(Binary Search on Answers)

1) ~~Find~~ Square root of an integer :- or [max ans which on squaring <= n]

$\sqrt{25} = 5$ $\sqrt{35} = 5.25$ $\sqrt{36} = 6$

→ **Brute :-**

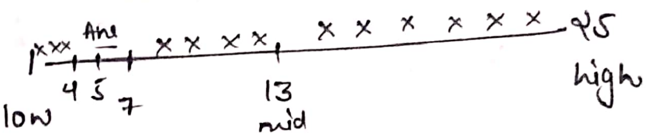
```
for( i=0 → n/2) {
    if( i*i == n) return i;
    if( i*i > n) return i-1;
}
```

~~return n/2;~~

TC :- O(N/2)
SC :- O(1)

→ **optimal :-**

for n = 25




```
⇒ low = 1, high = n    ans = 1
while ( low <= high) {
    mid = (low + high) / 2
    if( mid * mid <= n) {
        ans = mid ;
        low = mid+1 ;
    }
    else {
        high = mid-1
    }
}
return high
```

TC :- O(logn)
SC :- O(1)

Note :- BS → answers

[lies in a range]
min max

⇒ whenever you have to find min or max in a given or fix range (1 to n) then apply BS on Answers

2) N^{th} root of a number ?-

```
// helper fcn to compute mid^n
Power( mid, n, m) {
    ans = 1
    for(i=0 -> n) {
        ans = ans * mid
        if (ans > m) return ans
    }
    return ans
}

// main fcn
Nthroot (n, m) {
    low=1 , high=m
    while ( low<=high) {
        mid= (low+high)/2;
        val= Power (mid, n, m)
        if(val== m) return mid
        else if(val > m) high= mid-1
        else low= mid+1
    }
    return-1;
}
```

TC :- $[O(\log_2(m) \times O(n)]$

SC :- $O(1)$

3) Koko eating bananas :-

arr = [7, 5, 3, 11]   h = 8   speed = 4 bananas per hour

⇒ **Brute** :-

```
required time (nums, hourly) {
    int totalhrs = 0
    for (i=0 → nums.size()) {
        totalhrs += ((nums[i] + hourly - 1) / hourly) → calculate ceil
    }
    return totalhrs
}
```

```
// linear Search Approach
min Rate (nums, h) {
    for (i=1 → max(nums)) {
        reqtime = required time (nums, i)
        if(reqtime <= h) {
            return i
        }
    }
    return -1
}
```

TC :- O(max(nums)) × O(N)

SC :- O(1)

⇒ **Optimal** :-

⇒ we know a range [1 ——— max(nums)]

min can eat 1 banana per hour

max can eat 11 " "

1	2	3	$\boxed{4}$	5	6	7	8	9	10	11
×	×	×	✓	✓	✓	✓	✓	✓	✓	✓

↓ Ans

ode :-

```
requiredtime ( ); → like brute
minimumRate ( ans, h) {
  low = 1 ,  high = max(nums)
  while (low <= high) {
     mid = (low+high)/2;
     requiredtime = required-time ( nums , mid)
     if (requiredtime <= h) {
          high = mid-1;
     }
     else {
          low = mid+1;
     }
  }
return low
}
```

TC :- $O(\log_2(\max(ans)) \times O(N)$

SC :- $O(1)$

Minimum Days to make M bouget :-

arr [ ] = { 7, 7, 7, 7, 13, 11, 12, 7} m=2, k=3

m ↓ no. of bouget

k ↓ adjacent no. of flowers

→ Brute :-

```
//helper fcn
countBouget ( ) {
   flower=0,  bouget=0
```

```
for (i=0 -> n) {
        if(nums[i]<=day) {
            flowers ++;
            if(flowers ==k) {
                bouget ++;
                flowers =0;
            }
        }
        else {
            flowers =0;
        }
}
return bouget>=m
}

//linear search fcn
minDays () {
    for(i=1-> max (nums)) {
        if(countBouget (---)) {
            return i
        }
    }
return -1;
}
```

TC :- O(max (nums)).
O(N)

SC :- O(1)

optimal :-

```
countBouget ();
minDays () {
    low=1 , ans=-1,  high =max(ans)
    while (low <=high) {
        mid= low + (high-low)/2;
        if(countBouget(--)) {
            ans =mid
            high=mid-1
        }
        else {
            low=mid+1;
        }
    }
return ans
```

TC :- $O(\log_2(\max(ans)) \times O(N))$

SC :- $O(1)$

Scanned with OKEN Scanner