# Problems in Hashing.

## (1). Count of distinct elements :

I/P: arr[] = { 15, 12, 13, 12, 13, 13, 18 }

O/P: 4

Naive

- O(n2)
- Run 2 loops, loop checks if ele already present on left side of it.

---

```
int countDistinct(int arr[], int n)
{
    int res = 0;
    for (int i=0; i<n; i++)
    {
        boolean flag = false;
        for (int j=0; j<i; j++)
        {
            if (arr[i] == arr[j])
            { flag = true;
              break;
            }
        }
        if (flag == false) { res++; }
    }
}
```

Eff Using Hashing

O(n) } space
O(n) } time.

```
int countDistinct(int arr[])
{
    HashSet<Integer> s = new HashSet<>();
    for(int i=0; i<arr.length; i++)
    {
        s.add(arr[i]);
    }
    return s.size()
}
```

add fn designed such that it ignores already present items.

---

② freq of every elem.

I/p: arr[] = {10, 12, 10, 15, 10, 20, 12, 12}

O/p:
| | |
|---|---|
| 10 | 3 |
| 12 | 3 |
| 15 | 1 |
| 20 | 1 |

Algo.

Naive.

⇒ For every ele see on right side, how many times it occurs.

⇒ 2 loops

Eff. Hashmap used. ⇒

i=0 : h = {(10,1)}
i=1 : h = {(10,1), (20,1)}
i=2 : h = {(10,1), (20,2)}
i=3 : h = {——}
i=4 : h = {———}
i=5 : h = {(10,3), (20,2), (30,1)}

Code

```
void printfreq (int arr[])
{   for (int i=0; i<n; i++)
    {   boolean flag = false;
        for (int j=0; j<i; j++)
        {  if(arr[i] == arr[j])
           {  flag = true; break;
           }
        }
        if (flag == true) continue;
        int freq = 1
        for (int j= i+1; j<n; j++)
        {  if (arr[i] == arr[j])
           {  freq++; }
        }
        print (arr[i] + " " + freq);
    }
}
```

checking on left if already seen the element.

checking on right side the freq of ele

» Hashmap Code

(* → If order is to be maintained we here linked Hashmap in Java).

```
int countReq (int arr[])
{ HashMap<Integer, Integer> h = new HashMap<>();
  for (int x : arr)
    { h.put (x, h.getOrDefault (x,0)+1); }
**for (Map.Entry<Integer, Integer> e : h.entrySet())
    { System.out.println (e.getKey() + " " + e.getValue());
    }
}
```

* → we put x i.e 10, 20 something.
  & getOrDefault for either puts x (or) puts 0 by default.
  ↳ i.e (we are getting per freq of x).
  if not there we put 0.

** → for printing.

---

② . Intersection of 2 arrays.

I/p: a[] = {10, 15, 20, 5, 30}
b[] = {30, 5, 30, 80}

O/p: 2 (i.e 30 & 5).

I/p: {10, 10, 10}
{10, 10, 10}

O/p: 1

Naive

» Simi to code in last page
O(n²), check left if num already seen
& then (add it/print it).

Eff sol

Mn-1

Insert a[] in a set (S-a)

Insert b[] in a set (S-b)

Traverse S-a and increment if same ele in S-b. count.

Mn-2

⇒ Implementation of improved eff sol

⇒ Only create S-a, and Traverse through b[].

Search for every ele, if found :-

1) increment count

2) Remove b[i] from S-a

⇒

```
int intersect(int a[], int b[])
{
    Set<Integer> s = new HashSet<>();
    for(int x : a) { s.add(x); }
    for(int x : b)
    { if(s.contains(x))
        { res++; s.remove(x); }
    }
}
```

## (4) Union of 2 sorted Arrays

→ Sols same as prev problems. (Had was intersect, this is union).

I/p : a[ ] = {15, 20, 5, 15}
b[ ] = {15, 15, 15, 20, 10}

O/p : 4 distinct elem.

Sol.

```
for (int x : a)
    h.add(x)
for (int x : b)
    h.add(x)
return h.size()
```

---

## (5) Pair with Given sum in unsorted Array.

I/p : [arr[ ] = {3, 2, 8, 15, -8}, sum = 17.]

O/p : ↳ True.

→ Naive is obvious., {8, 4, 3, 9} → [O(n2), O(1)]

→ In Naive → $\frac{(n) \times (n-1)}{2}$ pairs. for every elem.

Eff

2 pointer approach if sorted array.

→ if not sorted

→ boolean isPair (int [ ]arr, int sum)

```
{
    Set<Integer> h = new HashSet<>();
    for(int x : arr)
    {
        if (h.contains(sum - x))
            return true;
        else {h.add(x);}
    }
    return false;
}
```

* If Imp you compare while adding & not after adding

Variation → Instead of 0, sum is given

## (6) Subarray with sum = 0

I/p : arr = {10, 20, 30}

Subarrays are continuous, subsets of array

∴ {10, 20, 30}
{10}
{10, 20}
{20}
{20, 30}
{30}.

