DSA

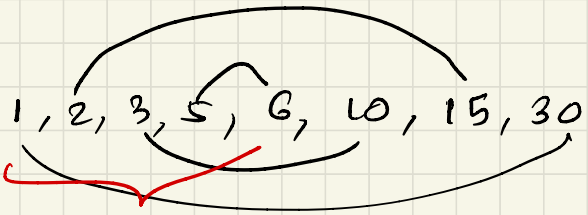
Contents

Lecture TC-1

(1)

```
def isPrime(num):
    n = int(num)

    count = 0
    for i in range(1, √n+1)
        if n % i == 0:
            count += 2

    if count == 2
        return True
    else:
        return False
```

num = 30



$\sqrt{n}+1$

$5.x + 1$

range (1, 6.x)

Lecture TC-2

Big O notations

$$\log N < \sqrt{N} < N < N\log N < N\sqrt{N} < N^2 < N^3 < 2^N < 3^N < N!$$

A.P. & G.P.

$T_n = a + (n-1)d$

$T_K = ar^{K-1}$

$S_n = \frac{n}{2}[2a + (n-1)d]$

$S_K = \frac{a(r^K - 1)}{r-1}$

Lecture 3 : Introduction to Arrays

(2) Given n elements in a list. count no. of elements which have at least 1 element greater than itself present in the list

[2, 5, 1, 4, 8, 0, 8, 1, 3, 8]

→ 8 - 3 = 5

Brute Force Approach :

```
cnt = 0
for i in range(len(l)):
    for j in range(len(l)):
        if l[j] > l[i]
            cnt += 0
            break
return cnt
```

$O(N^2)$

Optimized Algorithm :

1. find largest value & no. of times it occurs
2. Return n - cnt ← cnt

```
max = l[0]
for i in range(len(l)):
    if l[i] > max:
        max = l[i]

cnt = 0
for i in range(0, len(l)):
    if l[i] == max:
        cnt += 1

return n - cnt
```

O(N)

③ Given N array elements, check if there exists a pair (i, j) s.t. arr[i] + arr[j] == k

arr : [3, -2, 1, 5, 6]

(0,0) (0,1) (0,2) (0,3) (0,4)

(1,1) (1,2) (1,3) (1,4)

(2,2) (2,3) (2,4)

(3,3) (3,4)

(4,4)

```
for i in range(0, len(l)-1):
    for j in range(i+1, len(l)-1):
        if arr[i] + arr[j] == k:
            return True
return False
```

for i = 0 j = 1, N
i = 1 j = 2, N

$O(N^2)$

iterations = $\frac{n(n-1)}{2}$

④ Given a list, reverse the list

Brute force

```
def reverse_list(l):
    return l[::-1]
```

O(N)

S(N)

Optimized Algorithm

swapping

```
n = len(l)
for i in range(0, n//2):
    l[i], l[n-1-i] = l[n-1-i], l[i]
```

O(N) S(1)

⑤ Given the list, rotate the list (right rotation)

A = 1, 2, 3, 4 → 4, 1, 2, 3 → 3, 4, 1, 2

B = 2

```
B = B % n
part 1 = A[ : (n - B)]
part 2 = A[ -B : ]
for x in part 1 :
    part2 append (x)
```

Lecture 4: Prefix sum

⑥ Given an array of N elements, find sum of all elements from index "s" to "e"

Brute Force

```
sum = 0
for i in range (s, e+1):
    sum += A[i]

return sum
```

TC: O(N)

SC: O(1)

But if there are multiple pairs of (s,e)

TC: O(N * Q) SC: O(1)

Prefix sum: sum of all elements starting from s to e

```
ps = [0]*N
ps[0] = A[0]
for i in range(1,N):
    ps[i] = ps[i-1] + A[i]
```

sum from index 4 to 8

ps[8] − ps[3]

sum from index 0 to 6

ps[6]

(7) Find the equilibrium index of an array

↳ sum on left == sum on right

```
for i in range(1,n):
    if ps[i-1] == ps[n-1] - ps[i]
        return True
return False
```

	1	2	3	10	3	2	1
	0	1	2	3	4	5	6

ps[2] == ps[6] - ps[3]

(8) Find the count of special indices in an array if that element is removed,

sum of even-indexes = sum of odd-indexes

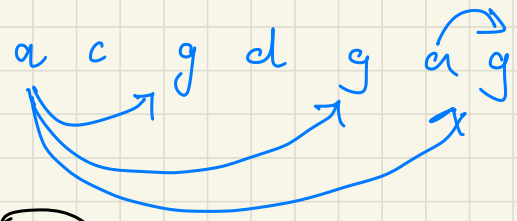
```
N = len(A)
even_PS = [0 for i in range(N)]
odd_PS = [0 for i in range(N)]

even_PS[0] = A[0]
odd_PS[0] = 0

for i in range(1, N):
    if i%2 == 0:
        even_PS[i] = even_PS[i-1] + A[i]
        odd_PS[i] = odd_PS[i-1]
    else
        odd_PS[i] = odd_PS[i-1] + A[i]
        even_PS[i] = even_PS[i-1]

count = 0
for i in range(N):
    if i == 0:
        even_sum = odd_PS[N-1]
        odd_sum = even_PS[N-1] - even_PS[0]
    else:
        even_sum = even_PS[i-1] + odd_PS[N-1] - odd_PS[i]
        odd_sum = odd_PS[i-1] + even_PS[N-1] - even_PS[i]

    if even_sum == odd_sum:
        count += 1
return count
```

⑨ Pick from both sides!

Given an array of A of N elements, select B elements; some (all) from left / right to get maximum sum

```
"get prefix sum array"
if B == N
    return ps[N-1]
max-sum-possible = ps[N-1] - ps[N-B-1]
for i in range(B):
    prospect = ps[i] + ps[N-1] - ps[N-B+i]
    if prospect > max-sum-possible:
        max-sum-possible = prospect
return max-sum-possible
```

Lecture 5 : Carry Forward



find (a, g) pairs s.t. $i < j$

(10) ⟹ 4 pairs

Brute Force

TC : $O(N^2)$

SC : $O(1)$

```
cnt = 0
for i in range(len(l)):
    for j in range(i+1, len(l)):
        if l[i] == 'a' and l[j] == 'g'
            cnt += 1

return cnt
```

Optimized

carries forward the num of "g"'s adds to the count for subsequent "a"'s

```
ans = 0, count = 0
for i in range(len(l)-1, -1, -1):
    if a[i] == 'g':
        count += 1
    if a[i] == 'a':
        ans += count

return ans
```

TC : $O(N)$

SC : $O(1)$

(11) Given an array of N elements, count the no. of leaders in array if its strictly greater than all the element on its right

15 -1 7 2 5 4 2 3

⇒ 5

```
cnt = 1
max = A[-1]

for i in range(len(A)-2, -1, -1):
    if A[i] > max:
        cnt += 1
        max = A[i]

return cnt
```

(12) Find the closest minmax

length of smallest subarray which has at least one occurence of min & max element

```
ans = len(A)
min-index = -1
min-value = min(A)
max-value = max(A)

for i in range(len(A)-1, -1, -1):
    if A[i] == max-value and min-index != -1:
        ans = min(ans, min-index - i + 1)
    if A[i] == min-value
        min-index = i

max-index = -1

for i in range(len(A)-1, -1, -1):
    if A[i] == min-value and max-index != -1:
        ans = min(ans, max-index - i + 1)
    if A[i] == max-value:
        max-index = i

return ans
```

(13) Given N bulbs in an initial state, find minimum number of switches needed to turn ON all bulbs

a switch also changes state of all bulbs on the right [0 1 0 1] ⇒ [1 0 1 0] ⇒ [1 1 0 1]

```
start_idx = None
for i in range(len(A)):
    if A[i] == 0:
        start_idx = i
        break

if start_idx == None:
    return 0

min_switches = len(A) - start_idx

for i in range(start_idx, len(A)-1):
    if A[i] == A[i+1]:
        min_switches -= 1

return min_switches
```

(14) Given an integer element A.

