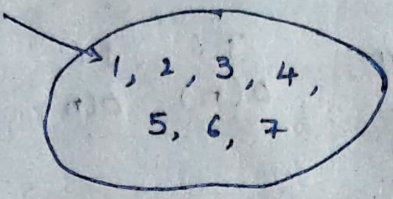
Hash Set:

1, 2, 3, 3, 3, 3, 4, 5, 5, 6, 7



* It will not allow duplicate elements.

Syntax:

HashSet<DataType> set = new HashSet<>();

```
Set.add (1)
set.add (2)
Set.add (3)
Set.add (3)
set.add (4)
```

1
2
3
4

* The add() method internally calls the put() method of Map.

map.put (e, "PRESENT");

→ Before adding the elements the add methods checks if its available in the map.

```
boolean add (Integer e) {
return map.put(e, "PRESENT") == null;
}
```

→ adding '1' to set → map.put (1, "PRESENT") == null → True
map.put (2, " ") == null → True
map.put (3, ") == null → True
map.put (3, ") == null → false

So second time (or) whenever the same element is added to the set it returns false, So it will not add duplicates in the set.

Insertion, Deletion, Searching:

O(1) O(1)

** **Two Sum:**

[1, 2, 3, 4, 5, 6, 7] Target = 5

o/p: (1, 4), (2, 3)

Brute Force:

```
for(int i=0; i<arr.length; i++) {
  for (j= i+1; j<arr.length; j++){
    if (arr[i] + arr[j] == target)
      S.o.pln (arr[i], arr[j]);
  }
}
```

$O(n^2)$

Using hashSet:

```
for (int i=0; i<n; i++){
    int num = arr[i]
    int res = target-num;                T.C      S.C
     if(set.contains (res)){             O(n)     O(n)
       S.O.pln (num, res);
     }
       Set.add (num);
}
```

Contains Duplicate:

I/P: {1,1,1, 2, 3, 2, 2, 4, 4, 5} O/P: True

→ add elements in the set by iterating the array.
→ check if the elements is already available in the set.
→ If available return True.

```
        for (int i=0; i<arr.length; i++){
              if(set.contains (arr[i)] {
                  return true;
              }
        }
            return false;
```

Jewels and Stones:

Jewel = "aA", stones= "aAAbbbb"

→ Check each jewel in the stone.

Brute Force: →Iterating through Jewel & Stones and check each character, if found count++,

```
    for (int i = 0; i< stones.length(); i++){
     for (j= 0;  i < jewel.length(), j++){
        if (jewel.charAt(i)== stones.charAt(j)){
            Count++;
        }
     }
  }
            return Count;
```

T.C → O (mn)

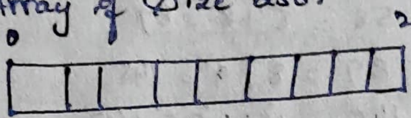
We can use Index of Approach:

t = aA → t.indexof(a) → 0
t.indexof (A) → 1

```
for (int i=0; i< stones.length(); i++){
    char ch = stones.charAt(i);
        if ( jewels.indexof (ch) != -1){
            count++;
        }
}
        return count;
```

Using hash Set: (or) D.A.T

→ Create an Array of Size 256.



→ add '1' to that character to frequency array by iterating the jewels.

→ Iterate the stones, if char from stones is available in the array by checking if the value is 1.

→ If yes increment count.

```
        int[] frequency = new int [26];
        for(int i=0; i< jewels.length(); i++){
            frequency [jewels.charAt(i)]+=1;
        }
        int count =0;
         for (int j=0; j< stones.length();j++){
            if(frequency[stones.charAt(j)] ==1)
            {
             count ++;
            }
         }
            return count;
```

T.C → O(n+m), S.C → O(1)

Happy Number

→ A happy number is defined by the following Process.

a) Starting with any positive integer, replace the number by sum of the square of its digits.

b) Repeat the process until the number equals 1.

c) If ends with '1', it is happy number.

