INDEX

NAME: Anushka Jain STD.: B.Tech CSE (AI-ML) SEC.: A ROLL NO.: RA2211026030006 SUB.: DSA

S. NO.	DATE	Topic	Page No.	Teacher's Sign / Remarks
1	17/7/23	Basics about DSA	1	
2	18/7/23	Types, Applications, Operations on DS.; Algorithms (Life cycle) Time & Space Complexity, ques.	2-13	
3	19/7/23	Time complexity ques	14-16	
4	21/7/23	Searching (Linear & Binary)	17-19	
5	24/7/23	Sorting (Insertion & Bubble)	20-26	
6	25/7/23	DSA LAB (Ques)	26-29	
7	28/7/23	Asymptotic Notations	30-33	
8	31/7/23	Theta Notation + Ques	34-38	
9	1/8/23	Ques + Primitive Data Types + Structures + Dynamic Mem Allocation	39-45	

Analytics Vidya → Job-A-Thon (Apply here!)
(Every year once)

Date: 17/7/23
Page No. 1

Data Structures & Algorithms

↓

how data is organised & stored

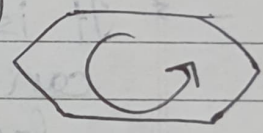
Data structures (D.S.)

Linear D.S.
- Arrays
- Linked list
- Stack
- Queue

Non-linear D.S.
~~cyclic~~
- Trees
- Graphs

↓

Always form cyclic structure



★ Operations on D.S.

1. Traversing (Accessing all the data inside D.S. atleast once)
   (Updation)
2. Insertion (start, end, specific position)
3. Deletion
4. Searching
5. Sorting

Stu. Name = 'Prakhar'
SEM = 3rd
D.O.B = 18/8/2003
Branch = CSE-AIML

Database → application of data structure (D.S)

→ A data structure is a particular way of organising data in the system so that we can use it efficiently.

→ It is a set of algorithms that we can use in any programming language to structure the data in the memory.

D.S.

- Linear
  - → Arrays
  - → linked list
  - → Queues
  - → Stacks
- Non-Linear
  - → Trees
  - → graphs

Applications of Data structures

1) Operating Systems
2) DBMS
3) Compiler design
4) AI (Knowledge-Base)
5) Analysis

Linear D.S.

① Arrays

→ It is a collection of elements. It is used in mathematical problems like matrix, algebra, etc.

→ Each element of an array is referenced by a subscripted variable or value or index enclosed in parenthesis.

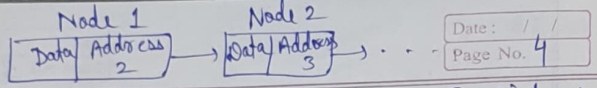
② Linked list

→ It is a collection of data elements.

→ It consists of 2 parts → information (data) & link (address).

Node

Data	Address



→ Info gives information & link is an address of next node.

→ Linked list can be implemented by using pointers

H.W. Applications of linked list

③ Stack

→ It is a list of elements.

→ In stack, an element may be inserted or deleted at one end which is known as top of the stack.

→ It performs 2 operations → push & pop.

(i) Push → Adding an element in stack

(ii) Pop → Removing an element from stack.

→ It is also called LIFO.

Implementation of stack

Push → overflow तो नहीं मान रही

Pop → underflow (khaali तो नहीं है)

④ Queue

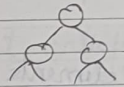
→ Queue is a linear list of elements.

→ In queue, elements are added at one end called Rear end. & are deleted from another end called front end.

→ It is also called FIFO

Non-linear D.S.

① Trees

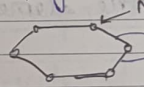


→ It is a Non-linear D.S.
↳ flexible, versatile & powerful

→ used to represent data items processing hierarchical relationship b/w the root & it's children

→ It is an ideal data structure for representing any hierarchical data.

② Graph



→ It is a non-linear D.S. which consists of a finite set of ordered pairs called edges.

→ graph is a set of elements connected by edges. Each element is called vertex or node

D.S. Operations

① **Traversing**

→ Accessing each element of D.S. atleast once.

② **Searching**

→ Search for any element in a D.S.

③ **Sorting**

→ Sort the elements of a D.S. either in an ascending or descending order.

④ **Insertion**

→ Insert any element in a D.S.

⑤ **Deletion**

→ To remove the element from the D.S.