Decide whether its possible to divide the array into one or more subarrays of even length s.t. the first and last element of all subarrays is even

[2, 4, 8, 6] ⟹ [2, 4] , [8, 6]

[2, 4, 8, 7, 6] ⟹ ✗

```
n = len(A)
if n % 2 == 1:
    return False
elif A[0] % 2 == 1 or A[n-1] % 2 == 1:
    return False
else:
    return True
```

Lecture 6 : Subarrays

How many subarrays are there? $\frac{N*(N+1)}{2}$

(15) Find sum of every subarray

Brute force

```
n = len(A)
for i in range(n):
    for j in range(i, n):
        sum = 0
        for k in range(i, j+1):
            sum += A[k]
        print(sum)
```

Using prefix sum

```
ps = [0]*n
ps[0] = A[0]

for i in range(1,n):
    ps[i] = ps[i-1] + A[i]


for i in range(n)
    for j in range(i,n):
        if i==0:
            print(ps[j]):

        else:
            print(ps[j]-ps[i-1])
```

(16) Find the sum of all subarrays

Brute Force

```
n = len(A)
res = 0
for i in range(n):
    for j in range(i, n):
        sum = 0
        for k in range(i, j+1):
            sum += A[k]
        print(sum)   ← sum of a subarray
        res += sum
print(res)   ← sum of all subarrays
```

$[a_0, a_1, a_2, a_3]$

$\Rightarrow$ $[a_0]$ $[a_1]$ $[a_2]$ $[a_3]$

$[a_0\ a_1]$ $[a_1\ a_2]$ $[a_2\ a_3]$

$[a_0\ a_1\ a_2]$ $[a_1\ a_2\ a_3]$

$[a_0\ a_1\ a_2\ a_3]$

(1)(4) (2)(3) (3)(2) (4)(1)

$4^* a_0 + 6^* a_1 + 6^* a_2 + 4^* a_3$

Optimized

```
n = len(A)

ans = 0
for i in range(n):
    ans += (i+1) * (n-i) * A[i]

return ans
```

17 Find the contiguous non-empty subarray within an array A of length N, with the largest sum

[1, 2, 3, 4, -10] ⇒ 10

```
n = len(A)
max_sum = min(A)
sum = 0
for i in range(n):
    sum += A[i]
    max_sum = max(max_sum, sum)
    if sum < 0:
        sum = 0
return max_sum
```

(18) Given a subarray of size N, find the subarray of size K with the least average return the first index of the subarray

```
n = len(A)

ps = [0]* n
ps[0] = A[0]

min_sum = min(A), temp_sum = 0
ans = 0
for i in range(1, n):
    ps[i] = ps[i-1] + A[i]
for j in range(0, n-k+1):
    if j==0:
        temp_sum = ps[k-1]
    else:
        temp_sum = ps[j+k-1] - ps[j-1]
    if temp_sum < min_sum:
        min_sum = temp_sum
        ans = j
return ans
```

Kadane Algorithm

(19) Find the maximum sum of contiguous array A where sum < B

```
n= len(A)
if B > min(A):
    max = min(C)
    for i in range(n):
        sum = 0
        for j in range(i,n):
            sum += A[j]
            if (sum <= B) and (sum > max):
                max = sum
        return max
else :
    return 0
```

(20) Given an array A of N elements, containing only 0's and 1's. Return all the indices which can act as center of alternating subarrays of length $2*B+1$

A = [1, 0, 1, 0, 1]

B = 1

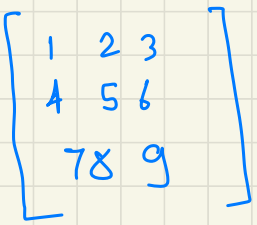
length = $2*1+1 = 3$

⇒ indexes [1, 2, 3]

```
n = len(A)
if B == 0:
    return list(range(0, n))

subarray_len = 2**B + 1
if subarray_len % 2 == 0:
    return []

ans = []
prev = A[0]
start = 0

for i in range(1, n):
    if start > n - subarray_len:
        break

    if A[i] == prev:
        start = i

    if (i - start + 1 == subarray_len):
        ans.append(start + (subarray_len//2))
        start += 1
    prev = A[i]
return ans
```

Lecture 7 : 2D Arrays

(21) Given N * N matrix, return an array of its anti-diagonals



⇒

1
2 4
3 5 7
6 8
9

Explanation on Pg. 31 →

Code on Pg. 32 →

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

j = 0 to 2

i = 0 j = 0 Bj = 0

while (j >= 0):

B[0].append(1)

i → 1

j = -1 ✗

Bj → 1

j = 1 i = 0 Bj = 1

B[1].append(A[0, 1])

(2)

i → 1 j = 0

B[1].append(4)

j = 2 i = 0 Bj = 2

B[2].append(3)

(5)

(7)

$$\begin{bmatrix} 1 & & \\ 2 & 4 & \\ 3 & 5 & 7 \\ 6 & 8 & \\ 9 & & \end{bmatrix}$$

i = 1, 2

j = 2 i = 1 Bj = 3

while (i < 3):

B[3].append(6)

i → 2 (8)

j = 1

Bj = 4

i = 2

j = 2 i = 2 Bj = 4

B[4].append(9)

```
    # @return a list of list of integers
    def diagonal(self, A):
        sizeOfMatrix = len(A)
        B = [[] for _ in range(2* sizeOfMatrix -1)]
        # print(B)
        Bj =0
        for j in range(sizeOfMatrix):
            i = 0
            while(j >= 0 ):
                # print(i,j)
                B[Bj].append(A[i][j])
                i+=1
                j-=1
            Bj+=1
        # print(B)
        for i in range(1,sizeOfMatrix):
            j = sizeOfMatrix-1
            while(i < sizeOfMatrix ):
                # print(i,j)
                B[Bj].append(A[i][j])
                i+=1
                j-=1
            Bj+=1
        for Bi in range(2 *sizeOfMatrix -1):
            while (len(B[Bi])< sizeOfMatrix):
                B[Bi].append(0)
        # print(B)
        return B
```

(Upper Triangle)

(Lower Triangle)

Lecture 8 : Interview Problems

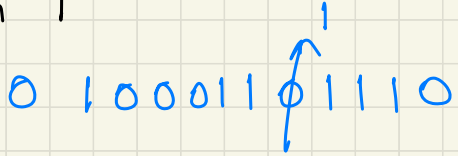
(22) Given matrix, print boundary in clockwise direct^{n}

```
i = 0 , j = 0
while N > 1 :
    for k in range (0, N-1):
        print ( A[i][j] )
        j += 1

    for k in range (0, N-1):
        print ( A[i][j] )
        i += 1

    for k in range (0, N-1):
        print ( A[i][j] )
        j -= 1

    for k in range (0, N-1):
        print (A[i][j])
        i -= 1
    i += 1 , j += 1
    N = N - 2
if N % 2 == 1
    print A[N//2][N//2]
```

1	14	15	19
8	9	10	20
7	12	11	21
6	24	23	22

N = 4

(23) Make maximum no. of consecutive 1's by swapping a 0 with 1




```
n = len(A)
left = [0] * n
right = [0] * n
ans = 0

# count 1's
for i in range(n):
    if A[i] == "1":
        count1 += 1
if count1 == n:
    return n

# count 1's on left
if A[0] == "0":
    left[0] = 0
else:
    left[0] = 1

for i in range(1, n):
    if A[i] == "0":
        left[i] = 0
    else:
        left[i] = left[i-1] + 1
```

```
# count 1's on the right
right[n-1] = int(A[n-1])
for j in range(n-2, -1, -1)
    if A[j] == "0":
        right[j] = 0
    else:
        right[j] = right[j+1] + 1

L = 0
R = 0

for k in range(n):
    if A[k] == "0":
        if k == 0:
            L = 0
        else:
            L = left[k-1]

        if k == n-1:
            R = 0
        else:
            R = right[k+1]

        total = L + R + 1

        if total > count1:      # no 1's to swap
            total = count1

        ans = max(ans, total)
        total = 0
return ans
```

(24) Given an integer A. Generate a square matrix 1 to A^2 elements

I/P = 4 ⇒

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

Same logic as (22)

