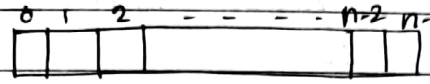
12/07/21

8. Computational Complexity :-

1. **Space & Time complexity – Searching for a number in a list :-**

→ Mathematical way to measure / quantify (program/algo/soln) takes how much space and time.

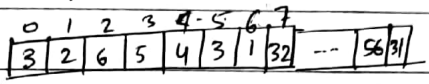
→ Ex :-



list of size = n

prob :- given a no. 'q' find if 'q' exists in l.

soln :-



find q = 32

Case-1 Go to first index 0, it is not 32

" " 2nd " 1, " " " 32

⋮ ⋮ ⋮ ⋮

At some point index = 7 we see 32

∴ 32 exists in l at index = 7

Case-2 q = 325

∴ Check all and you will not find any 325. So q is not present

Case-3 q = 31

∴ it is the last element

∴ 31 is present in list.

For case-1,
we did 8 comparisons.

For case-2,
we did n comparisons.

For Case-3,
We did n comparisons.

If a list has n element,

① How many comparisons are needed?

↳ n comparison → (in worst case)

↳ 1 " → (best case)

~ n " → (avg ")

as n increases, no. of comparisons increase proportional to n.

∴ no. of comparison ∝ n.

each comparison take some time.

12/07/21

Code example :-

```
import numpy as np
import random
l = list (range(100))
random.shuffle(l)
l
```

(Not sorted, randomly suffled list)

search for an element q in the list : O(n), where n is the length of list.

l = length of n

```
q = 31                          → takes 1 unit of time
isFound = False;                → 1 unit of time
for ele in l :                  → 1 unit of time
    if ele == 31 :              → 1 unit of time
        print("found")          → 1 unit of time
        isFound = True;         → 1 unit of time
        break;                  → 1 unit of time
if isFound == False :           → 1 unit of time
    print ("Not Found")         → 1 unit of time
```

This loop can run for n times. (n, n, n, n)

Total time = 1+1+ n+n+n+n + 1+1 = 4n+4 units of time

Where, n → no. of elements in l.

Total time ∝ n

→ **Time complexity = O(n)**

Time units		time complexity.
4n+4	→	O(n)
$3n^2+2n+4\times1$	→	$O(n^2)$
4 logn + 3	→	log(n)
5 ×1	→	O(1)

input of size n

$n^2, n \geq 1$

n=1, n^2=1

n=2, 4 > 2 > 1

n=3, 9 > 3 > 1

(we take the largest time)

logn+1

logn ≥ 1

Scanned with CamScanner

02/07/21

2. **Binary Search :-**

l =  ← list of size n.

input { q = 31

prob :- is q present in l or not? → O(n) time
→ O(1) space (Because we didn't use extra space)

lower time complexity → better.

space " → " .

O(n) time and O(1) space ← sequential search.

Using Binary search ↓

O(log n)

$\log_2(n) \leq n$ n = 1, $\log_2(n) = 0$

n = 2 ; $\log_2(n) = 1$

n = 4 ; $\log_2(n) = 2$

n = 8 ; $\log_2(n) = 3$

n = 16 ; $\log_2(n) = 4$

O(n) = 1024 comparison

$O(\log_2 n) = 10$ "

Binary Search :- ① list needs to be sorted. q = n-1

l:	0	1	2	3	4	5	6	7	8	9 (8)
	23	25	28	32	37	42	48	49	56	59

← l (sorted)

q = 25

i) middle element = $\left\lfloor \frac{left + right}{2} \right\rfloor = \frac{9}{2} = 5$

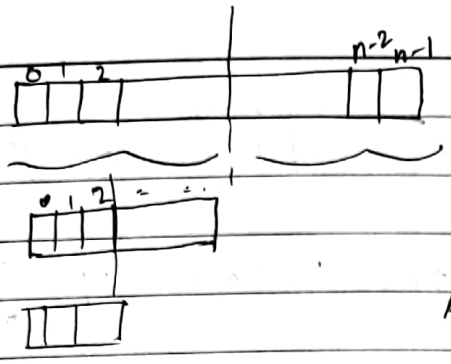
ii) Go to index 5 and check value.
25 < 42. So 25 should be at left (because list is sorted)

iii) Change r to index 5.
now middle element = $\lfloor \frac{0+5}{2} \rfloor = 3$
25 < 32. So it should be on left of 32.

iv) r = 3
$\therefore m = \lfloor \frac{0+3}{2} \rfloor = 2$
28 > 25.

v) r = 2.
$m = \frac{0+2}{2} = 1$
∴ 25 is present at index 1.

12/07/21



1st comp. (n)

I have $n/2$ elements after 1st comparison.

After 2nd comp., $n/4$ elements

" 3rd " , $n/8$ "

4th " , $n/16$ "

(?) ⟶ 1 "

$\therefore$ 4 → 16

5 → 32

3 → 8

$\log_2 16 = 4$.

(There is a logarithmic relation)

$\therefore$ This algorithm takes $O(\log_2 n)$ time / comparison.

