DAA Assignment - 1

Q1)

Ans) Asymptotic means towards infinity (i.e the input may be of infinite value). These notations are used to tell the time complexity of an algorithm when input is very large.

Following are Asymptotic notations:

i) Big - Oh (O) → $f(n) = O(g(n))$ if $f(n) \leq c \times g(n)$

This means that $g(n)$ is upper bound of $f(n)$.
A tight upper bound.

For Ex → / Algo 1

ii) Big - Omega (Ω) → $f_n = \Omega(g(n))$

This means that $g(n)$ is tight lower bound of $f(n)$

iii) Big - theta (Θ)

This gives us both the upper & lower bound.

iv) Small - Oh (o)

This gives just the upper bound. $f(n) = O(g(n))$

Q2)

A) $\log(n)$

if $n = 50$

then $i = 1, 2, 4, 8, 16, 32$

$\therefore \quad j = 2^k$

Hence i will take this much value i.e our program will run till 2^k

$\therefore \quad 2^k = n \Rightarrow \quad k = \log n$

Q3)

A) If $n = 4$

~~3x~~ $T(4) = 3 \times T(3) \rightarrow 3 \times T(2) \rightarrow 3 \times T(1) \rightarrow 3 \times T(0)$

81 27 9 3

Which is basically 3^k, k = no of calls.

Hence, $3^k = n \Rightarrow \quad k = \log_3(n)$

Q4)

Ans) $\log_2(n)$

Q5)

Ans) For $n = 15$

$j = 1, \quad s = 1$

$j = 2, \quad s = 3$

$j = 3, \quad s = 6$

$j = 4, \quad s = 10$

$j = 5, \quad s = 15$

The increment in 'j' is linear. But in 'S' we can observe that it is the sum of first 5 natural number (if $n = 5$)

So for k,

$n = 1 + 3 + 6 + 10 - -- k$

$= \frac{k(k+1)}{2}$ [The sum of first k natural num]

k -

$n = \frac{k^2 + k}{2}$ (Neglecting lower order terms)

$\therefore k = \sqrt{n}$

Hence, $O(\sqrt{n})$

Date : / /
Page No.

06)

Ans) Let $n = 15$

$i = 1 <= n$

$i = 4 <= n$

$i = 9 <= n$

$i = 16 <= n$ (X)

The value of 'i' is incrementing at with its Square.

~~So if we assume the val.~~

So if we find for a value of 'k'

Then, the last point at which the loop will stop will be near k^2 (i.e, $1^2, 2^2, 3^2, \ldots k^2$)

$\therefore n = k^2$

$k = \sqrt{n}$

Hence, ~~to~~ $O(\sqrt{n})$

07)

Ans) The j^{th} loop runs $n/2$ times (say p)

The j^{th} loop runs $\log_2(n)$ times

The k^{th} loop runs $\log_2(n)$ times.

$\therefore$ T.C $\rightarrow O\left(\frac{n}{2} * \log_2(n) * \log_2(n)\right)$

$\rightarrow O\left(n * (\log_2(n))^2\right)$

Q8)

Ans) if n = 15

$f(15) = f(12) \rightarrow f(9) \rightarrow f(6) \rightarrow f(3) \rightarrow f(1)$

~~Ans~~ The function returns in the order.

$1 \rightarrow 3 \rightarrow 6 \rightarrow 9 \rightarrow 12$

$a = 1, \quad r = \frac{3}{1} = 3$

$t_k = a r^{k-1}$

$n = 1 \times 3^{k-1}$

$\therefore n = \frac{3^k}{3} \Rightarrow k = \log_3 n$

~~Here~~, Also, there is a T.C of addition n^2 due to the loops inside the function.

$\therefore$ T.C $\rightarrow O(n^2 * \log_3(n))$

Q9)

Ans) The i'th loop runs 'n' times.

The j'th loop value depends on 'i'

i = 1, j = 1, 2, 3, 4, 5 ... j = 5, j = 1

i = 2, j = 1, 3, 5

i = 3, j = 1, 4

i = 4, j = 1, j = 5

We can ~~observe~~ observe that for each value of 'j' the j^{th} loop is running n/i times for a valid value of k

$$k = \frac{n}{1}, \frac{n}{2}, \frac{n}{3}, \frac{n}{4}, \frac{n}{5} \cdots \frac{n}{n}$$

$$= n\left(1 + \frac{1}{2} + \frac{1}{3} + \cdots \frac{1}{n}\right)$$

$\therefore k = \log(n)$

Hence, TC $\rightarrow$ $n(\log(n))$

i^{th} loop * j^{th} loop.

