - 1 = 1$
3) L = [1 - - - 2], R = [1 - - - 2]
4) for (i = 1 to 1) L[1] = A[1 + 1 - 1].
   L[1] = A[1]
   L[1] = 4.

L = $\boxed{4}$

5) For (j = 1 to 1)
   R[1] = A[2]
   R = $\boxed{8}$.

9) i = 1 , j = 1
   for (k = 1 to 2)

K = 1, if L[1] ≤ R[1]
   4 ≤ 8
   A[1] = L[1] = 4 , i = 2

4	8

K = 2 if L[2] ≤ R[1]
   ∞ ≤ 8 False
   A[2] = R[1].
   A[2] = 8

6) MS (A, 3, 4)
1) 3 < 4 True
2) q = (3+4)/2 = 7/2 = 3
   MS (A, 3, 3) → fail
   MS (A, 4, 4) → fail.
   Merge (A, 3, 4, 4) .

2	3

Merge (A, 3, 3, 4) (p = 3, q = 3, r = 4)
1) $n_1 = 3 - 3 + 1 = 1$
2) $n_2 = 4 - 3 = 1$
3) L = [1 - - - 2], R = [1 - - - 2]
4) i = 1
   L[1] = A[3]

L → | 3 | ∞ |

$j=1$
$R[1] = A[4]$ $\quad R \rightarrow$ | 4 | ∞ |

$L[2] = \infty$
$R[2] = \infty$

10) $i=1, j=1$
$K = 3$ to 4 $\quad$ | 2 | 3 |

if $L[1] \leq R[1]$
$3 \leq 2$ false
$A[3] = R[1] = 2$, $\quad \underline{j=2}$

$\underline{K=4}$ if $L[1] \leq R[2]$
$3 \leq \infty$ True
$A[4] = L[1] = 3$
$MS(A, 1, 2, 4)$

1. $n_1 = 2-1+1 = 2$
$n_2 = 4-2 = 2$
$L = [1 - - - 3]$, $R = [1 - - - 3]$
for $i = 1$ to 2
$\underline{i=1}$ $L[1] = A[1]$ $\quad L \rightarrow$ | 4 | 8 | ∞ |
$\underline{i=2}$ $L[2] = A[2]$
$\underline{j=1}$ $R[1] = A[3]$
$\underline{j=2}$ $R[2] = A[4]$
$L[3] = \infty$
$R[3] = \infty$
$i=1, j=1$

$\underline{K=1}$ $L[1] \leq R[1]$
$4 \leq 2$ false
$A[1] = R[1]$ $\quad$ | 2 | 3 | 4 | 8 |
$j = 2$

$\underline{K=2}$ $L[1] \leq R[2]$
$4 \leq 3$ false
$A[2] = R[2] = 3$
$j = 3$

$\underline{K=3}$ $L[1] \leq R[3]$
$4 \leq \infty$ True
$A[3] = L[1]$, $i=2$

$\underline{K=4}$ $L[2] \leq R[3]$
$8 \leq \infty$ True
$A[4] = L[2] = 8$ $\quad$ | 2 | 3 | 4 | 8 |.

---

* Insertion Sort Algorithm implementation

2	6	8	4	3	1

(Descending order).

1) For $j = 2$ to $A.length$
2) $Key = A[j]$
4) $i = j-1$
5) while $i > 0$ and $A[i] < Key$
6) $A[i+1] = A[i]$
7) $i = i-1$
8) $A[i+1] = Key$.

$A.length = 6$.

1) $j = 2$ to 6.
2) $Key = A[j] = A[2] = 6$.
4) $i = j-1$
$i = 2-1 = 1$, $i=1$
5) while $i > 0$ and $A[i] < Key$
$1 > 0$ and $A[1] < 6$
$1 > 0$ and $2 < 6$. $\underline{\text{True}}$
6) $A[i+1] = A[i]$
$A[2] = A[1]$
$\boxed{A[2] = 2}$.
7) $i = i-1$
$i = 1-1 = 0$.
8) $A[0+1] = 6$
$\boxed{A[1] = 6}$.

~~2~~	~~2~~	~~8~~	~~4~~	~~3~~	~~1~~

6	2	8	4	3	1

while $i > 0$ and $A[i]$ $\cancel{>}$ < Key
$0 > 0$ false

2nd pass

1) $j = 3$, A.length $= 6$
2) Key $= A[3] = 8$.
Key $= 8$
4) $i = j - 1 = 3 - 1 = 2$
$i = 2$
5) while $i > 0$ and $A[i] <$ Key
$2 > 0$ and $A[2] < 8$
$2 > 0$ and $2 < 8$ True
6) $A[i+1] = A[i]$
$A[3] = A[2]$
$\boxed{A[3] = 2}$
7) $i = i - 1 = 2 - 1 = 1$
$i = 1$.

6	8	2	4	3	1

8) $A[i+1] =$ key
$\boxed{A[2] = 8}$

5) while $i > 0$ and $A[i] <$ key
$1 > 0$ and $A[1] < 8$
$1 > 0$ and $6 < 8$ True.
6) $A[i+1] = A[i]$
$A[2] = A[1]$
$A[2] = 6$.
7) $i = i - 1$
$i = 1 - 1$
$i = 0$

8	6	2	4	3	1

8) $A[1] =$ Key
$A[1] = 8$.

5) while $i > 0$ and $A[i] <$ Key
$0 > 0$ false.

3rd pass

1) $j = 4$, A.length $= 6$
2) Key $= A[4] = 4$.
Key $= 4$.
4) $i = j - 1 = 4 - 1$
$i = 3$.
5) while $i > 0$ and $A[i] <$ Key
$3 > 0$ and $2 < 4$. True.
6) $A[i+1] = A[i]$
$A[4] = A[3]$
$A[4] = 2$.