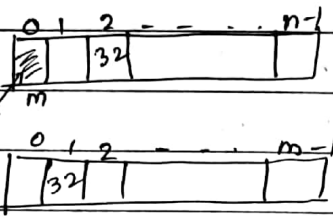
Code example:-

```
import math
# Returns index of x in arr if present, else -1
def binarySearch (arr, l, r, x):
    # check base case.
    if r >= l:
        mid = l + math.floor ((r-l)/2)
        # if element is present at the middle itself
        if arr[mid] == x:
            return mid
        # if ele is smaller than mid, it can only be present at left subarray.
        elif arr [mid] > x:
            return binarySearch (arr, l, mid-1, x)
        # Else the ele can be present at right subarray
        else: return binarySearch (arr, mid+1, r, x)
    else:
        # Element is not present in the array
        return -1.

J. sort ():
    arr = 1
    q = 1
    binarySearch (arr, 0, len(arr) -1, q)
```

13/07/21

3) Find elements common in two lists :-



→ n elements.

→ m elements.

l_1 & l_2 are not sorted.

problem :- Find elements that are common in l_1 & l_2.

Soln :-

let's assume $m \leq n$

for 1st element we are doing m comparisons. to check if it is present in l_2

2nd " " " m " " "

n-1th " " " m " " " "

∴ Total comparisons = $m + m + m + \cdots$ n times $= n * m$.

∴ time complexity = $O(n*m)$

∴ Space complexity = $O(1)$ because we are not using any other space.

Code example :-

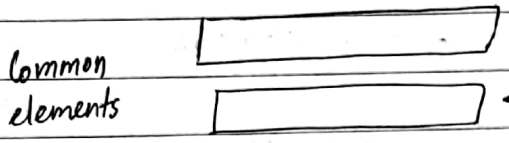
```
# Find elements common in two lists:
l1 = list(range(100))
random.shuffle(11)
l2 = list(range(50))
random.shuffle(12)

# Find common elements: O(n*m)
cnt = 0;
for i in l1:                    ← This loop runs n times
    for j in l2:                ← This loop runs m times. (l2 → m)
        if i == j:
            print(i)
            cnt += 1;
print("Number of common elements:", cnt)
```

Total $O(n*m)$

13/07/21

4. Find elements common in a list using a hashtable or dictionaries:-

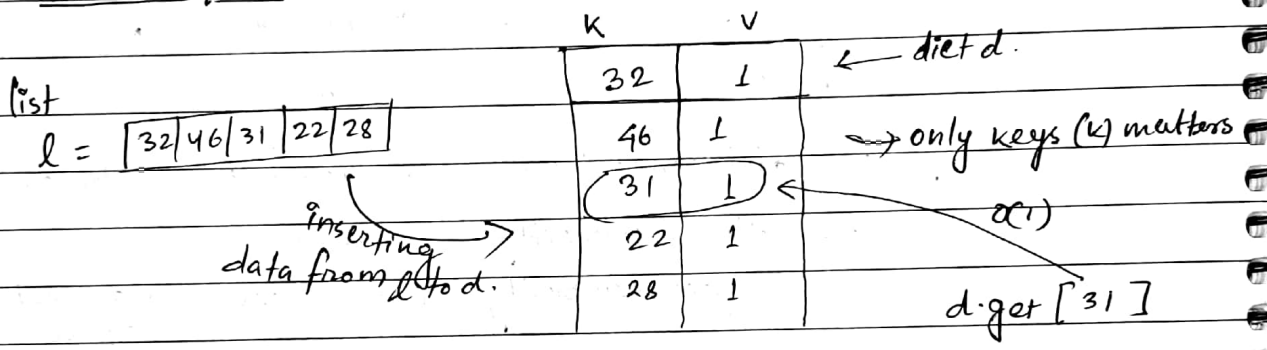


$O(n*m)$ time
$O(1)$ Space.
$n \geq m$

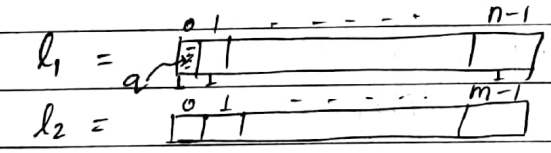
Python dictionary → Hashmap & Hash Table.
Trade off b/w space & time.

we will try to do
$O(n)$ time
$O(m)$ space.

Hashtable / Diet. :-



property of d:-
I can go directly to the key. (means in $O(1)$ search time). It uses hash functions.



$m \leq n$

① store l_2 in a dictionary d → it uses $O(m)$ space & $O(m)$ time.
↳ size m

② Sequentially search l_1 elements in d.
each search takes 1 time. So, total $O(n)$ time

Total time ~~$O(m)+O(n)$~~ $O(m+n)$ time
$O(m)$ space.

$m+n \leq n*m$

13/07/21

Code example :-

```
# Find elements common in two lists :
l1 = list (range (100))
random. shuffle (l1)
l2 = list (range (50))
random. shuffle (l2)

# Find common element in lists in O(n) time and O(m) space if m<n
# add all elements in the smallest list into a hash table / Dict :
                                                        O(m) space.
O(m) time  {  small List = { }
O(m) space {  for ele in l2 :
                  smallList [ele] = 1 ; # any value is ok. key is important.

# Now find common element
   cnt = 0 ;
      { for i in l1 :
O(n)  {       if smallList .get(i) != None :  # search happens in constant
Time  {             print(i);                                       time.
      {             cnt+ = 1 ;
   print ("Number of common elements:", cnt)
```