$\boxed{19} = 1^2 + 9^2 = 82$

$82 = 8^2 + 2^2 = 68$

$68 = 6^2 + 8^2 = 100$

$100 = 1^2 + 0^2 + 0^2 = 1$ ✓

$2 = 2^2 = 4$

$\boxed{4} = 4^2 = 16$

$16 = 1^2 + 6^2 = 37$

$37 = 3^2 + 7^2 = 58$

$58 = 5^2 + 8^2 = 89$

$89 = 8^2 + 9^2 = 145$

$145 = 1^2 + 4^2 + 5^2 = 42$

$42 = 4^2 + 2^2 = 20$

$20 = 2^2 + 0^2 = \boxed{4}$

Algorithm:

1. Add the element in the hashset
2. Compute the sum of squares and as u compute, add that number into the set.
3. If computed number is 1 ⇒ return true.
4. If computed number is already present in the set → return false;

```
while (true){
    int sum=0;               n=19
    while(n!=0){
    sum += Math.pow(n%10,2.0);       9² + 1² → 82
        n = n/10;
    }
    if(sum == 1) return true;
    n= sum;            ———→   n = 82
    if( set.contains (n)) {
     retur false;
    }
  set.add(n);   ———→ adding 82 to set
}
```

"Maximum Window SubString"

I/P: S = "ADOBECODEBANC"   O/P: BANC

t = "ABC

ex:- S = ADBC , t = ABC →

A	~~1~~ 0
B	~~1~~ 0
C	~~1~~ 0

'1' A is required
'1' B is required
'1' C is required

'0' A means 'A' is already found.

→ when all the keys in the map have become zero, we have reached one potential answer.

→ In the above approach, we need to iterate through the map to check all the values are zero.

→ So we will have a new Approach with count variable.
initialize count = hashmap.size();

→ when any value becomes zero, reduce the count,
→ when count becomes '0', we reach the potential window.

ex: S = TTTTA , t = TA
↑

T	~~1~~ ~~0~~ → ~~-1~~ -~~2~~ -3
A	~~1~~ 0

So potential window = 5

count = ~~2~~ ~~1~~ 0

→ we got one potential window and potential the answer by optimising it.

shrink the window S = TTTTA
↑↑↑
-2 -1 0

T	-~~3~~ -~~2~~ -~~1~~ 0
A	0

So potential window = ~~5~~ ~~4~~ ~~3~~ $\boxed{2}$

Algorithm:

1. Create two pointers → Window start, and Window End.
2. Dump the elements of String t along with the occurrence of characters in your hashmap.
3. Create another variable called Count → which will be initialized with map's size.
4. Start traversing using j
5. If the element at j location is present in the map.
   a) Decrement the value of that key. If key becomes zero, Count--
   b) If count = = 0
      i) Shrink the window and check whether after shrinking we are getting a better answer or not.
      (ii) Start traversing from ith location and check the value of that particular element in your hashmap.

(iii) Shrink the window till the count = 0.

(iv) Increment the count if any key, value > 0.

```
Map<Character, Integer> targetMap = new HashMap<>();
 for(char c: t.toCharArray()){
    targetMap.put(c, targetMap.getOrDefault(c, 0)+);
 }
 int i=0, j=0;
 int count = targetMap.size();
 int minLen = Integer.MAX-VALUE;
 int ~~minLeft~~ windowStartMin = 0;
 while(j < s.length()){
    char jchar = s.charAt(j);
    if (targetMap.containsKey(jchar)){
       targetMap.put(jchar, targetMap.get(jchar)-1);
       if(targetMap.get(jchar)==0){
          count--;
       }
    }
     while (count==0) {
        if(j-i+1 < minLen) {
           minLen = j-i+1;
           windowStartMin = i;
        }
        char ichar = s.charAt(i);
        if (targetMap.containsKey(ichar)){
           targetMap.put(ichar, targetMap.get(ichar)+1);
           if (targetMap.get(ichar) >0) {
              count ++;
           }
        }
        i++;
    }
    j++;
 }
 return minLen == Integer.MAX-VALUE ? "" : s.substring(windowStartMin,
                                                        windowStartMin+minLen);
```

Longest Consecutive Sequence :

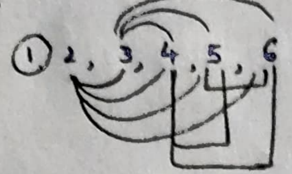
ex: I/P : { 100, ④ 200, ①,③,② } → O/P: 4

ex: I/P: { ⓪ ③ ⑦, ② ⑤ ⑧ ④ ⑥ 0, ① } → O/P: 9

Brute Force:

→ Iterate through the array and check if the next element is available in the array.

ex: ① 2, 3, 4, 5, 6



→ checking all the elements through the loop.

Optimized Approach:-

100, 4, 200, 1, 3, 2

→ Add all elements to the Hash Set

100
4
200
1
3
2

→ Check if the previous num is available in the hash set (because if the previous no. is then only we can consider this no.)

→ If the previous num is not available, increment the number and check in the set, if contains increment the number and length.

