ARRAYLIST

* In JAVA, ArrayList are inbuilt data structure
* It is a Linear data structure.

* ARRAY	* ARRAYLIST
→ fixed size	→ dynamic size
→ primitive data types can be stored.	→ primitive data types can't be stored directly.

⇒
```
import java.util.ArrayList; or import java.util.*;
public class Arraylist {
  public static void main (String args[]) {
    ArrayList<Integer> list = new ArrayList<>();
  }
}
```

(import java.util.*; → it include all java packages.)

ArrayList → Arraylist class
<Integer> → data type class
list → name of Arraylist
<> → used for data type.

★ OPERATIONS ON ARRAYLIST

i) Add Element ⟶ O(1)
ii) Get Element ⟶ O(1)
iii) Remove Element ⟶ O(n)
iv) Set Element at Index ⟶ O(n)
v) Contains Element ⟶ O(n)

* get() → in get it returns the element of specific index.

* Contains() → it returns true or False regarding checking that element is in list or not.

⇒
```
public class ArrayList {
    public static void main (String args[]) {
        ArrayList<Integer> list = new ArrayList<>();
        //Add Element O(1)
        list.add (1);
        list.add (2);
        list.add (3);
        list.add (4);
        list.add (5);
        System.out.println (list);          [1, 2, 3, 4, 5]
        list.add (1, 9); //O(n)
        System.out.println (list);          [1, 9, 2, 3, 4, 5]
                                             0  1  2  3  4  5
        //get Element O(1)
        int element = list.get (2);         2
        System.out.println (element);
        //Delete Element O(n)
        list.remove (1);
        System.out.println (list);          [1, 2, 3, 4, 5]
        //Set Element O(n)
        list.set (2, 10);
        System.out.println (list);          [1, 2, 10, 4, 5]
        //contains Element O(n)
        System.out.println (list.contains (1));     true
        System.out.println (list.contains (11));    false
    }
}
```

★ Size of the ArrayList

We use a method for finding the size, ⇒ .size().

* We can also traverse using this size method.

⇒
```
public class ArrayList{
  public static void main (String args[]) {
    ArrayList <Integer> list = new ArrayList <>();
    list.add (1);
    list.add (2);
    list.add (3);
    list.add (4);
    list.add (5);
    System.out.println (list.size());
    //print the arraylist
    for (int i=0; i< list.size(); i++) {
        System.out.print (list.get(i) + " ");
    }
    System.out.println();
  }
}
```

Output : 5
1 2 3 4 5

⇒
```
// for reverse print
for (int i = list.size()-1; i>=0; i--) {
    System.out.print (list.get(i) + " ");
}
SystemOut.println();
}
```

* Maximum Element in an ArrayList.

⇒
```
public static void main (String args[]) {
  ArrayList < Integer > list = new ArrayList<>();
  list.add (2);
  list.add (5);
  list.add (9);
  list.add (6);
  list.add (8);
  int max = Integer.MIN_VALUE;
  for (int i=0; i<list.size(); i++) {
      if (max < list.get(i)) {
          max = list.get(i);
      }
  }
  Syo ("max element = " + max);
}
```

★ SWAP TWO NUMBERS

→ List = 2, 5, 9, 3, 6 , → index : idx1 = 1 , idx2 = 3

swap

→ list = 2, 3, 9, 5, 6

⇒
```
public class Arraylist {
    public static void swap (ArrayList<Integer> list,
                             int idx1, int idx2 ) {
        int temp = list.get (idx1);
        list.set (idx1, list.get (idx2));
        list.set (idx2, temp);
    }
    public static void main (String args[]) {
        ArrayList<Integer> list = new ArrayList<>();
        list.add (2);
        list.add (5);
        list.add (9);
        list.add (3);
        list.add (6);
        int idx1 = 1, idx2 = 3;
        System.out.println (list);
        swap (list, idx1, idx2);
        System.out.println (list);
    }
}
```

