Q.2

1. The merge cost of array is 2×2^i maximum.

For n insertions,

arr[0] will merge $n/2$ times
∴ cost $= 2$

arr[1] will merge $n/4$ times
∴ cost 4

and so on.

So, $O(\log n)$ arrays will merge with $O(n)$ cost

∴ total cost $= O(n \log n)$

But the amortized cost for the insertion will be $O(\log n)$

2. For search we go through all arrays present in different levels.

So, $\because$ we have $\log(n+1)$ arrays & searching a sorted array takes $\log(i)$ time where i is the no. of elements.

The longest array will be of length $\leq n$

& maximum $\log(n+1)$ arrays will be searched where each search is $O(\log n)$

$\therefore$ Total cost $\approx O(\log^2 n)$

Q.3

1. $T(n) = 8 * T(n/3) + 2^n$

$a = 8 \quad b = 3 \quad f(n) = 2^n$

Case 1: $f(n) = O(n^{\log_b a - \epsilon})$

$= n^{\log_3 8 - \epsilon}$

$2^n \neq n^{\log_3 8 - \epsilon}$

So case 1 doesn't hold

Case 2: $\log_b a = \log_3 8 = 3 \log_3 2$

$x = 3 \frac{\log 2}{\log 3}$

$f(n) \neq x$

So case 2 doesn't hold

Case 3: $f(n) = \Omega(n^{\log_b a + \epsilon}) \quad \epsilon > 0$

Now since exponential function grows faster than polynomial function so,

$$af\left(\frac{n}{b}\right) \le cf(n) \qquad c \le 1$$

$$8f\left(\frac{n}{3}\right) \le c2^n$$

$$8 \times 2^{n/3} \le c2^n$$

This condition will hold true for $c = 1$

$$8 \times 2^{n/3} \le 2^n \times 1$$

$$2^n \le 2^n$$

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

2. $T(n) = 3 \times T(n/3) + n/2$

Case 1: $a = 3$ $b = 3$ $f(n) = n/2$

$$\log_b a = \log_3 3 = 1$$

$$f(n) = O(n^{\log_b a - \epsilon}) = O(n^{1-\epsilon})$$

$\because$ ϵ should be > 0
but in this case if $\epsilon > 0$
then $n^{1-\epsilon} \neq n/2$

Then this doesn't hold

Case 2:
If $f(n) = \Theta(n^{\log_b a})$
then, $T(n) = \Theta(n^{\log_b a} \log n)$

In this case, $\Theta(n^{\log_3 3}) = \Theta(n)$

$f(n) = \frac{n}{2}$ where $1/2$ is constant

$f(n) = n = n^{\log_3 3}$

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

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

$a = 2 \quad b = 4 \quad f(n) = \sqrt{n}$

Case 1: If $f(n) = O(n^{\log_b a - \epsilon})$

$\log_b a = \log_4 2 = 0.5$

$\therefore n^{\log_b a - \epsilon} = n^{0.5 - \epsilon}$

$\because \epsilon$ should be > 0 this case won't hold

Case 2: $\log_b a = \log_4 2 = 0.5$

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

then, $T(n) = \Theta(n^{\log_b a} \log n)$

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

$\therefore T(n) = \Theta(\sqrt{n} \log n)$

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

$a = 4 \qquad b = 2 \qquad f(n) = 3n$

Case 1:

If $f(n) = O\left(n^{\log_b a - \epsilon}\right)$ for $\epsilon > 0$

$T(n) = \Theta\left(n^{\log_b a}\right)$

In this case, $\log_2 4 = 2$

$n^{\log_b a - \epsilon} = n^{2-\epsilon}$

If $\epsilon = 1$

$n^{2-1} = n = f(n)$

{ 3 is const. }

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

Q.4

n operations performed
if 'i' is a perfect square
operation costs i or
else it costs 0.

So,

operation		cost
1	$\longrightarrow$	1
2	$\longrightarrow$	0
3		0
4		4
5		0
6		0
7		0
8		0
9		9

So, every $2 \times i^{th} + 1$ operation costs i

$$\frac{2n+1}{n}$$

$$\frac{2\cancel{n}}{\cancel{n}} + \frac{1}{n}$$

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

So, amortized cost $= 2 + n^{-1}$
$= 2\, n^{-1}$