Consider Iterated logarithm of n, written $\log^* n$. Defined as below.

$$\lg^* N = \min\{k \mid \underbrace{\lg\lg\cdots\lg}_{k} N \leq 1\}$$

Which of the following is TRUE?

(log base can be considered as any constant)

A. $\lg\lg^* n = o(\lg^* \lg n)$ here o is small oh
B. $\lg^* \lg n = o(\lg \lg^* n)$ here o is small oh
C. $\lg^* \lg n = \Theta(\lg \lg^* n)$
D. $\lg^* \lg n$ and $\lg \lg^* n$ are not comparable

Your Answer: C | Correct Answer: A | Incorrect | Time taken: 02min 32sec | Discuss

$f(n) = O g(n)$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \begin{cases} 0 \\ \text{finite} \\ \text{infinite} \end{cases}$ — 0, Ω, Θ

$g(n) = o\, f(n)$

$f(n) = \omega\, g(n)$

$\lg^* N \leftarrow \log\left\{\min\{k \mid \underbrace{\lg\lg\cdots\lg}_{k} N \leq 1\}\right\}^a$

$\lg\lg^* N < o(\lg^* \lg N)$

$\lg\lg\lg(N) \leq 1$

$\lg\lg M = \lg N$

$M = \lg N$

$\min\{k \mid \underbrace{\lg\lg\cdots\lg}_{k} M \leq 1\}$ — k, $a-1$

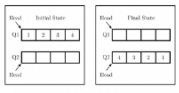
$= \lim_{a\to\infty} \frac{\lg(a)}{a-1}$

$= 0$

$a \to \infty$

8.11.12 Queue: GATE CSE 2022 | Question: 52

Consider the queues Q_1 containing four elements and Q_2 containing none (shown as the **Initial State** in the figure). The only operations allowed on these two queues are **Enqueue (Q, element)** and **Dequeue (Q)**. The minimum number of **Enqueue** operations on Q_1 required to place the elements of Q_1 in Q_2 in reverse order (shown as the **Final State** in the figure) without using any additional storage is _____.



gatecse-2022 numerical-answers data-structures queue 2-marks

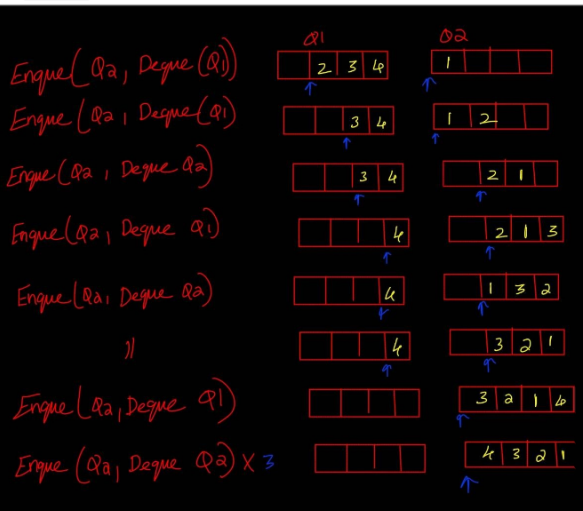
Answer key :

Hidden Layer 12/03/2025 7:12 PM

We don't need to solve puzzle to get answer zero.

Using induction we can reason that answer will be zero

1) If only two elements are there then it is possible for reversing que without external variable
2) if there are k elements in reverse order from 1st position to kth position and the k+1 the element is in the k +1 th position. This can be corrected using deque and enque k times.
3) if all elements in order then inserting one element only disturbs the the k +1 th element



Stack Implemented as dynamic array → New array of double the size allocated each time the stack is full.
→ All elements copied & old array deallocated.

Cost of n push if start with array of size 1.

$1 + 2 + 4 + \cdots 2^{\lfloor \log n \rfloor} = 2^{\lfloor \log n \rfloor + 1} - 1$

$= 2n - 1 = O(n)$

Amortized average time per push $= O(1)$

Array resized $O(\log n)$ times.

So if we want to implement stack using dynamic array with each push $O(1)$ on average ⇒ Double on overflow.

How to implement push(x), pop(), getMinimum() — Retrieve without remove. in a stack in $O(1)$ time for all 3 operation.

Solution 1 extra stack — Min stack

pop() → pop value from main stack, if same value in Min stack, then pop this also
push(x) → push value to main stack, if $x \leq$ top(Min) then push x to min stack also.
getMin() → return top of Min stack w/o poping

Q) Function f on stack S satisfies the following property.

$f(\emptyset) = 0$; $f(\text{push}(S, i)) = \max(f(S), 0) + i$ ∀ integers i

if $S = [2, -3, 2, -1, 2]$ then what if f(S)

$f(\emptyset)$ 0
$f(2)$ 2
$f([2,-3]) = -1$
$f([2,-3,2]) = 2$
$f([2,-3,2,-1]) = 1$
$f([2,-3,2,-1,2]) = 3$ //

Answer ③

☆ bottom-up approach for recursive questions

Another approach to implement getMin() operation in $O(1)$ time but without using extra array only extra 1 space available

strategy →

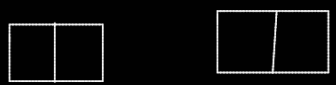
Push(x) → Case 1: Stack empty :- insert x in stack & set MinElement = x
→ Case 2: $x <$ Min Element :- insert ($k = 2x -$ MinElement) into stack
Set MinElement = x → $x <$ MinElement, $x -$ MinElement < 0, $2x -$ MinElement $< x$
Case 3: $x \geq$ Min Element :- insert (x) into stack.

