(8-4)

Ex:- 1 | 2 | 4 | 6 | 8 | 9 | 10

(or 4 9)

→ Sum of n perfect squares = $\frac{n(n+1)(2n+1)}{6}$

→ Total possible perfect squares in N numbers (maximum) = $\sqrt{N}$

→ So using $n = \sqrt{N}$, sum of $\sqrt{N}$ perfect squares is $\frac{\sqrt{N}(\sqrt{N}+1)(2\sqrt{N}+1)}{6}$

$\Rightarrow \frac{(N+\sqrt{N})(2\sqrt{N}+1)}{6} \Rightarrow \frac{2N\sqrt{N}+2N+\sqrt{N}+N}{6}$

This is the time complexity for all operations. Now the Amortized time complexity per operation is:- $\frac{O(\sqrt{N})}{N}$

$\Rightarrow \frac{2N\sqrt{N}+3N+\sqrt{N}}{6\times N} \Rightarrow O\left(\frac{1}{\sqrt{N}}\right) \approx O(\sqrt{N})$ #

So the Amortized time Complexity is $O\left(\frac{1}{\sqrt{N}}\right)$ or $O(\sqrt{N})$ ##

∵ Considering only operations where i has a perfect square, and the rest are O.

8.3].

1. $T(n) = 8 \times T\left(\frac{n}{3}\right) + 2^n$.

In Masters theorem, this means $a = 8, b = 3, f(n) = 2^n$

⇒ For trying all cases, we need value of $n^{\log_b a}$ and ϵ.

$\Rightarrow n^{\log_b a} = n^{\log_3 8} = n^{1.892}$.

Comparing with all the 3 cases of Masters theorem,

$f(n) = 2^n > n^{1.892} \rightarrow$ (~~$n^{\log_b a}$~~ ~~rest~~ $n^{\log_b a}$)

$\therefore f(n) = \Omega(n^{\log_b a})$

Cs And from Case 3,

$f(n) = \Omega(n^{\log_b a + \epsilon})$ for $\epsilon > 0$.

And second condition to satisfy :-

$a f(n/b) \leq c \cdot f(n)$ for $c < 1$.

$\Rightarrow 8 f(n/3) \leq c \cdot f(n)$

↓

let $c = 0.8 = \frac{8}{10}$.

$\Rightarrow 8 \times f\left(\frac{n}{3}\right) \leq \frac{8}{10} f(n)$.

$\Rightarrow f(n) \geq 2^n$ ∀ assuming $n \geq 100$

$\Rightarrow \underbrace{8 \times 2^{\frac{100}{3}}}_{} \leq \underbrace{\frac{8}{10} \times 2^{100}}_{}$

$\Rightarrow 2^{\frac{100}{3}} \leq \frac{2^{100}}{10}$

↳ This inequality is valid. And so

$T(n) = \Theta(f(n)) = \boxed{\Theta(2^n)}$.

2). $T(n) = 3 \cdot T\left(\frac{n}{3}\right) + \frac{n}{2}$.

$\Rightarrow a = 3, b = 3, f(n) = \frac{n}{2}$.

Trying all three cases in Master's theorem

$\Rightarrow n^{\log_b a} \Rightarrow n^{\log_3 3} = n$ #.

$\Rightarrow f(n) = \frac{n}{2} \leq n. \rightarrow O(n)$

So, Case 2 $\rightarrow f(n) = O(n^{\log_b a})$

then $T(n) = O(n^{\log_b a} \log n)$.

$\Rightarrow$ ~~$O(n \lg n)$~~ $\Rightarrow$

$\Rightarrow T(n) = O(n \lg n)$ #.

3. $T(n) = 2 \times T(\frac{n}{4}) + \sqrt{n}$.

$a = 2, b = 4, f(n) = \sqrt{n} \rightarrow n^{0.5}$ $(n^{\frac{1}{2}})$

$\Rightarrow n^{\log_b a} = n^{\log_4 2} = n^{0.5}$

$f(n) = n^{\log_b a} = n^{0.5}$

So, using case 2,

$\Rightarrow f(n) = \theta(n^{\log_b a})$ then

$T(n) = \theta(n^{\log_b a} \lg n)$.

$\Rightarrow T(n) = \theta(n^{0.5} \lg n)$

or

$\boxed{\theta(\sqrt{n} \lg n)}$ #

4). $T(n) = 4 \times T\left(\frac{n}{2}\right) + 3n$.

$\Rightarrow a = 4,\ b = 2,\ f(n) = 3n$.

$\Rightarrow n^{\log_b a} = n^{\log_2 4} = n^{2 \log_2 2} = n^2$.

$\Rightarrow n^2 > 3n$.

$\rightarrow$ ~~g(n) ≤ c f(n)~~, so we can use case 1 as $f(n)$ is $O(n^2)$.

$\Rightarrow$ If $f(n) = O(n^{\log_b a - \epsilon})$ for some $\epsilon > 0$, then $T(n)$ is

$\ldots \log_b a)$

$\Rightarrow$ Assuming $\epsilon = 1$,

$O(n^{\log_2 4 - 1}) \Rightarrow O(n)$.

And, $f(n) = 3n \leq c \cdot O(n)$.

$\Rightarrow$ So, $T(n) = \theta(n^{\log_b a})$

$T(n) = \boxed{\theta(n^2)}$ ##

Q 2).

1) For inserting a new key we first calculate the new number of levels in the Amortized dictionary

This is split into

Adding new element needs to merge all elements before adding them all at some new level.

↳ This can at worst case be $2^{k\text{th}}$ element → combine all 2^{k-1} elements or just some t operations where $t < 2^k$.

→ There are a total of $\log n$ arrays in an Amortized dictionary and each array has 2^i elements or 0 elements.

level 0 → 2^0 ⇒ 1 element
level 1 → 2^1 ⇒ 2 elements
level 2 → 2^2 ⇒ 4 elements.
⋮
level n → 2^n elements.

} $\log n$ arrays or levels

→ So for merging elements in each array ⇒ Merge elements of previous list + Sort elements of new list (2 operations)

So, l0 ⇒ 2 ops, l1 → 4 ops, l2 → 8 ops
+ ... + ln → 2^{n+1} ops [($n/2$ + times), ($n/4$ + times)]

So, $O(\log n \times 2 \times n)$ → (number of elements in each level)
↓
(Total number of levels)

→ For Amortized insertion complexity :-

~~$O(\log n \times n)$~~

n

→ For each level, the merge happens $\frac{n}{2^l}$ times each with 2^l cost.
where 'l' is the level.

→ $O(\log n \times \frac{n}{2^l} \times 2^l)$ → 'l' is all levels.

∴ So total cost for n insertions is
$O(\log n \times n)$ → (each is $O(n)$ cost)

→ For Amortized insertion Complexity :-

$$\frac{O(\log n \times n)}{n} = \boxed{O(\log n)}$$

H.

[Search]

2). For ~~deletion~~ in an Amortized dictionary, since we are going through all lists in an Amortized dictionary ~~bcc~~ because each level is not dependent on any other levels.

So going ~~through~~ ~~each~~ all levels would be $O(\log n)$.

And in each level, we use Binary search to search for a value.
So, Binary search takes $O(n \log n)$

So, the total time would be :-

$$\Rightarrow O(\log n \times n \log n) = O(n \log^2 n)$$

And for the Amortized cost, for each search ~~at a level would~~ would be :-

$$\Rightarrow \frac{O(n \log^2 n)}{n} = \boxed{O(\log^2 n)}$$