3. WRITTEN PROBLEMS.

(a)
```
def foo1(lst):
    i = len(lst) * 2          → θ[1]
    for j in range(len(lst) - 1):
        O[1][lst[j], lst[j+1] = lst[j+1], lst[
        if i+j <= j:
            break
    i //= 2
```

len(lst) is always +ve. ∴ i is always +ve.
But eventually i//=2, will reach 0.
In that case i+j = j and we will break out of the foor loop. So, we have to find out how much time it takes for i to become 0?

$\theta(\log_4 n)$

(b)
```
def foo2(lst):
    n = len(lst).
    for i in range(n-1):
        j = i+1
        while j>1:
            lst[j] = 3*j - 1
            if lst[j] > lst[j-1]/2:
                j -= 1
            else:
                j -= 3
```

if only this part happens, it happens only in $O(n^2)$

→ if only this part happens, then too it runs roughly $O(n^2)$ times.

Avg case = $O(n^2)$

```
def sum_list1(lst, low, high):
    print("sum_list1, low= ", low, ", high= ", high, sep = "")
    if low == high:
        return lst[low]
    else:
        sum_of_rest = sum_list1(lst, low+1, high)
        total_sum = lst[low] + sum_of_rest
        return total_sum
```

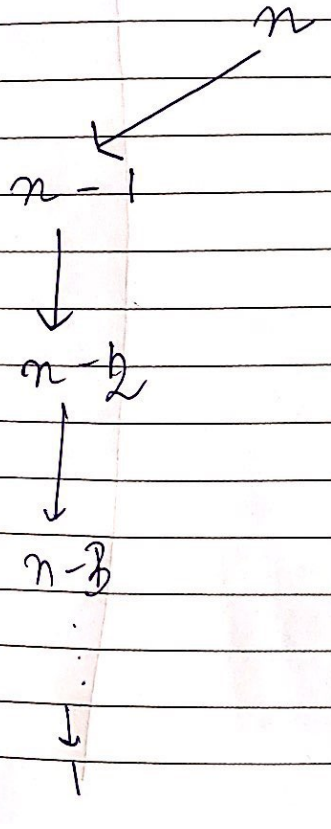
I = [1, 2, 3, 4, 5, 6, 7, 8] low = 0, high = 7

REQUIRED OUTPUT

sum_list1, low= 0, high= 7
sum_list1, low= 1, high= 7
" " " 2, high= 7
" " " 3 " "
" " " 4 "
" " " 5 "
" " " 6 "
" " " 7 "

REQ

O(1)



in each node, apart from the recursive step, only O(1) is done and number of nodes will be len(list)
∴ O(n) is the running time.

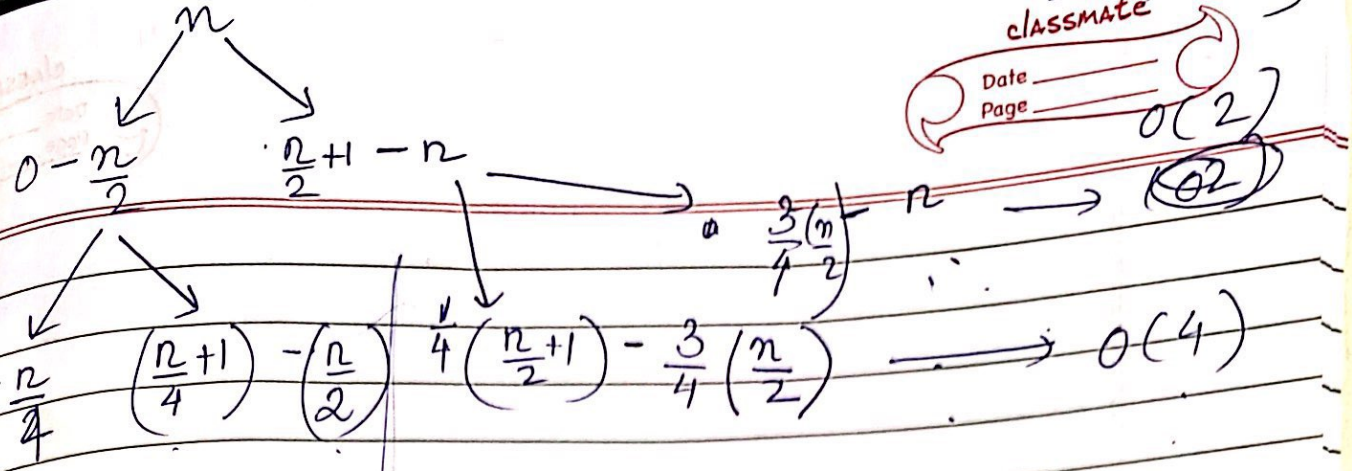
```
def Sum_list2 (lst, low, high):
    print ("Sum_lst2, low=", low, " ,high=", high, sep=" ")
    if low == high:
        return lst[low]
    else:
        mid = (low+high) // 2
        leftsum = sum_list2 (lst, low, mid)
        rightsum = sum_list2 (lst, mid+1, high)
        totalsum = leftsum + rightsum
        return totalsum.
```

Input = [1, 2, 3, 4, 5, 6, 7, 8] low=0 | high=7

REQUIRED OUTPUT

Sum_lst2, low = 0, high = 7
Sum_lst2, low = 0, high = 3
sum_list2, low = 0, high = 1.
Sum_lst2, low = 0, high = 0
Sum_lst2, low = 2, high 3
Sum_lst2, low = 4, high 7

Sum_lst2, low = 0, high = 7
Sum list2 " 0 " 3
" " 0 " 1
" " 0 " 0
" " 1 " 1
" " 2 " 3
" " 2 " 2
" " 3 " 3
" " 4 " 7
" " 4 " 5
" " 4 " 4
" " 5 " 5
" " 6 " 7
" " 6 " 6
" " 7 " 7



∴ Here instead of counting the nodes, we count how much work is done at each level.

∴ Total work:

1 + 2 + 4 + 8 . . . . .

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

$$= 1\,\frac{(1-2^{\log_2 n})}{(1-2)} = \frac{(1-n)}{-1} \approx O(n)$$

till low=high

Therefore depth of the tree is $\log n$