→ Get the Max length of length and max length.

$$\begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \\ 100, & 4, & 200, & 1, & 3, & 2 \rightarrow \end{array}$$

At 0: $\underline{\underline{100}}$ → 99 (x) → num+1, 100+1 = 101 (x)

max = 1 length = 1

At 1: $\underline{4}$ → (n-1) $\underline{3}$ ✓ →

It will not compute further

maxlength = 1.

At 2: 200 $\xrightarrow{\text{length = 1}}$ (n-1) $\underline{199}$ x num+1 → 200+1 = 201 (x)

At 3: 1 → (n-1) $\underline{0}$ x → num+1 → 1+1 = 2 ✓

num++; → 2 → length = 2

2 → (n+1) → 3 ✓

num++; 2 → 3 length = 3

3 → (n+1) → 4 ✓

num++; → 3 → 4 $\underline{\text{length = 4}}$

4 → (n+1) → 5 X

$\boxed{\text{maxlength = 4}}$

At 4: 3 → (n-1) → 2 (✓)

At 5: 2 → (n+1) → 1 (✓)

max length = <u>4</u>

$\textcircled{2}$: [0, 3, 7, 2, 5, 8, 4, 6, 0, 1]
(indices above: 0 1 2 3 4 5 6 7 8 9)

At 0: → 0 → (n-1) → -1 (x)
(n+1) → 1 → $\textcircled{1}$ l = 2
1 → n+1 → 2 → $\textcircled{2}$ l = 3
2 → n+1 → 3 → $\textcircled{3}$ l = 4
3 →
⋮

$\boxed{\text{maxlength} = 9}$

```
int maxlength = 0;
Set<Integer> set = new HashSet<>();
  for(int i=0; i<arr.length; i++){
      set.add(arr[i]);
  }
  for(int i=0; i<arr.length; i++){
      int num = arr[i];
      int length = 1;
      if (!set.contains(num-1)){
          while(set.contains(num+1)){
              num++;
              length++;
          }
      }
      maxlength = Math.max(length, maxlength);
  }
  return maxlength;
```

Four Sum:

[1, 0, -1, 0, -2, 2]

⇓

Sort the array.

```
[-2, -1, 0, 0, 1, 2]   → -2 + (-1) + 0 + 2  ⇒ Sum = -1 ≠ 0
  ↑   ↑  ↑        ↑
  i   j  left     right
```

left ++ , and array [left] = array[left-1]

So again left++;

```
[-2, -1, 0, 0, 1, 2]   → -2 + (-1) + 1 + 2 = 0
  ↑   ↑        ↑  ↑                  ↓
  i   j     left  right
                              Add to the list
                              [-2, -1, 1, 2] , left ++, right--;
```

Next:- increment j as left ≠ right

```
[-2, -1, 0, 0, 1, 2]   → -2 + 0 + 0 + 2 ⇒ 0
  ↑      ↑  ↑     ↑                ↓
  i      j  left  right
                              Add to the list
                              [-2, 0, 0, 2] , left ++, right --;

[-2, -1, 0, 0, 1, 2]   → left ≮ right
  ↑      ↑  ↑↑
  i      j  left right
```

So, increment i

```
[-2, -1, 0, 0, 1, 2]   → -1 + 0 + 0 + 2   → 1 ≠ 0
      ↑  ↑  ↑     ↑
      i  j  left  right                      1 > 0
```

right --;

```
[-2, -1, 0, 0, 1, 2]   → -1 + 0 + 0 + 1 = 0
      ↑  ↑  ↑  ↑                   ↓
      i  j  left right
                              add to the list
                              [-1, 0, 0, 1]
```

So final Result contains

[[-2, -1, 1, 2] [-2, 0, 0, 2] [-1, 0, 0, 1]]

```
int n = array.length;
long sum=0;
Set<List<Integer>> set = new HashSet<>();
Arrays.sort(array);
  for(int i=0; i< n-3; i++){
     if( i>0 && array[i]==array[i-1]) continue;
     for(int j=i+1; j<n-2; j++){
       if(j > i+1 && array[j]==array[j-1]) continue;
          int left = j+1;
          int right = n-1;
         while(left < right){
         sum= (long) array[i] + array [j] + array[left] + array[right]
          if(sum > target){
               right -= 1;
          }
          else if (sum < target){
               left += 1;
          }
          else{
               List<Integer> list=new ArrayList<>();
                 list.add (array[i]);
                 list.add (array[j]);
                 list.add (array [left]);
                 list.add (array[right]);
                 Set.add (list);
                 left++; right--;
                                  → while(left< right && array[left]
          }                                             == array[left-1])
       }                                  left++;
    }                                 → while (left <right  && array [right]
}                                                  == array[right+1])
                                                right--;

     return new ArrayList<>(set);
```