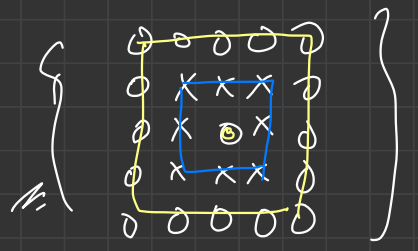
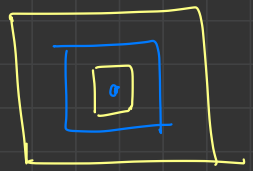
Q. given n, m: n = 5, m = 5





alternate rectangles of O's and X's

Subarrays

a = { 1 2 3 4 }

1, 2, 3, 4, 12, 23, 34, ~~13~~, ~~24~~, ~~14~~,

123, 234, ~~134~~, ~~124~~, 1234

the part of array obtained by deleting (possibly 0) elements from start and end.

a = {8 5 6 7}
0 1 2 3

start indx , end indx.

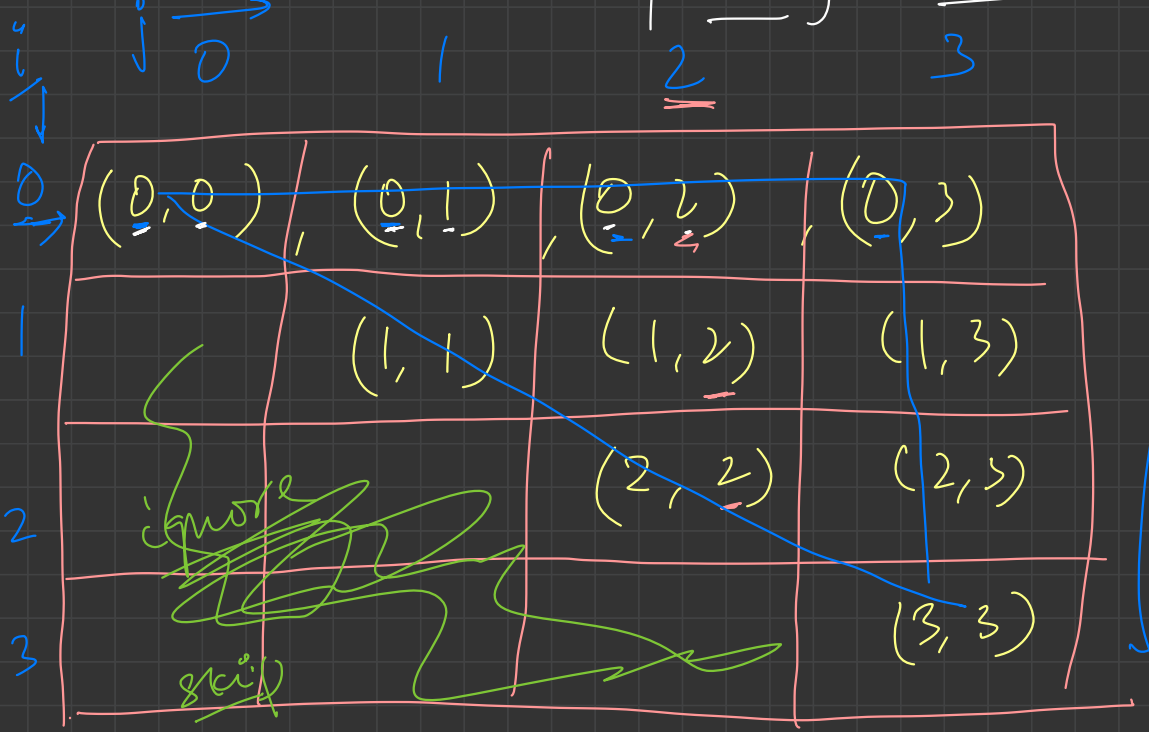
sub-arrays

8	0,0	85	0,1	856	0,2
5	1,1	56	1,2	567	1,3
6	2,2	67	2,3	8567	0,3
7	3,3				

Subarrays are characterized by its starting and ending index.