⑦ **Updation**

→ By performing any modification in a D.S. either by →
- swapping the elements
- inserting or deleting the elements
- sorting elements
- replacing element with another element.

Algorithm

→ Algo. is a finite sequence of instructions each of which is very elementary that must be followed to solve a problem.

→ An algo. is a well-defined computational procedure that transforms inputs into outputs by achieving the desired I/P - O/P relationship.

→ All algo's must satisfy the following criteria -

(i) I/P
(ii) O/P
(iii) finiteness
(iv) effectiveness
(v) definiteness

Algorithm Development life cycle

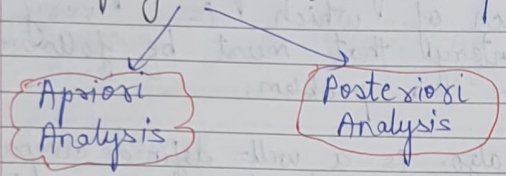
→ In the life cycle of an algo. for the development of algo. following phases are involved:

(1) Design Phase (Algo Design)

(2) Writing Phase

(3) Testing or Experiment Phase

(4) Analysing Phase (Time & Space complexity)



⊙ Apriori Analysis: It is system dependent
→ analysis of an algo. running on a specific system.

⊙ Posteriori Analysis: Actual amount of time & space taken by the algo's are recorded during the execution.

(★) Time and Space Complexity

(1) Space complexity :→
- amount of space needed for algo to solve the problem.
- An efficient algo. takes space as small as possible.
- Space needed by a program is the sum of the following components:→

✓(a) Instruction space
→ Space needed to store the executable version of a program & it is fixed.

✓(b) Data space
(i) → Space needed to store all elements – all constant, variables & the following components (ii) → space needed by constant & simple variable. The space is fixed.

~~Space~~

(iii) → Space needed by fixed size structured variables such as arrays & structure.

✓(c) Dynamically-allocated space from memory pool.
→ This space usually varies

✓(d) Inbuilt stack space
→ Space needed to store the info.
→ Space needed to resume the suspended function.

Questions

Q1 W.A.P to swap 2 elements without using 3rd variable.

Ans-1)

```
int a = 3;
int b = 4;
~~b = (a+b) - b.~~

b = (a+b) - (a=b);
cout << a << b;
```

M-2

```
a = a+b;
b = a-b;
a = a-b;
```

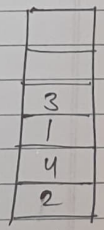
Q2 W.A.P. to access each element of an array {2, 9, 10, 3, 4, 6}.

Ans-2)

```
int arr[] = {2, 9, 10, 3, 4, 6};
for (int i=0; i< arr.length; i++)
{
    cout << arr[i] << endl;
}
```

Q3 Insert an element in stack

→ Insert = 8
→ Delete last element from stack.



~~int~~

```
import stack
stack<int> s = {2, 4, 1, 3};
s.push(8);
```



```
int s = [2, 4, 1, 3];
```

Ans-3)

```
import stack
stack<int> s = {2, 4, 1, 3};
int size = 7;
if (s.length < size) {
    s.push(8);
}
else {
    cout << "size overflow" << endl;
}
while (!s.empty())
{
    s.pop();
}
```

Time Complexity

Linear Search A = {3, 2, 0, 40, 6, 15, 1}

Key element = 3 → Best case ⇒ O(1)

Key element = 1 → Worst case ⇒ O(n)

Best case	Average case	Worst case
Θ	Ω	Big-oh (O)

→ The time complexity of an algo. is the ~~no. of computer~~ amount of time needed by computer to execute the complete code.

★ Rules for computing running time complexity

① **Sequence**

Add the time of the individual statements.

② **Loops**

Execution time of a loop is atmost the execution time of the statements of the body multiplied by the no. of iterations.

③ **Nested Loops**

Analyse them inside-out.

④ **Alternative Structures**

Time for testing the condition

+

Max. time taken by any of the alternative paths.

⑤ **Sub-programs**

Analyse them as seperate algo's & substitute the time wherever necessary.