(25) Return maximum size subarray of A w/ all non-negative elements. If there are multiple subarrays, choose the one with lowest starting index.

```
N = len(A)
maximum = 0
start_index = 0
count = 0

for k in range(N):
    if A[k] > 0:
        if count == 0:
            i = k  # starting pt. of tve subarray
        count += 1

    if A[k] < 0 or (k == N-1):
        if count > maximum:
            start_index = i  # start pt. of answer
            maximum = count  # logs length
        count = 0

return A[start_index : start_index + maximum]
```

[5, 6, -1, 7, 8]
 0 1 2 3 4

Lecture 9 : Bit Manipulations - 1

Converting any system to decimal

① $(100110)_2$

$= \underset{2^5}{1}\ \underset{2^4}{0}\ \underset{2^3}{0}\ \underset{2^2}{1}\ \underset{2^1}{1}\ \underset{2^0}{0}$

$= (37)_{10}$

② $(13)_8 = 8^1 \times 1 + 8^0 \times 3$

$= (11)_{10}$

Converting from decimal to binary

① $(1037)_{10} = (\underset{10\ 9\ 8\ 7\ 6\ 5\ 4\ 3\ 2\ 1\ 0}{10000001101})_2$

1024 + 8 + 4 + 1

OR

dividing by 2

	1037
1	568
0	184
⋮	⋮

end reverse the remainders

^ - XOR

& - AND

| - OR

$a \wedge a = 0$

$a \wedge 0 = a$

$a \wedge 1 \rightarrow a - 1$ odd

$a \wedge 1 \rightarrow a + 1$ even

$a \mid 1 \rightarrow a$

$a \mid 1 \rightarrow a + 1$

$a \,\&\, 1 \rightarrow 0$

$a \,\&\, 1 \rightarrow 1$

$(11)_{10} = (1011)_2$

$(22)_{10} = (10110)_2$

$(44)_{10} = (101100)_2$

$11 << 1 \quad == \quad 11 * 2$

left shift operator

$11 >> 1$ right shift operator

$== \quad 11 // 2$

Check if 6^{th} bit is ON/OFF

$k = 6$

$x \;\&\; (1 << (k-1))$

$x >> (k-1) \;\&\; 1$

(26) Count the number of bits which are ON

Brute Force

```
cnt = 0
while x != 0:
    if x & 1:
        cnt += 1
    x = x >> 1
return cnt
```

O(logN)

Trick

```
cnt = 0
while x:
    x = x & (x-1)
    cnt += 1
return cnt
```

(27) Given an array, find integer that occurs only once. (All others appear twice)

```
n = len(A)
res = A[0]
for i in range(1, n):
    res = res ^ A[i]
return res
```

$a \hat{} a = 0$

Lecture 10: Bit Manipulation - 2

Represent -3 in binary

+3 : 0 0 0 0 0 0 1 1

2's complement 1 1 1 1 1 1 0 0

\+ 1

1 1 1 1 1 1 0 1 ⇒ -3

(28) Help from Sam.

If Alex solves a question → score doubles
takes help → score +1

Initial score : 0 Target : 5

0
help 1
solves 2
solves 4
help 5

Ans. (2) help taken

```
if A == 0:
    return 0
cnt = 0
while A:
    cnt += 1
    A = A & (A-1)
return cnt
```

In hindsight of the story, this problem basically is checking the ON bits

(29) Given an array of integers A. Two integers appear only once & all others twice. Find them.

[1 2 3 1 2 4] ⇒ [3, 4]

```
xor = 0
answer = [0, 0]

for i in range(n):
    xor ^= A[i]

msb = 0
while xor:
    xor = xor & (xor-1)
    msb += 1

set1 = []
set2 = []

for e in A:
    if e >> (msb-1) & 1:
        set1.append(e)
    else:
        set2.append(e)

for x in set1:
    ans[0] ^= x
for y in set2:
    ans[1] ^= y
return sorted(ans)
```

xor == 7 [0 1 1]

finding most significant bit

even no. set
odd no. set

(30) Maximum Satisfaction

Given an array A on N integers. A[i] represents quality of the i^{th} fruit. Pick four fruits s.t. satisfaction value will maximum. → a & b & c & d

(& ↑ bitwise AND)

Find maximum satisfaction value.

This means we have to check which MSB is ON using the integer's 32 bit binary rep"

```
ans = 0
for i in range(32, -1, -1):
    temp = ans | (1<<i)
    if check(temp, A) == 1:
        ans = temp
return ans
```

```
def check(x, A):
    cnt = 0
    for a in A:
        if (a & x) == x:
            cnt += 1
        if cnt == 4
            return 1
    return 0
```

(31) Excel column number

"AB" ⇒ 28 ((1)(26¹) + (2)(26⁰))

"YZ" ⇒ 25(26) + (26)(1)

```
N = len(A)
ans = 0
for i in range(N-1, -1, -1):
    ans += (ord(A[i]) - 64) * (26 ** (N-1-i))

return ans
```

Lecture 11 - Modular Arithmetic

① Linearity in modular addition

$$(a+b)\%M = [(a\%M) + (b\%M)]\%M$$

② Linearity in modular multiplication

$$(a*b)\%M = [(a\%M) * (b\%M)]\%M$$

32 Power function using Recursion

```
def power(a, b):
    if b == 1:
        return a
    x = power(a, b//2)
    if b % 2 == 0:
        return x*x
    else:
        return x*x*a
```

(33) Leap Year or not!

```
if A % 400 == 0 and A % 100 == 0 :
    return 1

elif A % 100 != 0 and A % 4 == 0 :
    return 1

else :
    return 0
```

(34) Find LCM

(2,3) ⟹ 6

(9,6) ⟹ 18

```
gNum = max(A, B)
lcm = gNum
while True :
        if A % gNum == 0 and B % gNum == 0 :
                lcm = gNum
                break
        gNum += 1
return lcm
```

35 Find GCD

(24, 32) ⟹ 8

Pseudo code :

```
lNum = min(24, 32)
gcd = lNum
while gcd >= 1 :
        if A % lNum == 0 and B % lNum == 0 :
                gcd = lNum
                break
        lNum -= 1
return gcd
```

Lecture 12: Sorting

l.sort()

$O(N \log N)$

l_new = sorted(l)

Types of sort:

1. Selection sort
2. Bubble sort
3. Insertion sort
4. Merge sort
5. Quick sort
6. Heap sort
7. Counting sort
8. Radix sort
9. Bucket sort

(38) Given an array, remove every element from array one by one; cost to remove is the sum of all elements in it. Find minimum cost to remove all elements.

```
A.sort(reverse=True)
ans = 0
for i in range(len(A)):
    ans += A[i] * (i+1)

return ans
```

(37) Find the minimum difference of an array

min. value is $\min(|A[i]-A[j]|)$

where i, j are distinct.

Brute Force

```
ans = INF

for i in range(n):
    for j in range(i+1, n):
        ans = min(ans, abs(A[i]-A[j]))

return ans
```

$O(N^2)$

[1, -5, 3, 5, -10, 4]

⇒ 1

[-10 -5 1 3 4 5]

5 6 2 1 1

Optimized

```
n = len(A)
A.sort()
ans = INF
for i in range(n-1):
    ans = min(ans, A[i+1] - A[i])
return ans
```

$O(N \log N)$

(38) Given an array of N distinct elements, find count of Noble elements

[1 -5 3 5 -10 4]

2 1 3 5 0 4

⇒ 3

Brute Force

→ go to every no.
→ counts less than no. == no. itself

Optimized

```
l.sort()
cnt = 0
for i in range(len(l)):
    if A[i] == i:
        cnt += 1
return cnt
```

(39) Noble elements when elements are not distinct.

