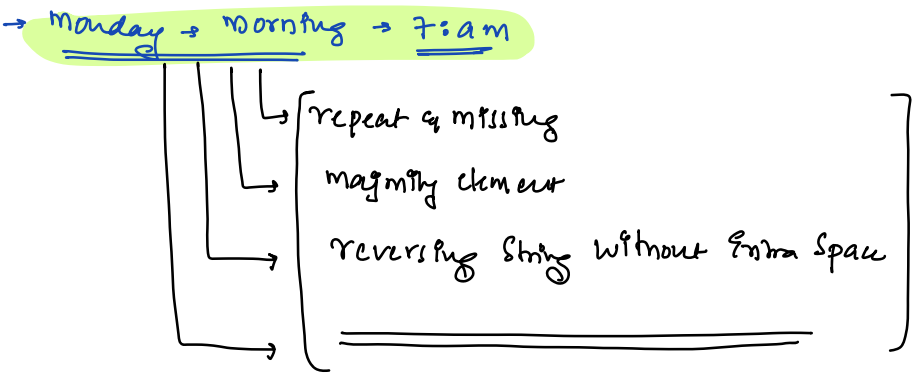
Todays Content :

- Pair Sum = k
- Distinct elements in every window of len = k



10 Given N array elements, check if there exists a pair (i,j) such that $ar[i] + ar[j] = k$ && $(i != j)$ → {bool}

$a + b = k$

	0	1	2	3	4	5	6	7	8	9
ar[] :	8	9	1	-2	4	5	11	-6	7	5

	i	j	A[i] + A[j] == k
k = 11	4	8	4 + 7 == 11
k = 6	2	5	1 + 5 == 6
k = 22	C# Not possible)		

ideas: Check for every pair sum == k

TC: O(N²) SC: O(1)

Pseudo Code: Given ar[N], k

```
i = 0; i < N; i++){
    a = ar[i], b = k - a
    j = i+1; j < N; j++){
        //searching for b in ar[]
        if( ar[j] == b){
            return True
        }
    }
}
return false
```

	0	1	2	3	4	5	6	7	8	9
ar[] :	8	9	1	-2	4	5	11	-6	7	5

Idea: optimization using hashset

↳ Insert all elements in hashset, { This approach won't work }

↳ hs : { 8 9 1 -2 4 5 11 -6 7 }

a + b = 11

a	b[k-a]	b present in hs or Not
8	3	*
9	2	*
1	10	*
-2	13	*
4	7	✓ { return True }

a + b = 5

a	b[k-a]	b present in hs or Not
8	-3	*
9	-4	*
1	4	✓ { return True }

a + b = 22 → Edge Case

a	b[k-a]	b present in hs or Not
8	14	*
9	13	*
1	21	*
-2	24	*
4	18	*
5	17	*
11	11	{ 11 is present in hashset, Return True }

Idea3: Optimization Using hashmap<int, int>

ar[i] → Frequency

	0	1	2	3	4	5	6	7	8	9
ar[]:	8	9	4	-2	4	5	11	-6	7	5

K = 10

Insert all elements in hashmap

{8,1} {9,1} {4,2} {-2,1} {5,2} {11,1} {-6,1} {7,1}

a + b = 10

a	b [k-a]	b present in hm or Not
8	2	*
9	1	*
4	6	*
-2	12	*
4	6	*
5	5	a == b, freq[a] > 1 → freq[5] > 1

We can get required target sum

K = 22:

a + b = 22

a	b [k-a]	b present in hm or Not
8	14	*
9	13	*
4	18	*
-2	24	*
4	18	*
5	17	*
11	11	a == b, hm[a] > 1, hm[11] > 1 *

a	b	b present in hm or Not
-6	28	*
7	15	*
5	17	*

Return False, Because no pair

	0	1	2	3	4	5	6	7	8	9
ar[] :	8	9	4	-2	4	5	11	-6	7	5

k = 11

Insert all elements in Hashmap

{8,1} {9,1} {4,2} {-2,1} {5,2} {11,1} {-6,1} {7,1}

a + b = 11

a	b [k-a]	b present in hm or Not
8	3	x
9	2	x
4	7	a != b : if b is present in hm : return True
		7 is present in hm : return True

bool PairSumMap(int ar[], int k) { ← k → TargetSum

```
    Hashmap<int, int> hm
    Insert all ar[] -> hm // TODO -> N -> O(N)        ] T.C: O(N)
    for(int i=0; i<N; i++) { -> N[1+1] -> O(N)        ] S.C: O(N)
        a = ar[i], b = k-a
        if (a != b && hm.search(b) == true) { return True }
        else if (a==b && hm[a] > 1) { return True }
    }
    return false
}
```