Q1

```
int a=0, b=0;
for (int i=0; i<n; i++)   => O(n)
{
    a= a+ rand();
}
for (j=0; j<m; j++)   => O(m)
{
    b = b + rand();
}
```

$\Rightarrow O(n+m)$

Q2

```
int a=0.
for (i=0; i<n; i++)   --> O(n)
{ for (j=n; j>i; j--)   --> O(n)
    { a=a+i+j
    }
}
```

$\Rightarrow O(n * n) \Rightarrow O(n^2)$

Q3

```
int i,j,k = 0;
for(i=n/2; i<=n; i++)
{ for (j=2; j<=n; j = j*2)
    { k=k+n/2;
    }
}
```

$\Rightarrow O(n \log n) = O(n \log_2 n)$

Agr 3, 4, 5 etc. base हो तो उसे mention krna zaroori hai BUT base 2 को mention krna zaroori नहीं

Date: 19/7/23
Page No. 14

Questions on Time Complexity

1, 2, 4, 8, 16.

Q1
```
int k=1.                                  16
while (k<=n)                              2^2+1
{
  cout << k << endl;
  k = k*2;
}
```
$O(\log_2 n)$

Q2
```
for (int i=0; i<n; i++)
{
  for (int j=i; j>0; j--)
  {
    cout << i << j;
  }
}
```
$O(n^2)$

Q3
```
import random
def fun(N)
  counter = 0
  for i in range(N)
      count += random.randint(1,100)
      print(counter)
```
$O(N)$

Date: / /
Page No. 15

Q4
```
def function(N,M)
  counter = 0
  for i in range(N):
    for j in range(M):
      counter += 1
      print(counter)
```
$O(NM)$

Q5
```
def fun(n,m)
arr = [[0]*m for i in range(n)]
for i in range(n):
    for j in range(m):
        k = 1
        while k < n*m:
            k *= 2
```
$O(nm \log(nm))$

Q6
```
def recursion(N):
    if (N==0):
        return
    print(N)
    recursion(N-1)
```
$O(N)$

Q7
```
if (i > j)
{ (conditional statement)
}
```
$O(1)$

Q8 def fun(N):
counter = 1
for i in range(1, N+1): → N
for j in range(1, i+1):
counter += 1
print(counter)

$O(N^2)$

Q9 def fun(N)
for i in range(1, N+1):
j = N
while j > 0
j //= 2

$O(N \log_2 N)$

(* O, ω, Θ → Asymptotic Notations)

Searching

Linear Search

	0	1	2	3	4	5	6	7
A =	1	2	(3)	8	0	14	10	6

n = 8

i i i i i i i i

Key = 3 (returns index = 2)

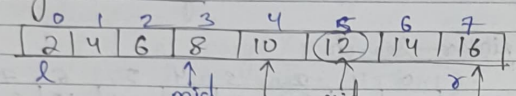
If Key = 11 (This key is NOT present)
∴ i = n

(Time complexity)
Best case → O(1)
Worst case → O(n)

code → for (i=0; i<n; i++)
{
if (a[i] == key)
{
printf ("Element found at index: %d", i);
break;
}
}
if (i == n)
{
printf ("Element not found");
}

Q (i) Explain linear search
(ii) Explain algo
(iii) Explain time complexity → all 3 with examples
Best, Worst → compulsory
(if time left → then avg too!)

Binary Search



★ Array should be "sorted" then ONLY we can apply Binary Search

$$mid = \frac{l+r}{2} = \frac{0+7}{2} = 3.5$$

case-1: if key == a[mid]

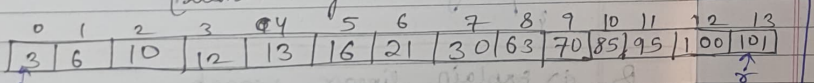
case-2: if key < a[mid]

case-3: if key > a[mid]

$$mid_1 = \frac{l_1+r_1}{2} = \frac{4+7}{2} = \frac{11}{2} = 5.5$$

(element is found here)

Q



(i) Key = 13

(ii) Key = 101

(i) Key = 13

l	r	mid	mid[data]
0	13	6	21 > key
0	5	2	10 < key
3	5	4	13 = key

(ii) Key = 101

l	r	mid	mid[data]
0	13	6	21 < key
7	13	10	85 < key
11	13	12	100 < key
13	13	13	101 = key

★ code →

```
Binary Search (a, n, data)
{
    l = 0 ; r = n-1 ;
    while (l <= r)
    {
        mid = (l+r)/2 ;
        if (data == a[mid])
            return mid ;
        else if (data < a[mid])
            r = mid - 1 ;
        else
            l = mid + 1 ;
    }
    return -1 ;
}
```

Time complexity → $O(\log_2 n)$

Sorting

Insertion Sort

A:

0	1	2	3	4	5
5	4	10	1	6	2

Sorted sublist | Unsorted sublist

Sorted sublist | Unsorted sublist

temp: 4

①

0	1	2	3	4	5
4	5	10	1	6	2

sorted | unsorted

temp: 10

②

4	5	10	1	6	2

sorted | Unsorted

temp: 1

4	5		10	6	2

	4	5	10	6	2

1

1	4	5	10	6	2

sorted | Unsorted

1	4	5	10		2

③

1	4	5		10	2

6

temp: 6

1	4	5	6	10	2

sorted | unsorted

1	4	5	6	10	

④

1	4	5	6		10

temp: 2

1	4	5		6	10

1	4		5	6	10

1		4	5	6	10

2

1	2	4	5	6	10

sorted ! (through insertion sort).

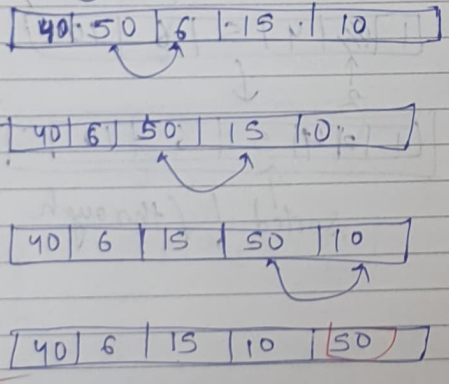
* Code of Insertion Sort

```
for (i=1; i<n; i++)
{
    temp = a[i];
    j = i-1;
    while (j>=0 && a[j] > temp)
    {
        a[j+1] = a[j];
        j--;
    }
    a[j+1] = temp;
}
```

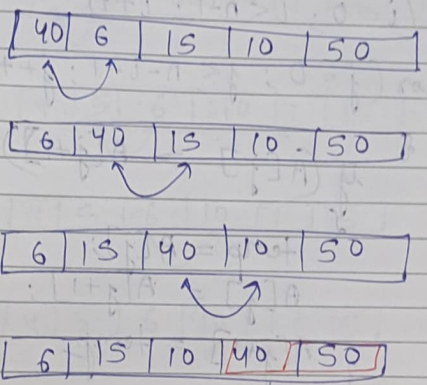
Bubble Sort

A: | 40 | 50 | 6 | 15 | 10 |

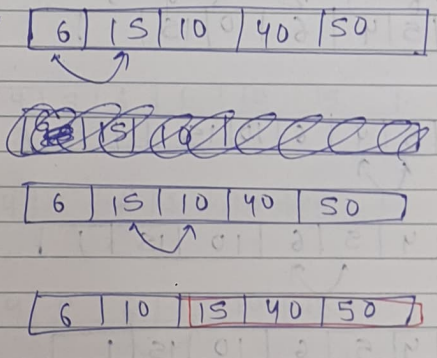
Pass-1



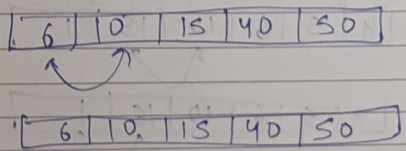
Pass-2



Pass-3



Pass-4



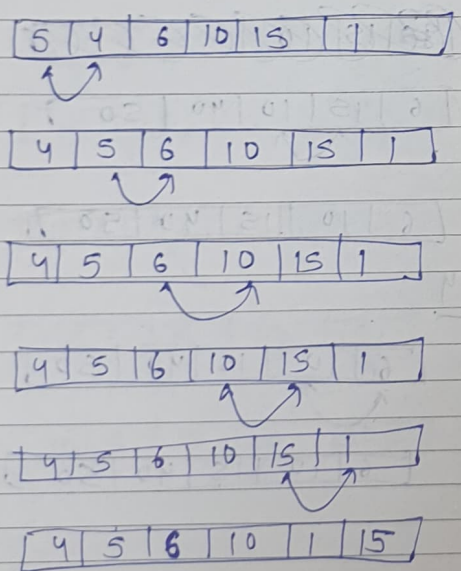
* Code of BUBBLE SORT

```
for (i = 0; i < n-1; i++)
{
  for (j = 0; j < n-i-1; j++)
  {
    if (A[j] > A[j+1])
    {
      temp = A[j];
      A[j] = A[j+1];
      A[j+1] = temp;
    }
  }
}
```