[-10 1 1 3 100]

0 1 1 3 4

```
l.sort()
check = [0]*n
for i in range(n):
    if l[i] = l[i-1]:
        check[i] = check[i-1]
    else
        check[i] = i
cnt = 0
for i in range(n):
    if l[i] == i:
        cnt += 1
return cnt
```

☆ Comparators and Key

(40) Sorting the array based on no. of factors

```
def count-factors(x):
    cnt = 0
    for i in range(1, x+1):
        if x % i == 0:
            cnt += 1
    return cnt
```

```
l = [12, 9, 1, 6, 7]
l.sort(key = count-factors)
```

→ [1, 7, 9, 6, 12]

1 2 3 4 6

```
def cmp(a,b):
    cnta = count-factors(a)
    cntb = count-factors(b)
    if cnta == cntb:
        return 0
    if cnta > cntb:
        return 1
    if cnta < cntb:
        return -1
```

-1 means it should be on the left

i/p [1, 7, 6, 9, 12]

```
from functools import cmp_to_key
l.sort(key = cmp_to_key(cmp))
```

→ [1, 7, 9, 6, 12]

1 will be compared to 7 first 1 < 7 returns -1

so, 1 will be before 7

1 ⟺ 6 1 before 6

& likewise for all elements to get final sorted list

comparators can be used for complex comparisons

Another example:

```
def descCmp(a,b):
    if a > b:
        return -1
    elif a < b:
        return 1
    return 0
```

-1 here means a should be on the left (over case a greater value on left)

```
l = [1, 2, 9, 12, 6]
l.sort(key = cmp_to_key(descCmp))
```

→ [12, 6, 9, 7, []]

(41) Given an array A of N integers. Arrange them in such a way that they form the largest number

[2, 3, 9, 0]

↳ 9320

```
from functools import cmp_to_key
def compare(a, b):
    if a+b > b+a:
        return -1
    else:
        return 1
A.sort(key = cmp_to_key(compare))
if A[0] == "0":
    return 0
return "".join(A)
```

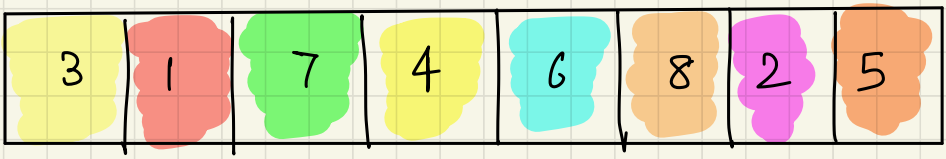
☆ Link to the blog :

https://towardsdatascience.com/5-sorting-algorithms-in-python-c7ece9df5dd6

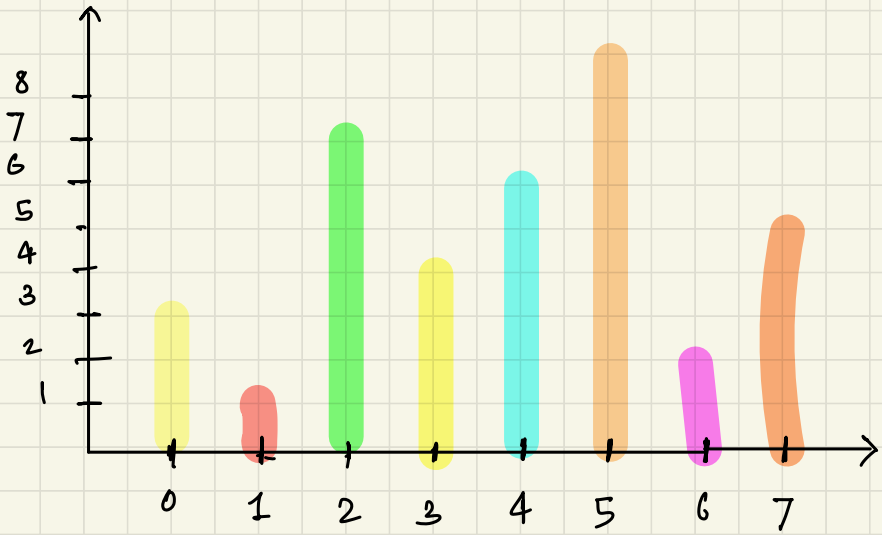
☆ Video for visualizing sorting algorithms

https://www.youtube.com/'watch?'v=kPRA0W1kECg'

Array Visualization



N = 8




```
def swap (arr, a, b):
    temp = arr[a]
    arr[a] = arr[b]
    arr[b] = temp
```

swaps elements at index "a" & "b"

1. Selection Sort

- in-place sorting algorithm, meaning the sorted items use the same storage as original elements

Ex. $[\underset{0}{1}, \underset{1}{4}, \underset{2}{-6}, \underset{3}{2}]$

i = 0 to 3 j = i+1 to 3

i = 0,

curr-min = 1
min-idx = 0

j : 1 ×

j : 2 curr-min > arr[2]
curr-min = -6
min-idx = 2

j : 3 ×
swap (1, -6) ⇒ [-6, 4, 1, 2]
(1 at index 0, -6 at index 2)

i = 1
& so on

```python
def selection_sort(self, unsorted, n):

    # iterate over array
    for i in range(0, n):

        # initialise with first value
        current_min = unsorted[i]

        # min_index initialiser
        min_index = i

        # iterate over remaining unsorted items
        for j in range(i, n):

            # check if jth value is less than current min
            if unsorted[j] < current_min:

                # update minimum value and index
                current_min = unsorted[j]
                min_index = j

        # swap ith and jth values
        swap(unsorted, i, min_index)
```

TC : $O(N^2)$

SC : $O(1)$

2. Bubble Sort

- bubble or sinking sort repeatedly passes over the list, comparing adjacent elements.
- Items are swapped depending on sorting condition

Ex. [64, 34, 25] (0, 1, 2)

i = 0 to 1 swapped = False

i = 0 j : 0 to 1

j = 0 64 > 34 → swap → [34, 64, 25] swapped = True

j = 1 64 > 25 → swap → [34, 25, 64] (n-1-1) 61

i = 1 j : 0 to 0

i = 1 j = 0 34 > 25 → swap → [25, 34, 64]

- larger values are bubbling to the end as program executes.

```python
def bubble_sort(self, unsorted, n):
    """ bubble sort algorithm """

    # iterate over unsorted array up until second last element
    for i in range(0, n - 1):

        # swapped conditions monitors for finalised list
        swapped = False

        # iterate over remaining unsorted items
        for j in range(0, n - 1 - i):

            # compare adjacent elements
            if unsorted[j].value > unsorted[j + 1].value:

                # swap elements
                swap(unsorted, j, j + 1)
                swapped = True

        # no swaps have occured so terminate
        if not swapped:
            break
```

TC:

Best case: when array is sorted

"n-1" comparisons $O(N)$

Worst case: sorting a descending array

$\frac{n * (n-1)}{2}$ comparisons $O(N^2)$

Average case: $O(N^2)$

SC: $O(1)$

3. Insertion sort

- builds the final array one item at a time.
- each object encountered on the outer loop is put between current closest minimum and maximum elements.

Ex. [9, 6, 7, 2, 5, 8]
0 1 2 3 4 5

i = 0 ×

i = 1 val = 6

hole = 1
arr[0] > val → arr[1] = 9

hole = 0
arr[0] = 6 [6 9 7 2 5 8]

i = 2 val = 7
hole = 2
arr[1] > 7 → arr[2] = 9
hole = 1
arr[0] > 7 ×
arr[1] = 7 [6 7 9 2 5 8]

& so on

```python
def insertion_sort(unsorted, n):
    """ insertion sort algorithm """

    # iterate over unsorted array
    for i in range(1, n):

        # get element value
        val = unsorted[i].value

        # insertion "hole" is at index i
        hole = i

        # loop backwards until a value greater than current value is found
        while hole > 0 and unsorted[hole - 1].value > val:

            # swap elements towards correct position
            unsorted[hole].value = unsorted[hole - 1].value

            # move backwards
            hole -= 1

        # insert value into correct position
        unsorted[hole].value = val
```

TC : $O(N^2)$

SC : $O(1)$

Lecture 13: Strings

ASCII

'A' - 65	'a' - 97	'0' - 48
'B' - 66	'b' - 98	'9' - 57
⋮		
'Z' - 90	'z' - 122	

'A' → 65 → $(0\,1\,0\,0\,0\,0\,0\,1)_2$

'a' → 97 → $(0\,1\,1\,0\,0\,0\,0\,1)_2$

flip from lower to upper & vice-versa by add/subtract $2^5 = 32$

ord ('a') = 97

chr (97) = 'a'

(42) Given a string flip the case of chars.

```
def flip(A):
    l = list(A)
    for ix in range(len(l)):
        l[ix] = chr(ord(l[ix])^32)
    return "".join(l)
```

Follow-up

If any character is in uppercase, convert it to lowercase

```
l = list(A)
ans = ""
for c in l:
    if c > 'A' or c < 'Z':
        c = chr(ord(c)^32)
        ans += c
    else:
        ans += c
return ans
```

(43) Given the string in lowercase, sort it.