⇓

pair of indices (l, r) $\boxed{l \le r}$




```
for(i=0; i<n; i++){
    for(j=i; j<n; j++){
        i, j
    }
}
```

arr = { 9, 5, 8, 4 }
0 1 2 3

(1, 4)
len

9 5 8 4 !

9 5 | 5 8 | 8 4 | 4 * | 2

9 5 8 | 5 8 4 | 8 4 * | 4 * | 3

9 5 8 4 | 5 8 4 * | 8 4 * | 4 * | 4

if I tell you the length of the subarray and its starting index can you print the subarray?

arr = {9, 5, 8, 4}

length	starting indices
1	0, 1, 2, 3
2	0, 1, 2
3	0, 1
4	0

arr: {9, 5, 8, 4}.
0, 1, 2, 3

upto

n - l + 1 th index

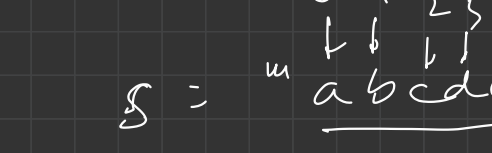
n = 4 l = 3

n - l + 1 = 4 - 3 + 1

= 1 + 1 = (2)

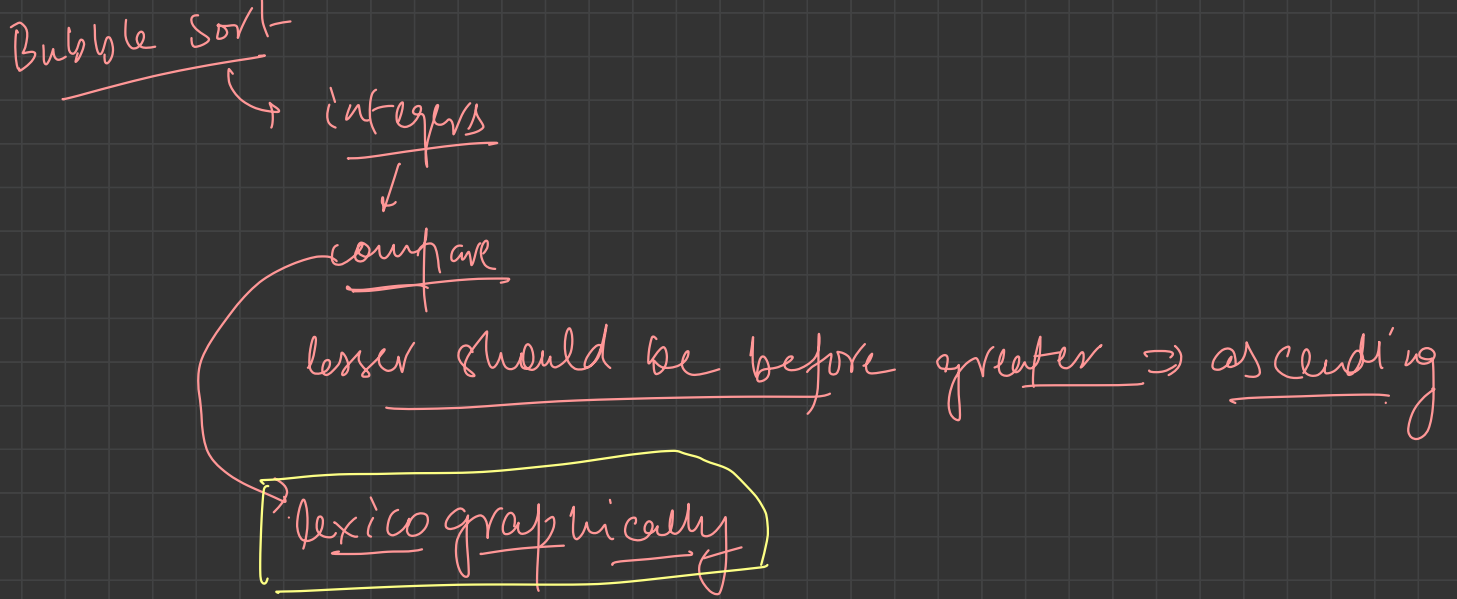
```
for(l=1; l<=n; l++){                    →  i <= n-l
    for(i=0; i < n-l+1; i++){
        for(k=0; k < l; k++){
            a[k+i];
        }
    }
}
```