Idea: Optimization using Hashset

	0	1	2	3	4	5	6	7	8	9
ar[]:	8	9	1	-2	4	5	11	7	5	-6

↳ Idea: If we are at i^{th} index, Insert only [0, i-1] elements in hashset

a + b = 11

a	b	hs	b present in hs
8	3	{ }	*
		] Insert 8	
9	2	{8}	*
		] Insert 9	
1	10	{8, 9}	*
		] Insert 1	
-2	13	{8 9 1}	*
		] insert -2	
4	7	{8 9 1 -2}	*
		Insert 4	
5	6	{8 9 1 -2 4}	*
		Insert 5	
11	0	{8 9 1 -2 4 5}	*
		Insert 11	
7	4 =	{8 9 1 -2 4 5 11} ~ {return True}	

idea: Optimization using Hashset

	0	1	2	3	4	5	6	7	8	9
ar[]:	8	9	-1	5	4	11	5	7	-2	-6

↳ Idea: If we are at i^{th} index, Insert only [0, i-1] elements in hashset

a + b = 10

a	b	hs	b present in hs
8	2	{ }	*
		Insert 8	
9	1	{ 8 }	*
		Insert 9	
-1	11	{8 9}	*
		Insert -1	
5	5	{8 9 -1}	*
		Insert 5	
4	6	{8 9 -1 5}	*
		Insert 4	
11	-1	{8 9 -1 5 4}	Return True
		Insert 11	{// we continued, just for explanation sake}
5	5	{8 9 -1 5 4 11}	Return True

Pseudocode:

```
bool TargetSumSet(int ar[], int k){
    int n = ar.length                    TC: O(N)  SC: O(N)
    HashSet<int> hs
    for(int i=0; i<n; i++){
        a = ar[i], b = k - ar[i]
        if( hs.search(b) == True) { return True }
        hs.insert(a)
    }
    return false
}
```

2Q) Given N elements, calculate no of distinct elements in every subarray of size k

	0	1	2	3	4	5	6	7	8	9
Qn: ar[10]:	2	4	3	8	3	9	4	9	4	10

k=4:

Subarrays : print

[0 - 3] : 4
[1 - 4] : 3
[2 - 5] : 3
[3 - 6] : 4
[4 - 7] : 3
[5 - 8] : 2
[6 - 9] : 3

Ideal:
- for every subarray of len = k
  insert into hashset and
  get no of distinct elements → TODO
- TC: (N - k + 1) * (k)
  - k = 1 : (N)(1) = N
  - k = N : (1)(N) = N
  - # Subarray len = k
  - # To get no: of distinct elements in subarray of len = k
  - k = N/2 → (N/2 + 1)(N/2) = O(N²) = max value

SC: O(k)

Idea: optimization using hashset *

	0	1	2	3	4	5	6	7	8	9
Qn: ar[10]:	2	4	3	8	3	9	4	9	4	10

	delete	add	hs	hs.size
[0 3]			hs = {2 4 3 8}	4
[1 4]	ar[0]	ar[4]	hs = {4 3 8}	3
[2 5]	ar[1]	ar[5]	hs = {3 8 9, 3}	3
[3 6]	ar[2]	ar[6]	hs = {8 9 4}	3 *

Note: In hashset, deleting an element will indirectly delete all occurences

Ideas: Optimization Using Hashmap

	0	1	2	3	4	5	6	7	8	9
Eg: ar[10]:	2	4	3	8	3	9	4	9	4	10

	hashmap	timestamp
[0 - 3] →	[<2,1> <4,1> <3,1> <8,1>]	4

Since freq=0, remove from hashmap

s c	remove	add		
[1 - 4]	ar[0]	ar[4]	[~~<2,0>~~ <4,1> <3,2> <8,1>]	3
[2 - 5]	ar[1]	ar[5]	[~~<4,0>~~ <3,2> <8,1> <9,1>]	3
[3 - 6]	ar[2]	ar[6]	[<3,1> <8,1> <9,1> <4,1>]	4
[4 - 7]	ar[3]	ar[7]	[<3,1> ~~<8,0>~~ <9,2> <4,1>]	3
[5 - 8]	ar[4]	ar[8]	[~~<3,0>~~ <9,2> <4,2>] →	2
[6 - 9]	ar[5]	ar[9]	[<9,1> <4,2> <10,1>] →	3

```
void distinctmap(int ar[], int k){
    int n = ar.length
    Hashmap<int, int> hm;
    // Insert 1st subarray elements in hm → [0, k-1]
    for(int i = 0; i < k; i++){ ——→ k*[O(1)+O(1)+O(1)]
        if(hm.search(ar[i]) == True){ ↳ TC: O(k)
            hm[ar[i]]++ = 1
        }
        else{ hm.insert({ar[i], 1})
    }
    print(hm.size())
    s = 1, e = k
    while(e < N){ ——→ TC: N-k [O(1)+O(1)+O(1)+O(1)]
                  TC: O(N-k)
        // sub[s  e], remove ar[s-1] add ar[e]
        hm[ar[s-1]] -= 1 // reduced frequency by 1
        if(hm[ar[s-1]] == 0){
            // remove key ar[s-1] from hashmap
            hm.remove(ar[s-1])
        }
        if(hm.search(ar[e]) == True){
            hm[ar[e]]++ = 1
        }
        else{ hm.insert({ar[e], 1}) }
        s = s+1
        e = e+1
        print(hm.size())
    }
}
```

TC: O(N)
SC: O(k)
At any given point of time we can have k elements