[a g d c a b]

⇒ [a a b c d g]

Brute Force

str → list

list.sort()

"".join(list)

O(N log N)

Trick

```
freq = [0] * 26
ans = ""
for x in A :
    freq[ ord(x) - ord('a') ] += 1

for i in range(26):
    for j in range ( freq[i] ):
        ans += chr(97+i)

return ans
```

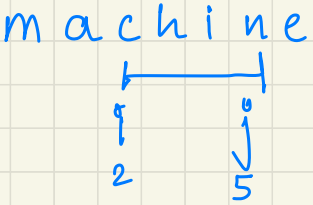
a	b	c	d	e	f	g	h		
0	1	2	3	4	5	6	7	8	9	10

freq [2 1 1 1 0 0 1 0 0 0 ...]

TC: const. time

→ [a a b c d g]

(44) Given a substring, reverse it



① convert to list

② while i < j:

```
l[i], l[j] = l[j], l[i]
i += 1
j -= 1
```

③ return "".join(list)

(45) Reverse the words in a sentence

"I am a slave"

1. sentence.split()
2. Reverse the list
3. join()

OR

```
l = s.split()          i = 0   j = len(l)
while i < j:
    l[i], l[j] = l[j], l[i]
    i += 1
    j -= 1
return "".join(l)
```

(46) Given 2 strings, find the length of longest common prefix string

"professor"
"project"
⇒ 3

```
ans = 0
for i in range(min(len(a), len(b))):
    if a[i] == b[i]
        ans += 1
    else:
        break
return ans
```

(47) Find longest common prefix string in N strings

[professor
project
pronunciation
pronouns]

```
ans = 0
n = len(l)
min-word-len = len(l[0])
for word in l:
    if len(word) < min-word-len:
        min-word-len = len(word)

for i in range(min_word-len):
    for j in range(n-1):
        if l[j][i] == l[j+1][i]:
            ans += 1
        else:
            break
return ans
```

Alternative soln :

```
def longestCommonPrefix(self, A):
    size = len(A)

    # if size is 0, return empty string
    if (size == 0):
        return ""

    if (size == 1):
        return A[0]

    # sort the array of strings
    A.sort()

    # find the minimum length from
    # first and last string
    end = min(len(A[0]), len(A[size - 1]))

    # find the common prefix between
    # the first and last string
    i = 0
    while (i < end and A[0][i] == A[size - 1][i]):
        i += 1

    pre = A[0][0: i]
    return pre
```

(48) Check if a string is palindrome

"nayan" ↔ "nayan"

Brute Force

```
if s == s[::-1]:
    return 1
return 0
```

O(N)

Optimized

```
i = 0 , j = n-1
while i < j:
    if a[i] != a[j]
        return False
    i += 1
    j -= 1
return True
```

(49) Find the length of longest palindrome substring

a a b c c b a d

check if the substring is a palindrome

```
def helper(l, r):
    while (l >= 0 and r < len(A)) and
          (A[l] == A[r]):
        l -= 1
        r += 1
    return (A[l+1 : r])
res = ""
for i in range(len(A)):
    odd-sub = helper(i, i)
    even-sub = helper(i, i+1)
    if len(odd-sub) > res:
        res = odd-sub
    if len(even-sub) > res:
        res = even-sub
return res
```

abba

i = 0 to 3

c = 0

helper(0, 0) → a

res = "a"

helper(0, 1) → a

res = "a"

c = 1

helper(1, 1) → b

res = "b"

helper(1, 2) → l = 0 r = 3 l = -1 r = 4

res = "bb"

[0 : 4]

i = 2

helper(2, 2) → "b"

helper

(50) Given a string, find no. of vowel substrings

"ABEC" ⇒ "A"
"AB"
"ABE"
"ABEC"
"E"
"EC"
cnt = 6

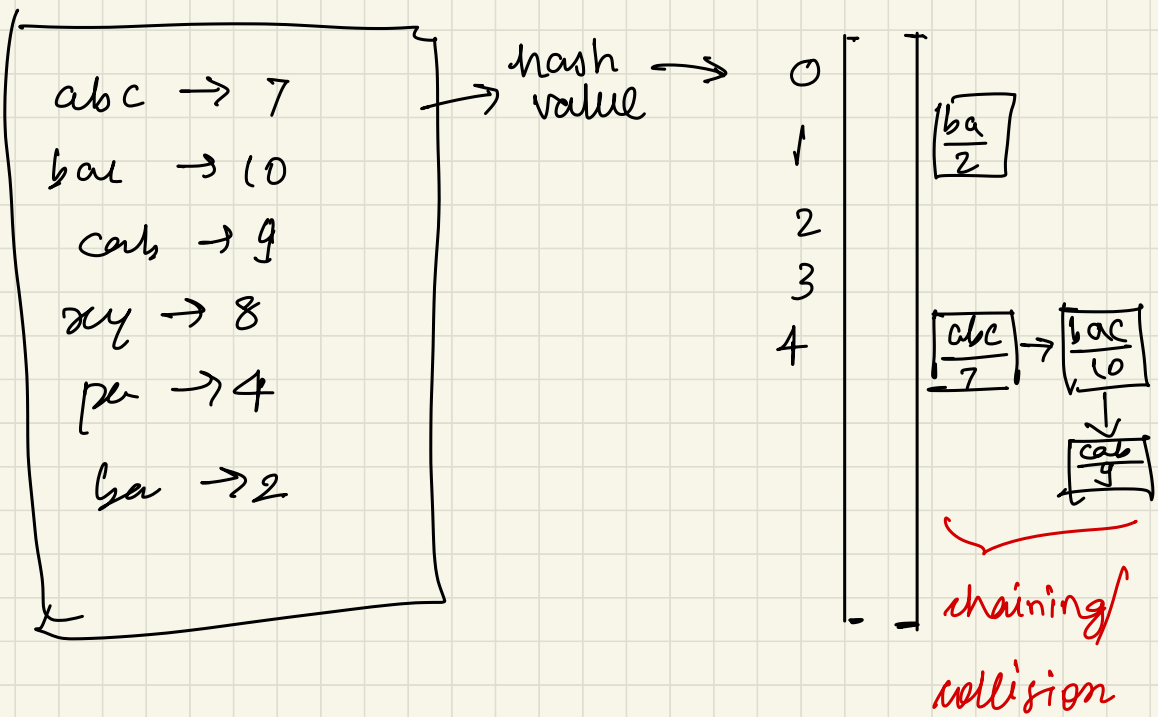
```
vowels = [        ]
cnt = 0
for i in range(len(A)):
    if A[i] in vowels:
        cnt += len(A) - i
return count
```

2, 10003

Lecture 14 : Hashing - 1

Hash maps : maps any keys to values

→ space-efficient solution to store K-V pair
→ fast access of data
→ check if a key is present
→ Insert K-V pairs
→ Update K-V pairs



Hash map should be able to

- map the values
- indexable based on keys
- get in O(1) time

(51) Given an array, you have Q queries. Return count of elements in array for each query

[1 1 10^9 0 10^9 0]

Q = [1 → [ ] int = 2

0 →

10^9]

Brute force

$O(N * Q)$

Using dictionary

```
freq = {}
for val in arr:
    if val in freq:
        freq[val] += 1
    else:
        freq[val] = 1
for query in Q:
    if query in freq:
        return freq[query]
    else:
        return 0
```