{10, 30} not a subarray.

O/p : NO.

Sol - Naive

```
boolean zerosum (int arr[])
{ for(int i=0; i<n; i++)
  { int curr_sum=0;
    for (int j=i; j<n; j++)
    { curr_sum += arr[j];
      if (curr_sum==0)
      { return true; }
    }
  }
  return false
}
```

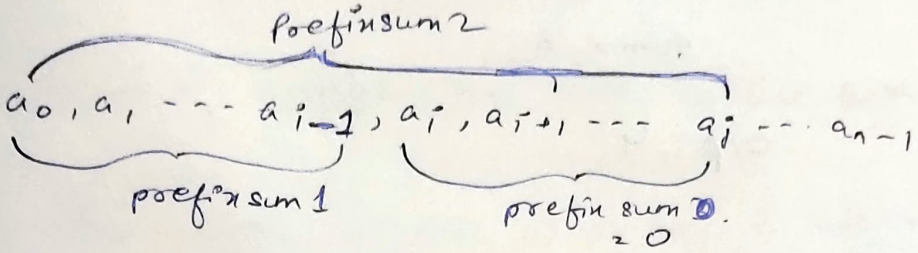
All subarray sum taken

$O(n^2)$.

Eff sol

Prefix sum and Hashing used.

Idea:




∴ Idea is if $a_i + a_{i+1} \cdots a_j$ sum = 0
then
prefix sum 1 = prefix sum 2.

⇒ Code:

```
boolean isZeroSum(int[] arr)
{ HashSet<Integer> h = new HashSet<>();
  int pre_sum = 0;
  for (int i=0; i<arr.length; i++)
  { pre_sum += arr[i];
    if (h.contains(pre_sum))
      {return true;}
    if (pre_sum == 0)
      {return true;}
    h.add(pre_sum);
  }
  return false;
}
```

O(n) time.

## ⑦ Longest Subarray with given sum.

I/p : arr[] = {3, 1, 0, 1, 8, 2, 3, 6}

sum = 5

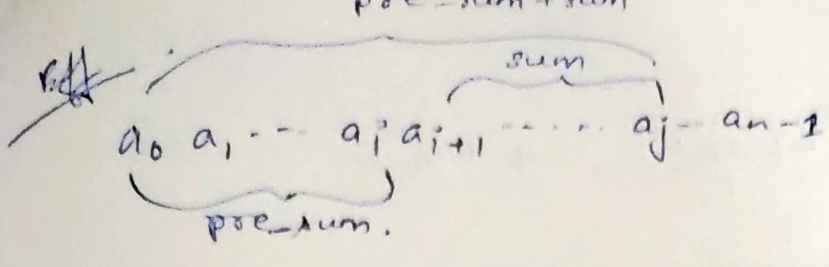
O/p : 4

**Naive :-** Find all subarrays, store length of subarray where it matches sum, return max. length at the end.

```
int maxLen(int arr[], int n, int sum)
{   int res = 0;
    for (int i=0; i<n; i++)
    {   int curr_sum = 0;
        for (int j=i; j<n; j++)
        {
            curr_sum += arr[j];
            if (curr_sum == sum)
            { res = max(res, j-i+1); }
        }
    }
    return res;
}
```

↳ for getting length.

O(n²)




} Already discussed in prev ques.

but instead of hashset, we have to use hashmap.

⇒ To know length we store $a_i$ as key & sum as value.

Eg:

⇒ arr[] = {8, 3, 1, 5, -6, 6, 2, 2} sum = 4

res = max length of subarray.

⇒ m = { }

m = { (8,0) } res = 0
↓ value. ↳ $a_i$ as start point.

m = { (8,0), (11,1) } res = 0

m = { (8,0), (11,1), (12,2) } res = 2

m = { (8,0), (11,1), (12,2), (17,3) } res = 2

m = " } -6,6 can be included in any subarray.

m = "

m ⇒ { (8,0), (11,1), (12,2), (17,3), (19,6) } res = 2

m ⇒ { " , (21,7) } res = 4

we increament res when 21 - sum = already there in Hashmap
Simi 12 - 4 = 8 was there.

Code

```
→ int maxLen(int arr[], int sum)
  {
    Map<Integer, Integer> m = new HashMap<>();
    int pre_sum=0, res=0;
    for(int i=0; i<n; i++)
    {
      presum += arr[i];
      if (pre_sum == sum){res = i+1;}
      if (m.contains(pre_sum) == false)
        { m.put(pre_sum, i); }
      if (m.contains(presum - sum))
        { res = Math.max(res, i - m.get(presum - sum));
        }
    } return res;
  }
```