Subarray :- Substrings

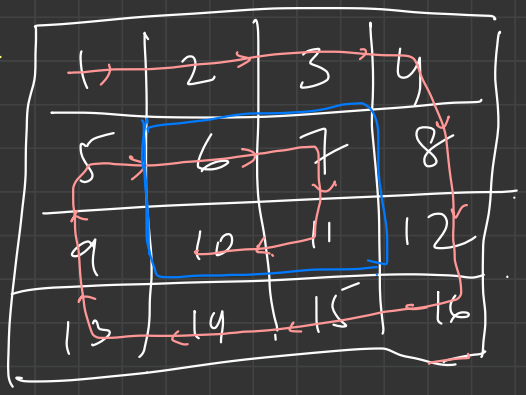


a, b, c, d, e, f, ab, bc, de, ef, - - - -

Q. Print all substrings for a given string.



Q.



→ 1, 2, 3, 4, 8, 12, 16, 15, 14, 13, 9, 5, 6, 7, 11, 10

first row (top)

last column (rightmost)

last row (bottom)

first column (left)

top-row → right-col → bottom-row → left-col

loop ( ) {

repetition of values

top-row : left to right

right-col : top to bottom

bottom-row : right to left

left-col : bottom - up

}

① TR from (LC to RC)

[1, 2, 3, 4]

② RC (from TR to BR)

[8, 12, 16]

③ BR (from RC to LC) [15, 14, 13]

④ LC (from BR to TR) [9, 5]

⑤ TR (from LC to RC) [6, 7]

⑥ RC (from TR to BR) [11]

⑦ BR (from RC to LC) ⑩

⑧ LC (from BR to TR)

left col < right col

and

top row < bottom row

Binary Search

Search in array → by going at each index and checking whether its the same element

Linear Search

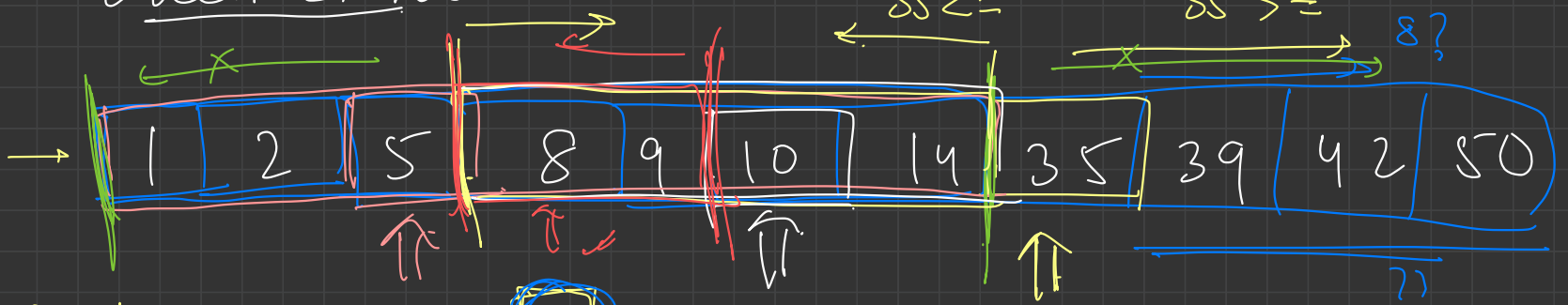
→ in the worst situation we may need to look into all indices

→ it depends no. of elements / indices

no. of lookups needed $\propto$ size of array
no. of elements / indices

n

Linear Search:



Find out whether (8) is present in the dict. ??