(52) Given an array, check if there exists a subarray with sum = 0

Brute Force

```
n = len(arr)
for i in range(n):
    for j in range(i, n):
        sum = 0
        for k in range(i, j+1):
            sum += arr[k]
        if sum == 0:
            return True
return False
```

$O(N^3)$

Using Prefix sum

```
n = len(arr)        <- calculate prefix sum
for i in range(n):
    for j in range(i, n):
        if ps[j] - ps[i-1] == 0
            return True
return False
```

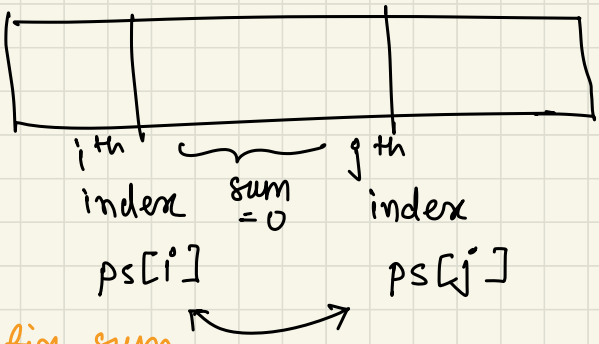
TC: $O(N^2)$ SC: $O(N)$

Using carry forward

```
n = len(arr)
for i in range(n):
    sum = 0
    for j in range(i, n):
        sum += arr[j]
        if sum == 0:
            return True
return False
```

TC: $O(N^2)$ SC: $O(1)$

Trick



Check if a prefix sum is repeated or not

```
sum = 0
freq-ps = {}
for i in range(n):
    sum += arr[i]
    if sum in freq-ps:
        return True
    freq-ps[sum] = 1
return False
```

TC: O(N)

SC: O(N)

easy code →

```
N = len(A)
ps = [0]*N
ps[0] = A[0]
for i in range(1,N):
    ps[i] = ps[i-1] + A[i]
if (0 in ps) or
   (len(set(ps)) != N):
    return 1
else:
    return 0
```

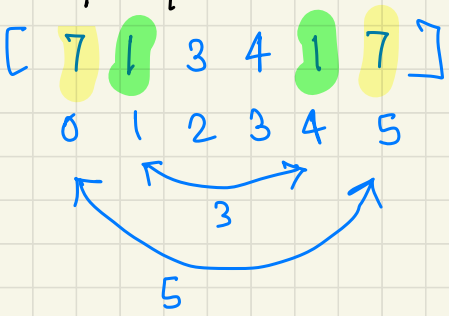
(53) Given an array A, find the first repeating element.

[10 5 3 4 3 5 6]

return the first occurence of repeating element otherwise return -1

```
N = len(A)
freq = {}
for i in A:
    if i in range(N):
        if A[i] in freq:
            freq[A[i]] += 1
        else:
            freq[A[i]] = 1
for e in A:
    if freq[e] > 1
        return e
return -1
```

(54) Given an array of N elements, call a pair of distinct indices in that array special if elements at those indices are equal. Find a special pair s.t. the distance b/w pair is minimum




```
min-dist = sys.maxsize
freq = {}

idx = 0
for i in A:

    if i not in freq.keys():
        freq[i] = idx

    else:
        min-dist = min(min-dist, idx - freq[i])
        freq[i] = idx

    idx += 1

if min-dist == sys.maxsize:
    return -1

else:
    return min-dist
```

(55) Largest Continuous Sequence Zero Sum

[1 2 -2 4 -4] ⟹ [2 -2 4 -4]

```
n = len(A)
mydict = {}
max = -sys.maxsize -1

ps = [0]*n   ps[0] = A[0]
for i in range(n):
    ps[i] = ps[i-1] + A[i]


for i in range(n):

    if (ps[i] in mydict) or (ps[i]==0):

        if (ps[i]==0):
            dist = i+1
            if dist > max:
                max = dist
                start = -1
                end = i
        else:
            dist = i - mydict[ps[i]]
            if dist > max:
                max = dist
                start= mydict[ps[i]]
                end =i
    else:
        mydict[ps[i]]=i
if max != -sys.maxsize-1:
    return A[start+1:end+1]
return []
```

(56) Given a string, check if the letters can be rearranged to create a palindrome

```
freq = {}
oddc = 0
s = set()

for i in range(n):
    if A[i] in freq:
        freq[A[i]] += 1
    else:
        freq[A[i]] = 1

for i in range(n):
    if freq[A[i]] % 2 == 0:
        continue
    elif A[i] in s:
        continue
    else:
        s.add(A[i])
        oddc += 1
if oddc > 1:
    return 0
else:
    return 1
```

(57) Given an array A, find if its COLORFUL or not.

3245 → 3, 2, 4, 5, 32, 24, 45, 324, 245

all of product of digits are different

```
A = str(A)
n = len(A)

check = set()

for i in range(n):
    prod = 1

    for j in range(i, n):
        prod = prod * int(A[j])

        if prod in check:
            return 0
        else:
            check.add(prod)
return 1
```

Lecture 15 - Hashing II

(58) Given N array elements, count distinct elements in every window of size K

N [1 2 1 3 2 1 2 1 1 2 3] K=4

⇒ [3, 3, 3, 3, 2, 2, 2, 3]

N-K+1

Brute Force

```
ans = [ ]
for i in range(n-k+1):
    l = [ ]
    for j in range(k)
        if A[j+i] not in l:
            l.append(A[j+i])
    ans.append(len(l))
```

TC: O(N)

Using frequency mapping

```
for i in range(n-k+1):
    freq = {}
    for j in range(k):
        if A[j+i] not in freq:
            freq[A[j+i]] = 1
        else:
            freq[A[j+i]] += 1
    ans.append(len(freq))
```

TC : O(N)

Optimize space complexity in frequency mapping

```
freq = {}
n = len(arr)

# 0, 1, 2, ... k-1
for i in range(k):
    if arr[i] in freq:
        freq[arr[i]] += 1
    else:
        freq[arr[i]] = 1


ans = []
ans.append(len(freq))

# a1 .... ak ; remove a0 ; add ak ; so on
for end in range(k, n):
    if arr[end] in freq:
        freq[arr[end]] += 1
    else:
        freq[arr[end]] = 1

    element_to_remove = arr[end - k]
    freq[element_to_remove] -= 1

    if freq[element_to_remove] == 0:

        del freq[element_to_remove]
    ans.append(len(freq))
return ans
```

59 Given N array elements, check if there exists a pair (i, j) s.t. arr[i] + arr[j] == k

[1 2 5, 7, 3, 2, 1]

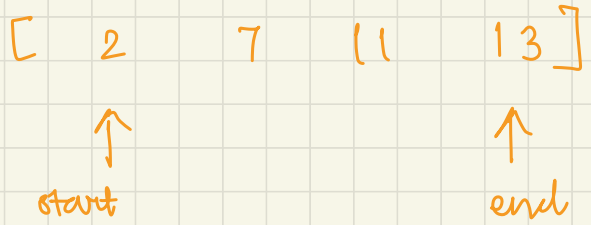
Brute Force

K = 12

```
for i in range(n)
    for j in range(i+1, n):
        if A[i] + A[j] == k:
            return True
return False
```

TC: $O(N^2)$

Using two-pointer approach ON THE SORTED ARRAY



(1) K = 9

start + end (15) > K ✗

end -= 1

start + end (13) > K → end -= 1

start + end (9) == K ✓

(2) K = 18

start + end (15) < K → start += 1

start + end (20) > K → end -= 1

start + end (18) == K ✓

TC : O(N log N)

Optimized

```
freq = {}
for i in range(n):
    if k - arr[i] in freq:
        return True
    else:
        if arr[i] not in freq:
            freq[arr[i]] = 1
return False
```

(60) given an array of elements A and a non-negative integer B. Check if there exists i and j s.t. $A[i] - A[j] = B$ $i \neq j$

```
hashmap = {}
for i in A:
    if A[i] in hashmap:
        hashmap[A[i]] += 1
    else:
        hashmap[A[i]] = 1

if len(A) < 2
    return 0

for i in range(N):
    a = A[i]
    b = A[i] + B

    if a != b:
        if b in hashmap:
            return 1
    else:
        if hashmap[b] > 1:
            return 1

return 0
```

case when B = 0 so num - num = 0

num's freq should be > 1

(61) Given N elements of an array, find the longest sequence which can be rearranged to form a sequence of increasing consecutive numbers