Output → [2, 5, 9, 3, 6]
[2, 3, 9, 5, 6]

★ SORTING AN ARRAYLIST

* Collections.sort (Arraylist_name);
↳ It is inbuilt & optimised method for Arraylist

⇒
```
import java.util.Collections;
public class Arraylist {
    public static void main (String args[]) {
        Arraylist<Integer> list = new Arraylist<>();
        list.add (2);
        list.add (5);
        list.add (9);
        list.add (3);
        list.add (6);
        System.out.println (list);
        Collections.sort (list); // for ascending order
        System.out.println (list);
    }
}
```
Output → [2, 5, 9, 3, 6]
[2, 3, 5, 6, 9]

```
//for descending order
Collections.sort (list, Collections.reverseOrder ());
```
↓
Comparator function

★ MULTI-DIMENSIONAL ARRAYLIST

→ list1 : 1 2 3 4 5
→ list2 : 2 4 6 8 10
→ list3 : 3 6 9 12 15

AL के अंदर AL के अंदर Int type

```
Arraylist<Arraylist<Integer>> mainlist
AL<Integer> arr = new AL<>()
mainlist.add (arr);
```

* 2-D List:

```
public static void main (String args[]) {
  ArrayList < ArrayList < Integer > > mainList = new ArrayList <>();
  ArrayList < Integer > list = new ArrayList <>();
  list.add(1); list.add(2);
  mainList.add (list);
  ArrayList <Integer> list2 = new ArrayList <>();
  list2.add (3); list2.add (4);
  mainlist.add (list2);
  for (int i=0; i<mainList.size(); i++) {
    ArrayList <Integer> currList = mainList.get (i);
    for (int j=0; j<currList.size(); j++) {
      System.out.print (currList.get (j) + " ");
    }
    System.out.println ();
  }
  System.out.println (mainList);
}
```

Output: 1 2
3 4
[[1, 2], [3, 4]]

* Multi-Dimensional ArrayList:

```
public static void main (String args[]) {
  ArrayList < ArrayList < Integer >> mainlist = new ArrayList <>();
  ArrayList < Integer > list1 = new ArrayList <>();
  ArrayList < Integer > list2 = new ArrayList <>();
  ArrayList < Integer > list3 = new ArrayList <>();
  for (int i=1; i<=5; i++) {
    list1.add (i*1); // 1 2 3 4 5
    list2.add (i*2); // 2 4 6 8 10
    list3.add (i*3); // 3 6 9 12 15
  }
  mainlist.add (list1);
  mainlist.add (list2);
  mainlist.add (list3);
```

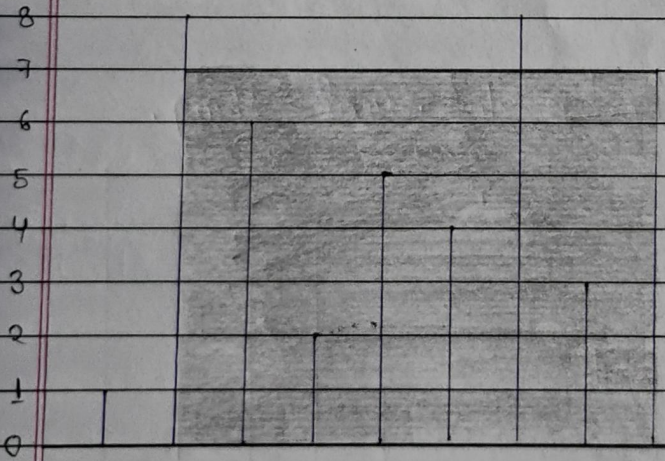
```
System.out.println(mainList);
//nested Loops
for(int i=0; i<mainList.size(); i++){
    ArrayList<Integer> currList = mainList.get(i);
    for(int j=0; j<currList.size(); j++){
        System.out.print(currList.get(j)+" ");
    }
    System.out.println();
}
}
```