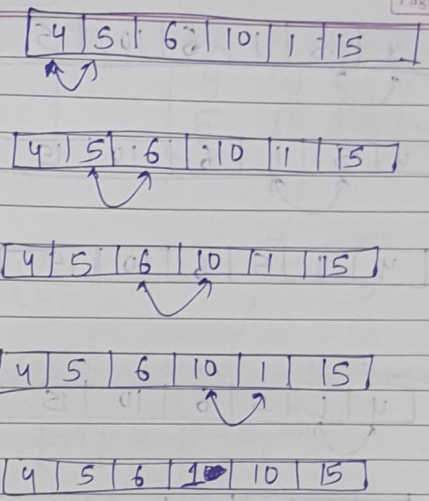
Q

0	1	2	3	4	5
5	4	6	10	15	1

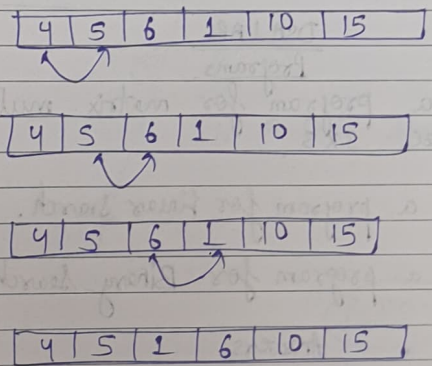
Pass-1



Pass-2



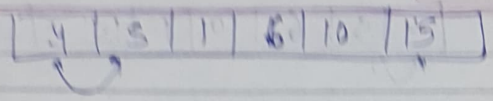
Pass-3



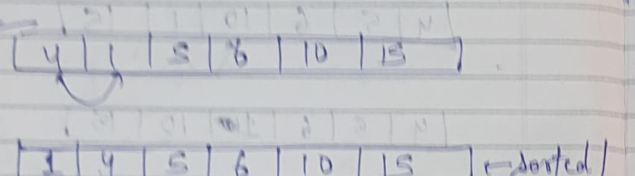
Pass

P.T.O.

Pass-4



Pass-5



29/9/23 DSA LAB Programs

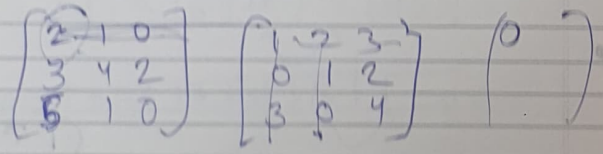
Q1 Write a program for matrix multiplication of size 3x3.

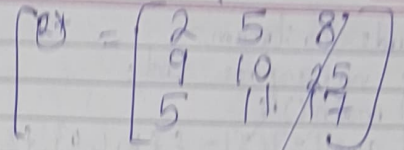
Q2 Write a program for linear search.

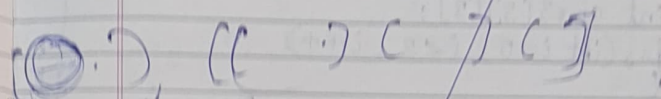
Q3 Write a program for Binary Search.

Answers

A-1








```
for (i=0; i< arr1.length; i++)
{
    for (j=0; j<3; j++)
    arr2[i][j] =
    arr1[i][j]
```

0 2 * 1

1 * 0.

P.T.O. →

```
#include<iostream>
using namespace std;
int main(){
  int arr1[3][3]={{2,1,0},{3,4,2},{5,1,0}};
  int arr2[3][3]={{1,0,3},{2,1,0},{3,2,4}};
  int arr3[3][3];
  for(int i=0; i<3; i++)
  {
    for(int x=0; x<3; x++)
    { int sum=0;
      for(int j=0; j<3; j++)
      {
        sum+=arr1[i][j]*arr2[x][j];
      }
      arr3[i][x]=sum;
    }
  }
for(int i=0; i<3; i++)
  {
    for(int j=0; j<3; j++)
    {
      cout<<arr3[i][j]<<" ";
    }
    cout<<endl;
  }
  return 0;
}
```


Imp

while initialising a 2-D (n-dimensional array), we can skip the length of first parameter BUT it is necessary to give the length of other parameters.

✓ eg) int arr1[ ][3]={{1,2,3},{4,5,6},{7,8,9}};

→ BUT while declaring a 2-D (n-dimensional array) it's necessary to give the first parameter also. we can NOT skip it. If we skip the first parameter while DECLARING, then it will give error.

✗ eg) int arr1[ ][3]; //error

it's imp. to give the first parameter also while DECLARING.