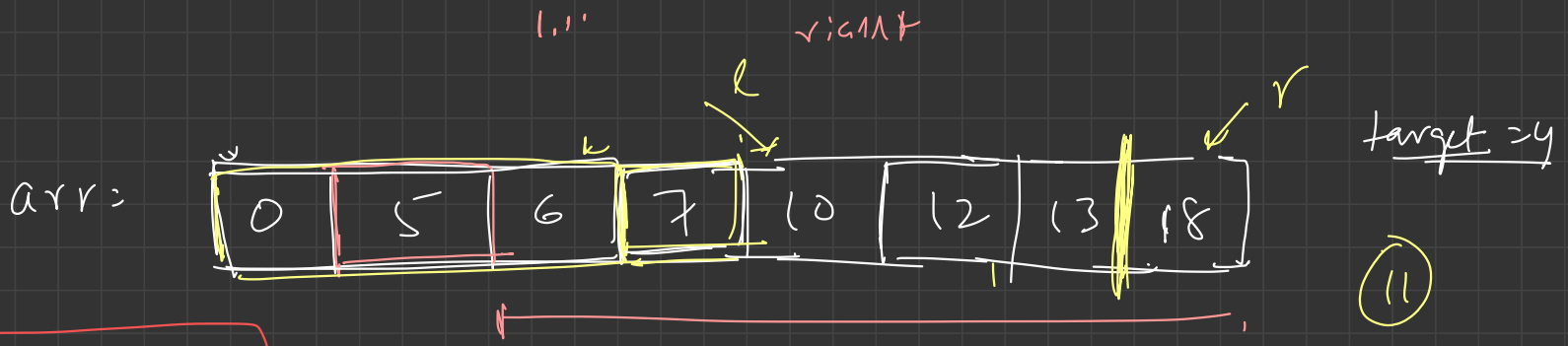
target

→ whatever page (index) you are at

— if the target value is >, look for it on the right pages (indices)

(8 < 10)

— if the target value is <, look for it on the left pages (indices)



target = 14

check the middle element → ??

↓

at the next we only have half elements to check from.

a[mid] = 7

target val = 14

a[mid] < target_val ⇒ towards right

a[mid] = 12 target_val = 14

a[mid] < target_val ⇒ right

a[mid] = 13 target_val = 13

a[mid] < target_val ⇒ right

a[mid] = 18 target_val = 14

a[mid] > target_val ⇒ left

left border right border

have overlapped ⇒ entire array checked

→ 14 doesn't exist

l = 0, r = n-1

while (l <= r) {

find mid;

if (a[mid] == target) → found

else if (a[mid] > target) → go left : shift the right (r) just before mid

else (a[mid] < target) → go right;

left = mid + 1;

}

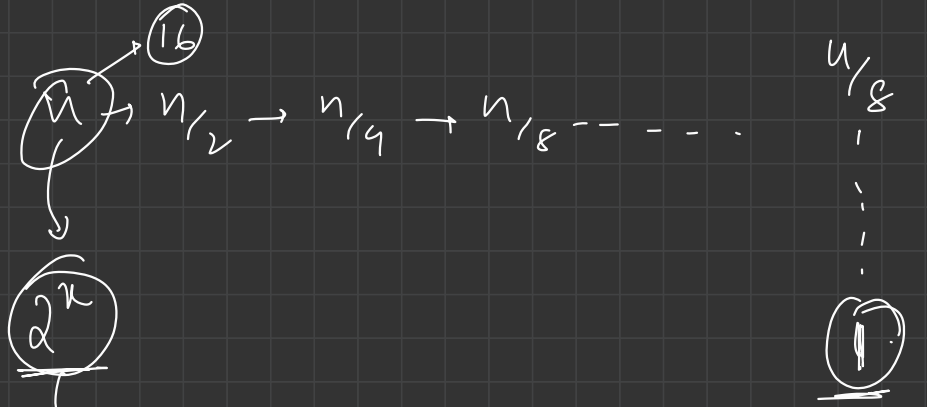
Not found

initially elements → n

n/2

n/4

after one step



we find / consider

$2^n >= n$ (min, n)