Output: [[1, 2, 3, 4, 5], [2, 4, 6, 8, 10], [3, 6, 9, 12, 15]]

1 2 3 4 5
2 4 6 8 10
3 6 9 12 15

★ CONTAINER WITH MOST WATER (2 pointer Approach)

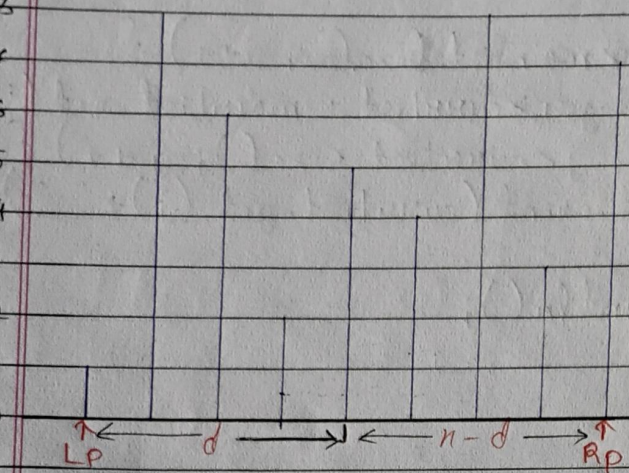
* For n lines on x-axis, use 2 lines to form a container such that it holds maximum ~~Ap~~water.
* 2 pointer Approach is better than Brute force.
  O(n) O(n²)



* Width = 7
* Height = 7
* Water = Width * Height
  or
  Area = 7 × 7 = 49
* Output ⇒ 49

Height = [1, 8, 6, 2, 5, 4, 8, 3, 7]

* 2 pointer Approach (O(n))



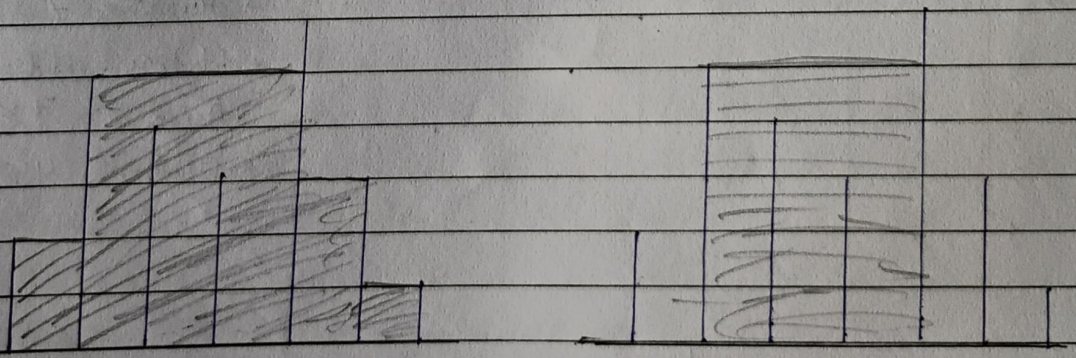
* We will change Lp & Rp by the conditions of height given below.

$\therefore d + n - d = n \Rightarrow O(n)$

* In this approach, we should take two pointers or variables or indexes.

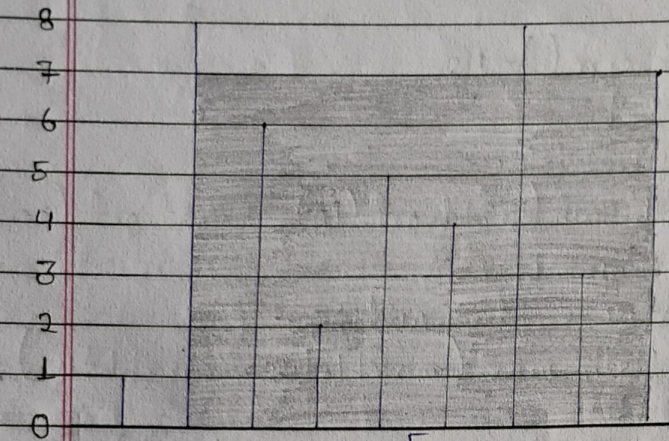
```
⇒ while (Lp < Rp) {
      currWaterWidth = height * width
      height = min (Lp height, Rp height)
      width = Rp - Lp
    if (Lp height < Rp height)
        Lp ++;
    else (Rp height >= Lp height)
        Rp --;
```