✓ eg) int arr1[3][3]; //correct

Asymptotic Notations

★ Types of fns

$O(1)$ → constant

$O(\log n)$ → Logarithm

$O(n)$ → Linear

$O(n^2)$ → Quadratic

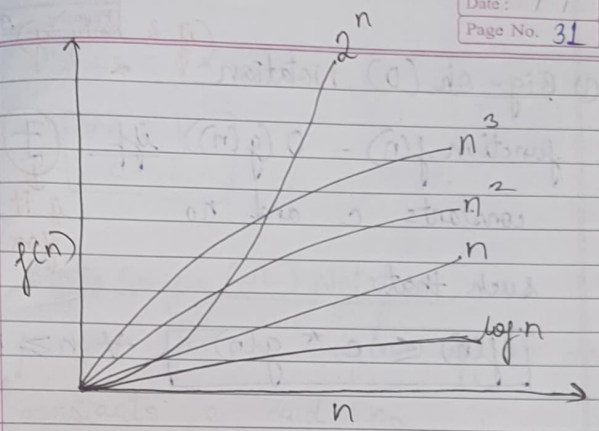
$O(n^3)$ → Cubic

$O(2^n)$ → Exponential

★ Order of function

$1 < \log n < \sqrt{n} < n < n\log n < n^2 < n^3 < \cdots \cdots < 2^n < 3^n \cdots < n^n$

	$\log n$	n	n^2	2^n	
$n=1$	0	1	1	2	small values of 'n'
$n=2$	1	2	4	4	
$n=4$	2	4	16	16	
$n=8$	3	8	64	256	



→ Asymptotic Notations are mathematical tools to represent time complexity of an algorithm.

→ They are used to express the rate of growth of an algo's running time in terms of I/P size.

★ Types of Asymptotic Notations

① Big Oh (O) Notation [Upper-bound]

② Omega (Ω) Notation [Lower-bound]

③ Theta (Θ) Notation [Average-bound]

① Big-oh (O) Notation

(if & only if)

function $f(n) = O(g(n))$ iff ∃ +ve constants c and n_0 such that (∃ = if it exists for all)

$$f(n) \leq c * g(n) \quad \forall\ n \geq n_0$$

Q. $f(n) = 2n+3$

⊙ $f(n) \leq c * g(n)$ (considering $g(n) = n$ (here))

$\Rightarrow 2n+3 \leq c * n$

$c = 10;\ n = 1$

$\Rightarrow 2+3 \leq 10$

$O(n)$

$n \geq 1$, $c = 10$

⊙ $2n+3 \leq c * (2n^2+3n^2)$ (Considering $g(n)$ (there) $= 2n^2+3n^2$)

→ Agar (Big-oh) satisfy krana hai to $g(n)$ aur c koi bhi aise lelo satisfy krao.

→ Aur agar check krna hoga for ∀ (big-oh) ∃ for which $f(n)$ aur $g(n)$ is done given honge.

$\Rightarrow 2n+3 \leq c * 5n^2$

$\forall\ n \geq 1$, $c = 1$

② Omega (Ω) Notation

function $f(n) = \Omega(g(n))$ iff ∃ +ve constants c and n_0 such that

$$f(n) \geq c * g(n) \quad \forall\ n \geq n_0$$

Q. $f(n) = 2n+3$

⊙ $2n+3 \geq c * n$

$5 \geq 1$

$\forall\ n \geq 1$ & $c = 1$

⊙ $2n+3 \geq c * \log_2 n$

$5 \geq 0$

$7 \geq 1$

$11 \geq 2$

$\forall\ n \geq 1$ & $c = 1$

Date 31/7/23
Page No. 34

③ Theta (Θ) Notation.

function $f(n) = \Theta(g(n))$ iff ∃ +ve constants c_1, c_2 & n_0 such that

$$c_1 * g(n) \leq f(n) \leq c_2 * g(n)$$

Q $f(n) = 2n+3$

Consider $g(n) = n$

$$1*(n) \leq 2n+3 \leq 5*(n)$$

For $c_1 = 1$
$c_2 = 5$
$n \geq 1$
$\Theta(n)$ is satisfied.

Q1 Redo $f(n) = 2n^2 + 3n + 4$

Find O, Θ, Ω

→ For such types of ques take 1 g(n) & satisfy O, Θ, Ω for that g(n)

① O (Big-oh)

Consider $g(n) = n^2$

$$2n^2+3n+4 \leq 10 * n^2$$

For $c = 10$
$n \geq 1$
$O(n^2)$ is satisfied.