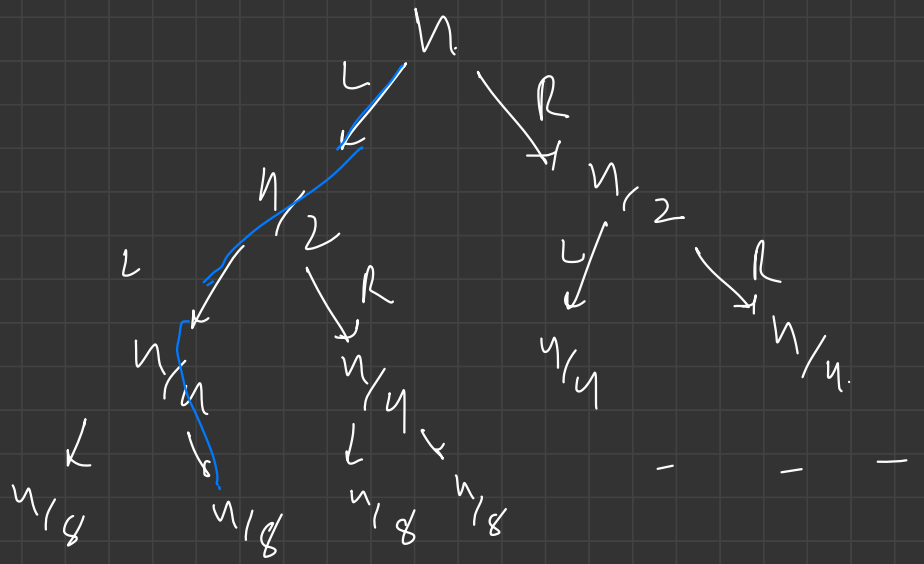
2^{n-1}

2^{n-2}

2^{n-3}

$1 = 2^0$

$n, n-1, n-2, \ldots, 0 \Rightarrow (n+1)$ steps



at each step no. of elements is halved,

last level has only element!

$n \to n/2 \to n/4 \to n/8 \to n/16 \to \cdots\cdots \to 1$

$2^x \to \frac{2^x}{2^1} = 2^{x-1} \to 2^{x-2} \to 2^{x-3} \to 2^{x-4} \to \cdots \to 1 = 2^0$