Pop() → Remove k from stack → if stack empty return
→ Case I — $k <$ Min Element → temp = MinElement; MinElement = 2MinElement − k; return temp
→ Case II — $k >$ Min Element → return k.

Q) In merge sort algorithm we are given n elements → Each element is in turn a list of m items in sorted order → What is the complexity of Merge sort. → $O(m.n\log n)$ ☆

Q) In merge sort algorithm we are given n strings → Each string is of length m.
→ What is the complexity of Merge sort. → $O(m.n\log n)$ ☆

Q) In merge sort, what is the expected no. of comparisons required in merging 2 arrays each of length 2.



1 2 3 4 #permutations = 4! = 24
#we care = 3 = 24/8

min = 2
max = 3
} expectation btw 2 & 3

	X
1 2 3 4	2
1 3 2 4	3
1 4 2 3	3

$E(X) = \frac{3\times 2}{3} + \frac{2\times 1}{3}$

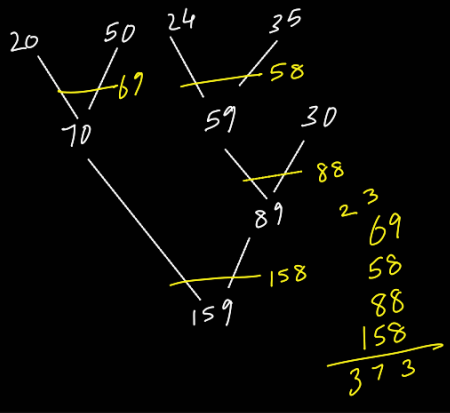
$E(X) = 2 + 2/3 = \boxed{8/3}$

Since 12 34 (2) 34 12
Since 21 34 (2) not allowed
12 43 (1) not allowed
21 43 (1) not allowed
8 combination

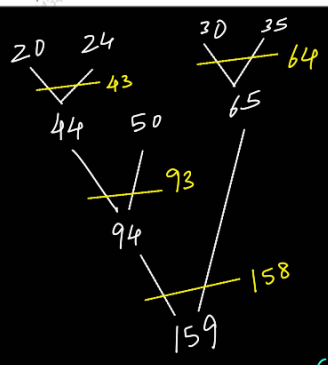
Suppose P, Q, R, S, T are sorted sequences having lengths 20, 24, 30, 35, 50 respectively. They are to be merged into a single sequence by merging together two sequences at a time. The number of comparisons that will be needed in the worst case by the optimal algorithm for doing this is ____.

A) 373
B) 382
C) 304
D) 358

HW



23
69
58
88
158
373



①
43 +
64 +
2 93
158
358

→ * Strategy: Huffman encoding → Always choose smallest 2 list available

☆

Huffman encoding / Huffman order (1-minute idea)

Problem type it solves:

> When you repeatedly combine items with costs, and combining two items of sizes a, b costs $a + b$, how do you minimize total cost?

Core rule (this is *the* definition)

👉 **Always combine the two smallest available items first.**
Repeat until only one item remains.

This greedy rule is called **Huffman order**.

Why it works (intuition)

- When you merge two items, their sum appears **again and again** in later merges.
- So **big numbers should be created as late as possible.**
- Merging the **smallest first** delays growth → minimal total cost.

Arrange the following functions in the ascending order of growth rate:

$f_1 = 2^{5 \log_{32} n}$ $= n$ — (3)

$f_2 = n\sqrt{\log_2 n}$ — (4)

$f_3 = \log(n\sqrt{n})$ — (2)

$f_4 = (\frac{n+4}{n})^{2n}$ $= \Theta(1)$ — (1)

$f_5 = 10n^2$ — (5)

$f_6 = n!$ (9)

$f_7 = (n/\lg n)^3$ $\left(\frac{2^k}{k}\right)^3$ — $3k\log 2 - 3\log k$ — (6)

$f_8 = 2^{n \log n}$ $2^{2^k \cdot k}$ — $2^k \cdot k \log 2$ — (10)

$f_9 = (\log n)^{\log n}$ k^k — $k \log k$ — (7)

$f_{10} = n^{(\log n)^{0.5}}$ $(2^k)^{k^{1/2}} = 2^{k^{3/2}}$ — $k^{1.5} \log 2$ — (8)

After arranging, the functions f_2 & f_{10} will come at which position, respectively?

$2^{5 \log_{32} n}$

$\log_{2^5} n = \frac{1}{\log_n 2^5} = \frac{1}{5 \log_n 2}$

$2^{\frac{1}{\log_n 2}}$

$\left(n^{\log_n 2}\right)^{\frac{1}{\log_n 2}} = n$

f_1 f_2 f_3 f_4 f_5 f_6 –

$\log n$ $\log(\frac{3}{2}\log n)$ $2n \log(\frac{n+4}{n})$ $2\log(10n)$ $n \log n$

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

$f_4 < f_3 < f_1 < f_2 < f_5 < f_7 < f_9 < f_{10} < f_6 < f_8$

1 2 3 4 5 6 7 8 9 10

$n = a^k$

$\log n = k$

f_9	f_{10}
k^k	$\left(2^k\right)^{(k)^{1/2}}$
k^k	$2^{k^{3/2}}$

taking log → $k \log k$ $k^{1.5} \log(2)$

n^2 $\log n^{\log n}$

$2 \log n$ $\log n \log(\log n)$

f_1 f_9

$\lg n$

$\left(\frac{n}{\lg n}\right)^3$ $\lg n$

$n = 2^k$

$\lg n = k$

$\left(\frac{2^k}{k}\right)^3$ k^k

taking lg $3k \cdot \lg 2 - 3\lg k < k \lg k$