Aur g(n) me jo f(n) ka highest term ho, we consider krte.

Date: / /
Page No. 35

② Ω (Omega)

Consider $g(n) = n$

$$2n^2+3n+4 \geq 1 * (n)$$

For $c = 1$
$n \geq 1$
$\Omega(n)$ is satisfied.

③ Θ (Theta)

Consider $g(n) = n^2$

$$1*(n^2) \leq 2n^2+3n+4 \leq 10*(n^2)$$

For $c_1 = 1$
$c_2 = 10$
$n \geq 1$
$\Theta(n^2)$ is satisfied.

Q2 Redo $f(n) = n^2 \log n + n$

① O (Big oh)

Consider $g(n) = n^3$

$$n^2 \log n + n \leq 10 * n^3$$

For $c = 10$
$n \geq 1$
$O(n^3)$ is satisfied.

② Ω (Omega)

Consider $g(n) = n$

$n^2 \log n + n \geq 1 * n$

For $c = 1$
$n \geq 1$
$\Omega(n)$ is satisfied

③ Θ (Theta)

Consider $g(n) = n^2$

$1 * (n^2) \leq n^2 \log n + n \leq 10 * (n^2)$

For $c_1 = 1$
$c_2 = 10$
$n \geq 1$
$\Theta(n)$ is satisfied.

Q1 $f(n) = 2n^2 + 3n + 4$. Satisfy O, Ω, Θ.

Ans Consider $g(n) = n^2$.

(i) $O(n^2)$

$f(n) \leq c * g(n)$

$2n^2 + 3n + 4 \leq 10 * (n^2)$

⊙ For ($c = 10$, $n \geq 1$)

$O(n^2)$ is satisfied.

(ii) $\Omega(n^2)$

$f(n) \geq c * g(n)$

$2n^2 + 3n + 4 \geq 1 * (n^2)$

⊙ For ($c = 1$, $n \geq 1$)

$\Omega(n^2)$ is satisfied.

(iii) $\Theta(n^2)$

$1 * (n^2) \leq 2n^2 + 3n + 4 \leq 10 * (n^2)$

For ($c_1 = 1$, $c_2 = 10$, $g(n) = n^2$)

$\Theta(n^2)$ is satisfied.

Q2 $f(n) = n^2\log n + n$

Ans Consider $g(n) = n^2\log n$

(i) $O(n^2\log n)$

$f(n) \leq c * g(n)$

$n^2\log n + n \leq 10 * n^2\log n$.

For $(c = 10, n \geq 2)$

$O(n^2\log n)$ is satisfied.

(ii) $\Omega(n^2\log n)$

$f(n) \geq c * g(n)$

$n^2\log n + n \geq 1 * n^2\log n$.

for $(c = 1, n \geq 1)$

$\Omega(n^2\log n)$ is satisfied.

(iii) $\Theta(n^2\log n)$

$c_1 * g(n) \leq f(n) \leq c_2 * g(n)$

$1 * (n^2\log n) \leq n^2\log n + n \leq 10 * (n^2\log n)$

for $c_1 = 10$, $c_2 = 10$, $n \geq 2$ $\Theta(n^2\log n)$ is satisfied.

Q1 $f(n) = 4n^2 + 5n + 3$

$g(n) = 5n^2$

Find all notations for above example.

Q2 Two Algo A_1 and A_2 run in same machine. Running time of A_1 is $100n^2$ & running time of A_2 is 2^n. For what value of n, A_1 runs faster than A_2?

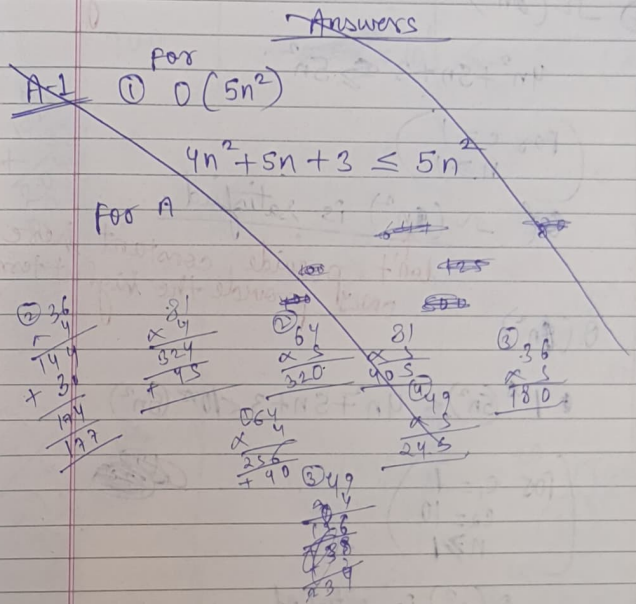
Q3 Find Ω for $f(n) = 3n^2 + 2n + 5$.