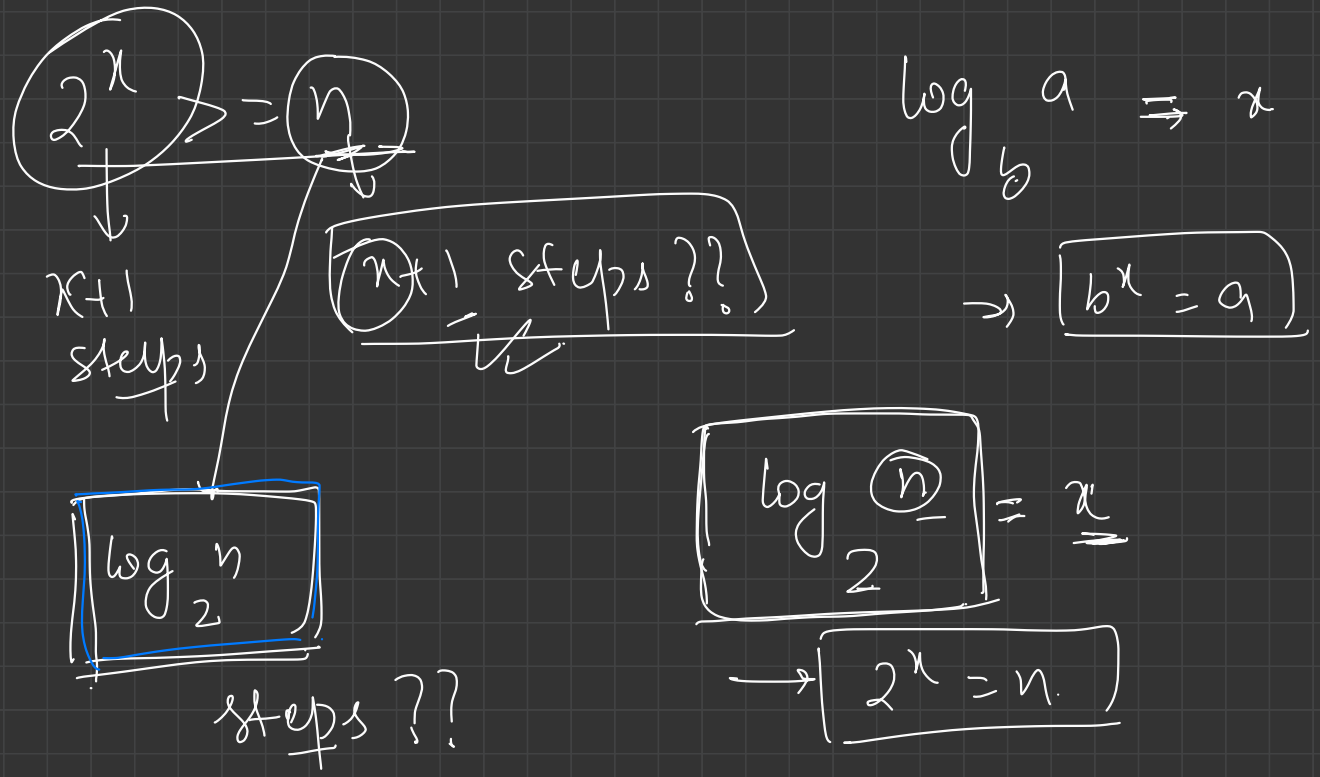
$2^x, 2^{x-1}, 2^{x-2}, 2^{x-3}, \cdots\cdots 2^0$

$x, x-1, x-2, \cdots$ no. of steps $\to \boxed{x+1}$

min value of x, such that $2^x >= n$

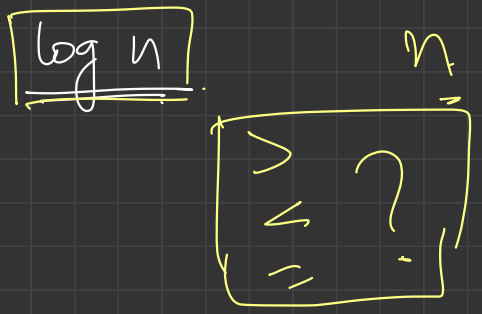
↳ if there are 2^x elements in an array then we need at most $\boxed{x+1}$ steps/search to find (not find) the element.)

complete the search



When I use Binary Search in a sorted array

then no. of steps needed are at most $(\log_2 n)$, where n is the size of array.



→ Binary Search is much more efficient than linear search but, it needs the array to be sorted.

Q. You are given a sorted, array and a value Q. Find how many values in the array are less or equal to Q.?

1, 1, 2, 2, 3, 4, 5, 8, 9, 9, 10, 12, 13, 14, 14, 15

d = 5

output → 7

freq[arr[i]]++;

freq[4]

freq:

0	0	0	0	0	0
0	1	2	3	4	5

the value at the index, represents the frequency/count of the index-value freq[3] → count of 3's