Q11)

Ans) The time complexity is $O(\sqrt{n})$

The increment of 'i' depends of value of 'j'

Let ~~n = 10~~ $n = 16$

So, $j = 1$, $i = 1 < n$

$j = 2$, $i = 3 < n$

$j = 3$, $i = 6 < n$

$j = 4$, $i = 10 < n$

$j = 5$, $i = 15 < n$

We can observe that the value of 'i' is the sum of first 'n' natural numbers.

So for a value of 'k'

k = 1, 3, 6, 10, ---k

= $\frac{k(k+1)}{2}$ ∫ The natural sum formula

$n = \frac{k^2+1}{2}$

or $k = \sqrt{n}$ (After dropping lower terms)

TC = $O(\sqrt{n})$

Q3)

Ans) i) for (i

Q3)

i) n log (n)

ii) for (i

Q3)

i) n log (n)

ii) for i in range (1, n+1):

for j in range (1, n+1, i):

print ("n log (n)")

ii) n^3

A)
```
for i in range(1, n+1):
    for j in range(1, n+1):
        for k in range(1, n+1):
            print("n^3")
```

iii)

~~iii)~~ for ~~ii~~ in range(2, n+1, pow(i, 2)): print(i)

Q15)

A) $O(n \times \sqrt{n})$

Q16)

A) Let's take $k = 2$ and $n = 50$

i) $j = 1 < 50$

$j = pow(1, 2) = 4 < 50$

$j = pow(4, 2) = 16 < 50$

$j = pow(16, 2) = 256 > 50$

We can observe that the value is incrementing at a rate of $(2^k)^k$

$2, (2^k)^k, ((2^k)^k)^k$ - - - -

$= 2, 2^{k^2}, 2^{k^3}, 2^{k^4}$ - - - -

which can be simplified to $2^{k^{\log_k(\log(n))}}$

∴ The last term must be equal to or less than n

$$\therefore \quad k^{\log_k(k_j(n))} = \log_{j_k}(n)$$

$$\therefore \quad T(n) = \log(\log(n))$$

Q18)

Ans i)

a) $100, \log(\log n), \log(n), \log(n!), \sqrt{n}, n\log n, n, n^2, 2^n, n!, 4^n, 2^{2^n}$

ii)

a) $1, \log(\log(n)), \sqrt{\log(n)}, \log(n), \log(2n), 2\times\log(n), \log(n!), n, n^{\log(n)}, 2n, 4n, n^2, 2(2^n), n!$

iii) $96, \log_2(n), \log_8(n), \log(n!), n\log_2(n), n\log_6(n), 5n, 8n^2, 7n^3, n!, 8n2^n$

Q19)

a) $j = 0, \quad j = 1$

Q2.)

a) Iterative

for i = 1 to n

key = arr[i]

j = i - 1

while j >= 0 and key < arr[j]

arr[j+1] = arr[j]

j = j - 1

arr[j+1] = key

Recursive

if ~~Size(arr)~~ n <= 1:

return

insertionSort (arr, n-1)

last = arr[n-1]