Answers

A1 (i) $O(5n^2)$ for

$4n^2 + 5n + 3 \leq 5n^2$

For A



Answers

Q1 ① $O(n^2)$

had $g(n)$ NOT been given, we will take it to be the highest term of the expression & c = (coeff of highest term + 1)

$4n^2 + 5n + 3 \leq 15n^2$

(For $c = 15$, $n \geq 1$)

$O(n^2)$ is satisfied → Now for that same value of 'c' satisfy all 3 notations for diff. values of 'n'.

② $\Omega(n^2)$

$4n^2 + 5n + 3 \geq 5n^2$

(For $c = 1$, $n \geq 1$)

$\Omega(n^2)$ is satisfied.

don't provide constant here ONLY provide the highest term.

③ $\Theta(n^2)$

$1 * (5n^2) \leq 4n^2 + 5n + 3 \leq 10 * (5n^2)$

(for $c_1 = 1$, $c_2 = 10$, $n \geq 1$)

$\Theta(n^2)$ is satisfied.

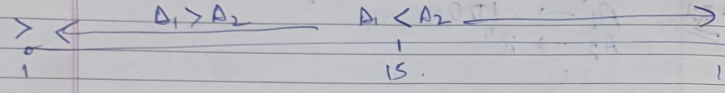
If time is there, draw graph too... otherwise show in tabular form!

Q2 $A_1 \rightarrow 100n$

$A_2 \rightarrow 2^n$

n	A_1	A_2
1	100	2
100	$\approx 10^6$	$< 2^{100}$ $\approx 10^{30}$
50	$100(2500) = 250000$	$< 2^{50} \approx 10^{15}$
25	62500	$< 2^{25}$ $\approx 2.43 \times 10^5$
20	$100(400) = 4 \times 10^4$	$< 2^{20} \approx 10^6$
15	$100(225) = 22500$	$< 2^{15} = 32768$
14	19600	$> 2^{14} = 16384$

$2^{10} \cdot 2^4 \approx 1024 \cdot 16$



A_1 takes more time than A_2

A_1 takes less time than A_2

A_2 is faster

A_1 is faster.

for $n \geq 15$, A1 is faster than A_2

Q3 $f(n) = 3n^2 + 2n + 5$

consider $g(n) = n^2$

for $\Omega(g(n))$

$f(n) \geq c * g(n)$

$3n^2 + 2n + 5 \geq 1 * n^2$

for $c = 1$, $n \geq 1$

$\Omega(n^2)$ is satisfied.

Primitive Data Types

- int
- char
- float
- double, etc.

Structures (User-defined data type)

[Syntax]:

```
struct structure_name {        (student_record)
    int roll_no;
    char name[10];
    float cgpa;
    int sem;
};
```

Calling / Invokation:-

* Create object

①
```
struct student record
{
    int roll_no;
    char name[10];
} s1[60];
```

②
```
void main()
{
    struct student record s2;        (for 1 student)
    struct student record s2[60];    (for 60 students)
```

* Self-referential structures

linked list, Trees, Graphs.

Code → struct node
{
int data;
struct node *ptr → next;
};

void main()
{
struct node N1;
printf("Enter value of node");
scanf("%d", N1.data);
}

Dynamic Memory Allocation (Run-time) [HEAP]
→ size increase

Static Memory Allocation (compile-time) [STACK]

Dynamic Memory Allocation

malloc()
calloc()
realloc()
free()

All these fns are declared in <stdlib.h>

① malloc (size of block) ← For 1 block

② calloc (n-blocks, size of each block) ← For more than one block
- It is a contigious size allocation.
- built-in fn in <stdlib.h>.
- used to dynamically allocate multiple blocks of memory & each block is of same size.

void * malloc (4);
void * calloc (5, 4);

③ realloc (ptr, size of block);
- No info. is lost
- First block (before) is deleted & new block with new size is created.

void * realloc (5, 7);

④ free (ptr);
- frees the memory space
void * free (ptr);