[100, 4, 200, 1, 2, 3] ⇒ [1, 2, 3, 4]

Brute Force

```
l.sort()
i = 0
while i < N:
    count = 1
    prev = arr[i]
    i += 1
    while arr[i] == prev + 1:
        prev += 1
        i += 1
        count += 1
    ans = max(ans, count)
```

TC: O(NlogN)

Using dict

0 1 2 3 4 5 6 7 8

[98 99 7 6 5 4 32 100 97]

{ 2 : 1
3 : 1
4 : 1
5 : 1
6 : 1
7 : 1
98 : 1
: }

```
freq = {}
for i in range(n):
    if A[i] in freq:
        freq[A[i]] += 1
    else:
        freq[A[i]] = 1
ans = 0
for i in range(n):
    if A[i]-1 in freq:
        continue
    else:
        val = arr[i]
        count = 0
        while val in freq:
            count += 1
            val += 1
        ans = max(ans, count)
```

TC: O(N)

(62) Given an array of integers A and target B. Find out the first continuous subarray which adds to B, otherwise return -1

[1, 2, 3, 4, 5]

ps [1 3 6 10 15] ⟹ [2, 3]

B = 5

Solution 1:

freq { 1 : 0

3 : 1

i = 2

6 - 5 = 1 in freq

A[0+1 : 2+1]

A[1 : 3]

[1 2 3 4 5]

```
N = len(A)
ps = [0]*N
ps[0] = A[0]

for i in range(N):
    ps[i] = ps[i-1] + A[i]

freq = {}
for i in range(N):
    if ps[i]-B in freq:
        return A[freq[ps[i]-B]+1 : i+1]

    elif ps[i]-B == 0:
        return A[: i+1]
    elif ps[i] not in freq:
        freq[ps[i]] = i

return [-1]
```

Solution 2 :

```
start_index = 0
end_index = 0
curr_sum = 0

while (start_index < N) and (end_index < N):

        if start_index > end_index:
            end_index = start_index
            curr_sum = A[end_index]

        if curr_sum < B:
            curr_sum += A[end_index]
            end_index += 1

        if curr_sum > B:
            curr_sum -= A[start_index]
            start_index += 1

        if B == curr_sum:
            return A[start_index : end_index]

return [-1]
```

(63) Given an array of N strings A. B contains a string of size 26 denoting the order. Return 1 iff given words are sorted lexicographically in order defined in B.

A = ["hello", "scaler", "interview"]

B = "adbhqrstijklmvw...."

h < s < i

same order as in A ⟹ return 1

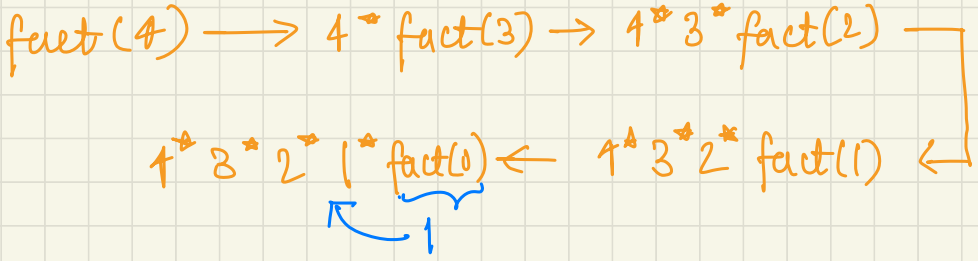
```
hb = {}
for i in range(len(B)):
    hb[ B[i] ] = i

for i in range(1, len(A))
    prev = A[i-1]
    curr = A[i]

    j = 0
    flag = False
    while j < len(prev) and j < len(curr):
        if hb[prev[j]] < hb[curr[j]]:
            flag = True
            break
        elif hb[prev[j]] > hb[curr[j]]:
            return 0
        else:
            j += 1
    if not flag and len(prev) > len(curr):
        return 0
return 1
```

Lecture 16 : Recursion - 1

(64) Factorial

```
def fact(n):
    if n==0:
        return 1
    return n*fact(n-1)
```

fact(4) = 24



(65) Fiboracci

ix:	0	1	2	3	4	5
v:	0	1	1	2	3	5 ...

```
def fib(n):
    if n==0:
        return 0
    if n==1
        return 1
    return fib(n-1) + fib(n-2)
```

fib(3) ⟶ fib(2) + fib(1) = 2
(fib(1) = 1)

fib(2) ↙
fib(1) + fib(0)
1 0

(66) Check if a string is PALINDROME or not (using Recursion)

Brute Force :

```
def checkPalindrome(s):
    if len(s) <= 1:
        return True

    if s[0] == s[-1] and solve(s[1:-1]):
        return True

    return False
```

Optimized

```
def isPalindrome(w, s, e):
    if s >= e:
        return True
    if w[s] == w[e]:
        return isPalindrome(w, s+1, e-1)
    return False
def solve(word):
    return isPalindrome(word, 0, len(word)-1)
```

(67) Print reverse string using Recursion

```
import sys
sys.setrecursionlimit(10**6)

def printrevstring(ip_string, s, e):
    if s>=e:
        return ip_string[s]
    print(ip_string[e], end="")
    return printrevstring(ip_string, s, e-1)

def main():
    ip = str(input())
    print(printrevstring(ip, 0, len(ip)-1))
```

Lecture 17 : Recursion - 2

(68) Sum of digits using Recursion

```
def sumDigits(num):
    if num <= 0:
        return 0
    dig = num % 10
    return dig + sumDigits(num//10)
```

NOTE :

1. no other variable was declared
2. no. of function calls : len(str(x)) or $\log_{10} x + 1$

SC : O(log N)

3. How to calculate time complexity

For a number N,

$T(N) = T(N//10) + O(1)$

time taken for whole fn to execute

$T(N//100) + 2 * O(1)$

$\vdots$

$T(N) = T\left(\frac{N}{10^k}\right) + k * O(1)$

$\frac{N}{10^k} = 0$

What will T(0) return (in O(1))

when? $\Rightarrow$

$10^k > N$

$k > \log_{10} N$

TC: $\log_{10} N * O(1)$

TC: $O(\log_{10} N)$

(69) Power function

$2^{10} \rightarrow 2^9 \times 2 \rightarrow 2^8 \times 2 \times 2$ .....

```
def power(a, n):
    if n==0:
        return 1
    return a * power(a, n-1)
```

Finding T.C.

assume time taken as $T(a, n)$

$$T(a,n) = T(a, n-1) + O(1)$$
$$= T(a, n-2) + 2 * O(1)$$
$$\vdots$$
$$= T(a, 1) + (n-1) * O(1)$$
$$= T(a, 0) + n * O(1)$$

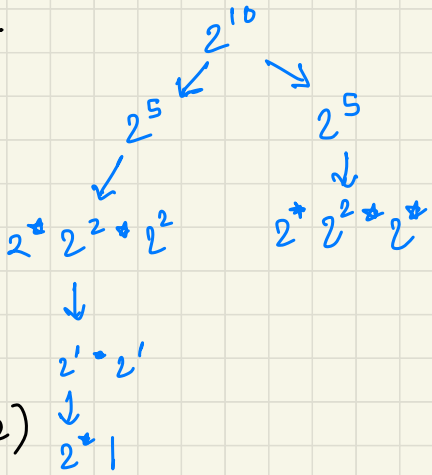
almost instantly returns 1

TC: O(N)

SC: O(N)

Optimized Power Function

```
def power(a,n):
    if n == 0:
        return 1
    if n%2 == 0:
        return power(a, n//2) * power(a, n//2)
    else:
        return power(a, n//2) * power(a, n//2) * a
```



T.C.

$$T(a,n) = 2^* T(a, n//2) + O(1)$$

$$T(a,n) = 2^k \cdot T\left(a, \frac{n}{2^k}\right) + (2^k - 1)O(1)$$

$T\left(a, \frac{n}{2^k}\right) \rightarrow O(1)$

$2^k > n$

$k > \log_2 n$

$= n \cdot O(1) + (n-1)O(1)$

TC ~ O(N) No improvement

SC: O(logN) Improved

Time-optimized Power Function

```
def power(a,n):
    if n==0:
        return 1
    x = power(a, n//2)
    if n%2==0:
        return x*x
    else:
        return a*x*x
```

This time, we store the function call in a var & its being called only once thus reducing time for us.

