Longest Substring

```
map < Character, int> map = new Hash map <>();
 int result =0, left =0 , right =0
 for ( right ; right < str.length ; right ++ );
 {   char ch = s. CharAt (right);
     if (map. Contains Key (ch))
      { left = Math . max ( left, map. get (ch)+1)}
     map. put (ch, r)
     result = Math . max ~~(m, right+1)~~
                          (result, right - left +1)
```

Longest Substring

```
map < Character, int> map = new HashMap<>();
  int result = 0 , left = 0 , right = 0
  for ( right ; right < str.length ; right ++ ).
  {   Char ch = s. CharAt (right);
      if (map. Containskey (ch))
       { left = math . max ( left, map.get(ch)+1)}
       map. put (ch, r)
       result = math . max ~~(m, r-l+1)~~
                                   (result, right - left +1)
```

algo
 map the characters and count
 loop from right side ele to length
 if element present in map

```
int start, end = 0
List < Character > list = new ArrayList < Character > ();

while (end < s. length())
{ if (!list. contains ( s. charAt (end))
   { list. add (s. charAt(end));
     end ++ ;
     max_length = math . max (max_length , list ~~length~~ . size());}

   else
     { list. remove(Character . valueOf (s. charAt (Start)));
       start ++ ;
     }
} return max_length
```

Binary Search.

→ first sort , mid element.

→ divide and Conquer.

~~Search~~

Median of two Sorted Array

```
left = 0 ,    right = list.length - 1 ;

while (left <= right)
      mid = (left + right) / 2

if (list(mid) == target)
      return mid ;
else if (list (mid) < target)
      left = mid + 1 ;
else
      right = mid - 1 ;
```

Merge Sort

~~mergeSort (arr[] , left, right)~~
~~{ if (left < right)~~
~~{ merge Sort (arr, left, mid right)~~
~~merge Sort (arr,~~

[2, 7, 5, 1, 4]

m-1 m+1

m 2

0 1 2 3 4

$n_1 = 2 - 0 + 1 = 3$

$n_2 = 4 - 2 = 2$

```
merge Sort (arr[] , left , right)
{ if (left < right)
    { mid = (left + right) / 2
        mergeSort (arr, left, mid) ;
        merge Sort (arr, mid+1 right) ;
        merge (arr , left, mid, right) ; }
```

(2 7 5)

2 [7 5]

2 [5 7]

1 2 5 7 1 4

1 2 5 7

```
merge (arr[] , left, mid, right)

n1 = mid - left + 1 ;

n2 = right - mid ;

L[] = [2, 7, 5]   R[] [1, 4]
```

```
merge (arr[], left, mid, right)                 [5, 4, 1, 7, 2]
                                                  0, 1  2  3  4
  n1 = mid - left + 1 ;                           l     m     r
  n2 = right - mid ;
                                                   n1 = 2 - 0 + 1 = 3
  L[] = int[n1]
  R[] = int[n2]                                    n2 = 4 - 2 = 2

  for (int i=0 ; i < n1 ; i++)
   { L[i] = arr[left + i]; }      [5, 4, 1]
  for (int j = 0 ; j < n2 ; j++)
   {  R[j] = arr[mid + 1 + j] ; }    [7, 2]

  int i=0 , j=0 , k = left
  while (i < n1 && j < n2)
    { if (L[i] <= R[j])
          arr[k] = L[i]
          i++

       else
          arr[k] = R[j]
          j ++
       } k++ }

  while (i < n1)
   { arr[k] = L[i] ;
       i++
       k++  }

  while (j < n2)
   { arr[k] = R[j]
       j++  }
       k++
```

in Merged two sorted array and find median

```
if (nums1.length > nums2.length)
        return median(nums2, nums1)  // Binary searching on shortest array

int x = nums1.length;
int y = nums2.length;
int low = 0, high = x;

int Part x = (low + high)/2
int Part y = (~~[illegible]~~ x+y+1)/2 - Part x

int maxLeft x = (Part x == 0) ? Integer.MIN_VALUE : nums1[Part x - 1]
int min Right x = (Part x == x) ? Integer.MAX_VALUE : nums1[Part x]

int max left y = (Part y == 0) ? Integer.MINVALUE : nums2[Part y-1]
int min Right y = (Part y == y) ? Integer.MAX_VALUE : nums 2[Part y]
```

nums1 = $\{1, 2, 3\}$, nums2 = $\{4, 5, 9\}$

$m = 3$ $n = 3$

(i) start = 0

(ii) end = 3

(iii) Part 1 = (start + end)/2 = (0 + 3)/2 = 1

(iv) Part 2 = $\frac{m+n+1}{2}$ - part$\left(\frac{3+3+1}{2}\right)$ - 1 = $\frac{7}{2} - 1 = 2$

(v) max left nums 1
~~or~~ max left nums 2
min Right nums 1
min Right nums 2

(vi) (max left num 1 < min Right nums 2) && (max left num 2 < min Right num 2)

median →

longest Palindrome

```
  if (s == null || s.length() < 1) return " ";
  int start = 0, end = 0;
  for (int i = 0; s.length(); i++)
    int len1 = expand(s, i, i)  //odd length palindrom
    int len 2 = expand (s, i, i+1)  // even length Palindrome
    int len = Math.max(len1, len2);

 if (len > end - start)
      start = i - (len -1)/2
      end = i + (len/2)

 return s.substring (start, end+1)


 expand (string, left, right)
   when ((left >= 0) && (right < s.length()) && s.charAt(left) == s.charAt(ri
          left --;
          right ++;
    return right - left -1; // length of palindrome.
```