8	6	4	2	3	1

7] $i = i - 1$
$i = \cancel{2}\ 3 - 1$
$i = 2$.
8] $A[i+1] =$ key
$A[3] = 4$.
5) while $i > 0$ and $A[i] <$ Key
$2 > 0$ and $6 < 4$ false.

4th pass

1) $j = 5$, A.length $= 6$
2) key $= A[5] = 3$.
key $= 3$
4) $i = j - 1 = 5 - 1 = 4$
$i = 4$.
5) while $i > 0$ and $A[i] <$ Key
$4 > 0$ and $A[4] < 3$.
$4 > 0$ and $2 < 3$ True.
6) $A[5] = A[4]$
$A[5] = 2$.

8	6	4	3	2	1

7) $i = 3$.
8) $A[4] = 3$.
5) while $i > 0$ and $A[i] <$ Key
$3 > 0$ and $A[3] < 3$
$3 > 0$ and $4 < 3$ false

5th pass 1) $j = 6$, $A.length = 6$

2) $Key = A[j] = A[6]$

$Key = 1$.

3) $i = j-1 = 6-1$

$i = 5$.

4) while $i > 0$ and $A[i] < Key$

$5 > 0$ and $2 < 1$. false

Ans. | 8 | 6 | 4 | 3 | 2 | 1 |

Insertion sort.

1) For $j = 2$ to A.length
2) key = A[j].
3) //Ins ---
4) $i = j-1$
5) while $i > 0$ and $A[i] < key$
6) $A[i+1] = A[i]$
7) $i = i-1$
8) $A[i+1] = key$

S.No.	Cost	Time	Total cost
1	c_1	n	$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} t_j + c_5 \sum_{j=2}^{n}(t_j - 1) + c_6 \sum_{j=2}^{n}(t_j-1) + c_7(n-1)$ --- eq(i)
2	c_2	$n-1$	
3		$n-1$	
4	$\cancel{c_4}$ c_3	$n-1$	
5	$\cancel{c_5}$ c_4	$\sum_{j=2}^{n} t_j$	
6	$\cancel{c_6}$ c_5	$\sum_{j=2}^{n} (t_j - 1)$	
7	$\cancel{c_7}$ c_6	$\sum_{j=2}^{n} (t_j - 1)$	
8	$\cancel{c_8}$ c_7	$(n-1)$	

$$\sum_{j=2}^{n} (j-n)$$

$$T(n) = \sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1 \quad --- (a)$$

$$\sum_{j=2}^{n} (j-1) = \frac{n(n-1)}{2} \quad --- (b)$$

Substitute the above values in eq(i)

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4\left[\frac{n(n+1)}{2} - 1\right] + c_5\left(\frac{n(n-1)}{2}\right) + c_6\left(\frac{n(n-1)}{2}\right) + c_7(n-1)$$

$$\boxed{T(n) = An^2 + Bn + C}$$ → worst case running time

$$An^2 + Bn + C = c_1 n + c_2(n-1) + c_3(n-1) + c_4\left(\frac{n(n+1)}{2} - 1\right) + c_5\left(\frac{n(n-1)}{2}\right) + c_6\left(\frac{n(n-1)}{2}\right) + c_7(n-1)$$

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4(n-1) + c_7(n-1)$$

$$= (c_1 + c_2 + c_3 + c_4 + c_7) - (c_2 + c_3 + c_4 + c_7)$$

$$\left(\frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2}\right)n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4}{2} - \frac{c_5}{2} - \frac{c_6}{2} + c_7\right)n - (c_2 + c_3 + c_4 + c_7).$$

$$= \left(\frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2}\right)n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4}{2} - \frac{c_5}{2} - \frac{c_6}{2} + c_7\right)n - (c_2 + c_3 + c_4 + c_7).$$

BOT

$O \ \Omega \ \Theta$

Asymptotic notation

Insertion sort's worst case running time is $\Theta(n^2)$.

$T(n) = \Theta(n^2)$

Θ notation:- $T(n) = \Theta(n^2)$

$\Theta(g(n)) = \{f(n)$: there exist positive constants c_1, c_2 and n_0 s.t

$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0\}$.

O-notation:- The Θ-notation asymptotically bounds a function from above and below. When we have only an asymptotic upper bound, we use O-notation.

$O(g(n)) = \{f(n)$: there exist positive constants c and n_0 such that

$0 \leq f(n) \leq c g(n)$ for all $n \geq n_0\}$.

Ω-notation:- Ω-notation provides an asymptotic lower bound.

$\Omega(g(n)) = \{f(n)$: there exist positive constants c and n_0 such that

$0 \leq c g(n) \leq f(n)$ for all $n \geq n_0\}$.

$O(g(n)) = \{ f(n)$: there exist positive

$0 \leq f_n \leq g(n)$ for all $n \geq n_0$

Q1- What is Pseudo-Code?

Ans- A mixture of natural language and high-level programming concepts that describes the main ideas behind a generic implementation of a data structure or algorithm.

The O notation

Definition: The O (big-oh) is the formal method of expressing the upper bound of an algorithm's running time. It's a measure of the longest amount of time it could possibly take for the algorithm to complete.

1) n = W.rows

(2) $D^0 = W$

(3) for k = 1 to n

(4) let $D^{(k)} = (d_{ij})^k$ be a new n x n matrix

(5) for i = 1 to n

FLOYD-WARSHALL (W)

(1) n = W.rows

(2) ~~for k = 1 to n~~ $D^0 = W$

(3) for k = 1 to n

(4) let $D^k = (d_{ij})^k$ be a new n x n matrix

(5) for i = 1 to n

(6) for j = 1 to n