TC

$$T(a,n) = T(a, n/2) + O(1)$$

$$\downarrow$$

$$T(a, n/4) + O(1)$$

$$\vdots$$

$$T(a,n) = T(a, n/2^k) + k * O(1)$$

$k > \log_2 n$

$$= O(1) + \log_2 n * O(1)$$

$$= O(\log_2 N)$$

TC : $O(\log N)$

SC : $O(\log N)$

(70) Finding remainder for a division by a large number ex. $2^{100} \% 7$

Using Modular Arithmetic Property

```
def mod-power(a, n, M):
    if n == 0:
        return 1
    x = mod-power(a, n//2, M)
    if n % 2 == 0:
        return (x * x) % M
    else:
        return (a * (x * x % M)) % M
```

✱ Calculating TC for a recursive relation

Q.1. $T(N) = 2 * 2T(N-1) + O(1)$

$\downarrow$

$2^2 * T(N-2) + 2 * O(1) + O(1)$

$\downarrow$

$2^3 * T(N-3) + 2^2 O(1) + 2^1 O(1) + 2^0 O(1)$

$= 2^K T(N-K) + (2^K - 1) O(1)$

$O(1)$ when $K = N$

T.C. $= 2^N O(1) + (2^N - 1) O(1)$

TC : $O(2^N)$

Q.2.

$$T(N) = 2T(N/2) + O(1)$$

$$\vdots$$

$$2^K T(N/2^K) + (2^K - 1)O(1)$$

$$\downarrow$$

$$2^K > N$$

$$K > \log_2 N$$

$$N * O(1) + (N-1)O(1)$$

TC : $O(N)$

Q.3 $T(N) = 2T(N/2) + O(N)$

$\downarrow$

$[2^0 T(N/4) + O(N/2)]$

$= 2^2 T(N/4) + 2^1 O(N/2) + 2^0 O(N)$

$\vdots$

$= 2^K T(N/2^K) + 2^{K-1} O(N/2^{K-1}) +$

$\vdots$

$2^1 O(N/2) +$

$2^0 O(N)$

$\sum_{0}^{K-1} 2^i O(N/2^i)$

$O(N)$ is added K times

K ~ $\log_2 N$ times

i.e. $N \cdot \log N$

$= 2^K T(N/2^K) + \sum_{0}^{K-1} 2^i O(N/2^i)$

$N/2^K \to O(1)$

$K > \log_2 N$

$= N \cdot O(1) + N^0 \log N$

TC : $N + N \log N$

(71) Return an array of integers representing gray code sequence

Ex. 2 [0 1 3 2]

00 01 11 10

```
def code(A):
    if A == 1
        return [0, 1]
    value = code(A-1)
    ans = value.copy()
    for i in range(len(value)-1, -1, -1):
        ans.append((1 << (A-1)) + value[i])
    return ans
return code(A)
```

code(2)

value = code(1) = [0, 1]

ans = [0, 1]

i : 1 to 0

i = 1 ans.append(2 + value[1]) → [0, 1, 3]

i = 0 ans.append(2 + value[0]) → [0, 1, 3, 2]

Lecture 18 - Subsequence & subsets

Subarrays : contiguous segment of an array

Non-contiguous segments :

subsequence ← depends on the order → sets

Ex.

$[\ 2, -3, 5, 6, 7, 10\ ]$

$-3, 6, 7 \longrightarrow$ subsequence

$-3, 7, 6 \longrightarrow$ subset

1. All subarrays are subsequences
2. All subsequences are subsets

Q. How many subsequences for an array of N elements ? $\Rightarrow$ 2^N

(72) Given N elements, check if there exists a subsequence which has sum of its elements as k.

arr = [2, 1, 3, -1, 4]

k = 0

[1, -1] → true

Brute Force

[-2, 6, 4]

0	0	0	–	[]
0	0	1	–	[4]
0	1	0		[6]
1	0	0		[-2]
0	1	1		[6, 4]
1	0	1		[-2, 4]
1	1	0		[-2, 6]
1	1	1		[-2, 6, 4]

```
for i in range (2**n):
    sum = 0
    for j in range(n):
        if i & (1<<j) != 0
            sum += A[j]
    if sum == k:
        return True
return False
```

Ex. [-2, 6, 4] K = 2

[-2, 4]

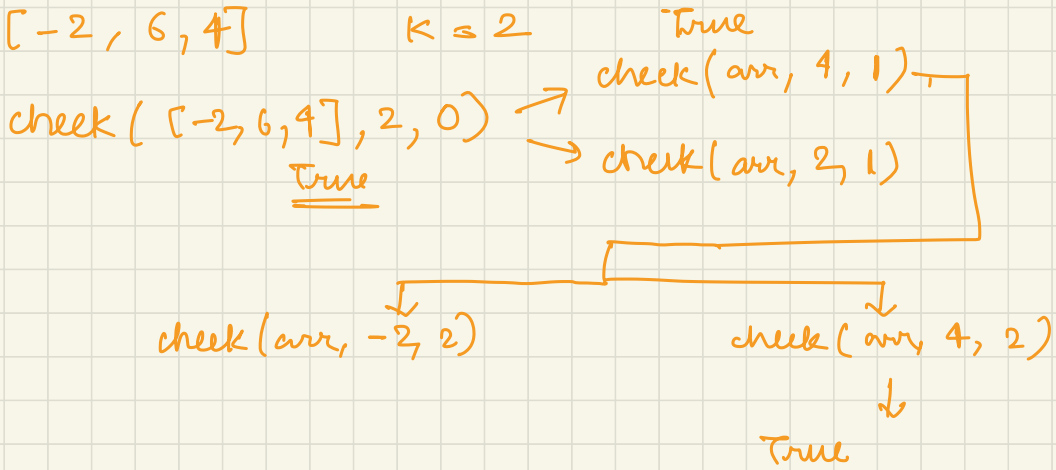
| 0 | (bit representation)

for i = 5 => sum == 2

TC: $O(2^N * N)$

Using recursion

```
def check(arr, k, i):
    if i == len(arr):
        if k == 0:
            return True
        return False
    if check(arr, k-arr[i], i+1) or
       check(arr, k, i+1):
        return True
    return False
```



(73) Find the sum of all subsequences

Brute Force

```
all-sum = 0
for i in range (2**n):
    sum = 0
    for j in range (n):
        if i & (1<<j) != 0
            sum += A[j]
    print(sum)
    all-sum += sum
return all-sum
```

$TC : O(2^N * N)$

Optimized

```
ans = 0
for i in range(n):
    ans += (arr[i] * 2^(n-1))

return ans
```

TC : O(N)

(74) Given N array elements, find sum of max of every subsequence

Brute Force

```
maxes = []
for i in range (2**n):
    max = -sys.maxsize - 1
    for j in range (n):
        if i & (1<<j) != 0
            if arr[j] > max:
                max = arr[j]
    maxes.append (max)
return sum(maxes)
```

TC : $O(2^N * N)$

Optimized

In every subsequence where max(arr) is present

→ max(arr) will be the max → 2^{N-1} subs—

2^{nd} largest (arr) → 2^{N-2} subseqn—

& likewise

```
sum = 0
arr.sort()
for i in range(n):
    sum += 2^i * arr[i]
return sum
```

TC: O(n log n)

75 Find the sum of (max - min) of every subsequence.

```
min_sum = 0
max_sum = 0
for i in range(n):
    max_sum += 2^i * arr[i]
    min_sum += 2^(n-i) * arr[i]
return (max_sum - min_sum)
```

TC : O(n log n)