j = n-2

while j >= 0 and arr[j] > last

arr[j+1] = last

j = j - 1

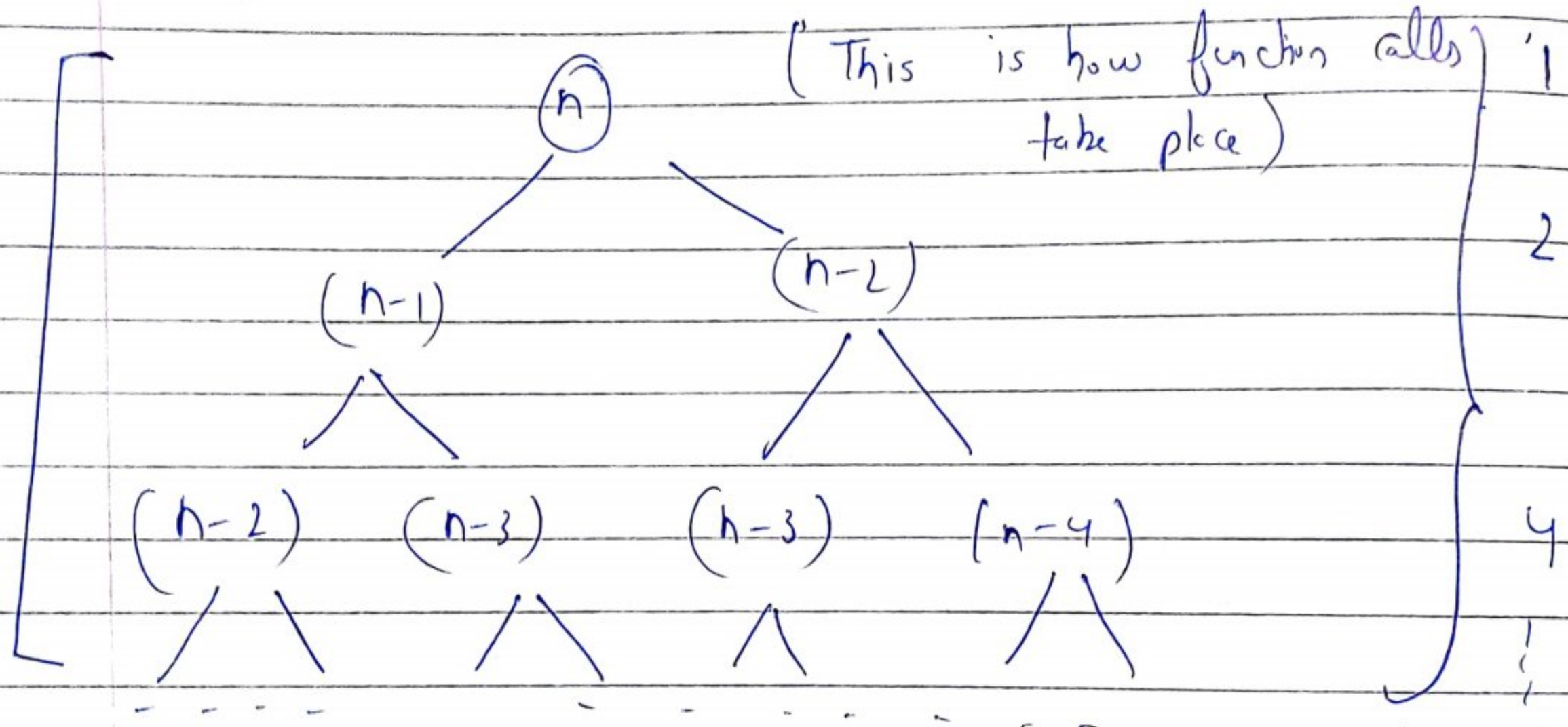
arr[j+1] = last.

Insertion Sort is called Online sorting algorithm as it does not make decisions based on the entire input array. It considers a single element and process a partial sorted arr.

Q12)

Ans) Recurrence Relation: $T(n) = T(n-1) + T(n-2) + 1$

Using Recursive Tree Method.



$TC = 1 + 2 + 4 + \cdots 2^n$

$a = 1, \quad r = \frac{2}{1} = 2$

T.C = Sum of 'n' terms of GP

$= \frac{a(r^{n+1} - 1)}{r - 1} = 2^{n+1} - 1$

$= O(2^n)$

If we consider the stack space then S.C → O(n)
else it will be O(1)

Q1)

Ans) Bubble Sort → ~~O(n)~~ O(n²) [W.C]

Selection Sort → O(n²) [W.C]

Insertion Sort → O(n²) [W.C]

Merge Sort → O(nlogn) [W.C]

Quick Sort → O(n²) [W.C]

Q2)

Ans) In-Place → Bubble Sort, Selection Sort, Insertion Sort

Online → ~~Bubble Sort~~ Selection Sort.

Stable → Bubble Sort, Insertion Sort, Merge Sort.

Q3)

Ans)

Q23)

A2

Iterative

low = 0

high = n-1

```
while low <= high
    mid = (low + high) / 2
    if arr[mid] = target
        return true.
    elif arr[mid] < target
        low = mid + 1
    else
        high = mid - 1
```

Recursive

```
binarySearch(arr, low, high)
    if low > high: return -1
    low = 0 mid = (low + high) / 2
    if arr[mid] = target
        return true
    elif arr[mid] < target
        return binarySearch(arr, mid+1, high)
    else
        return binarySearch(arr, low, mid-1)
```

Date : / /
Page No.

→ Iterative : TC → O(lg n)

SC → O(1)

→ Recursive : TC → O(lg n)

SC → O(lg n)

Q24)

Ans Recurrence Relation of BRS : $T(n) = T(\frac{n}{2}) + 1$