#NOTE: Trapping Rainwater is different thing.



⇒
```
public class AL {
  public static int storeWater (ArrayList<Integer> height) {
    int maxWater = 0;
    int lp = 0;
    int rp = height.size()-1;
    while (lp < rp) {
      //calculate water area
      int ht = Math.min(height.get(lp), height.get(rp));
      int width = rp - lp;
      int currWater = ht * width;
      maxWater = Math.max(maxWater, currWater);
      //update pointer
      if (height.get(lp) < height.get(rp)) {
        lp++;
      } else {
        rp--;
      }
    }
    return maxWater;
  }
  public static void main (String args[]) {
    ArrayList<Integer> height = new ArrayList<>();
    //1, 8, 6, 2, 5, 4, 8, 3, 7
    height.add(1);
    height.add(8);
    height.add(6);
    height.add(2);
    height.add(5);
    height.add(4);
    height.add(8);
    height.add(3);
    height.add(7);
    System.out.println(storeWater(height));
```

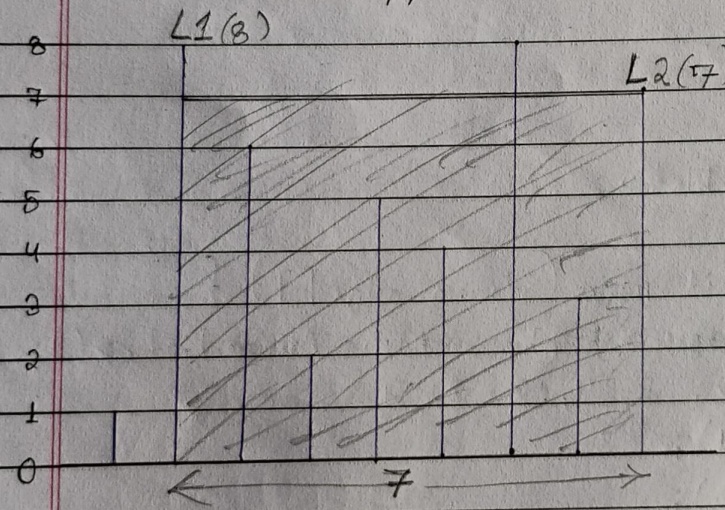
★ CONTAINER WITH MOST WATER – For given n ~~lines~~ lines on x-axis, use 2 lines to form a container such that it holds maximum water.



* width = 8 - 1 = 7
* height = 7
* area = 7 × 7 = 49
or
Volume
* Output = 49

height = [1, 8, 6, 2, 5, 4, 8, 3, 7]

• Brute - force Approach



* In this approach, we check all pairs.

→ Water = height × width

```
⇒ for (int i=0; i< ht.size(); i++)
    for (int j=i+1; j< ht.size(); j++)
        height = min(L1, L2)
        width = j-1
        water = height * width
```

```
⇒ public class AL {   //Brute force - O(n^2)
    public static int storeWater (ArrayList<Integer> height){
        int maxWater = 0;
        //brute force
        for (int i = 0; i<height.size(); i++) {
            for (int j = i+1; j<height.size(); j++) {
                int ht = Math.min(height.get(i), height.get(j));
                int width = j-i;
                int currWater = ht * width;
                maxWater = Math.max(maxWater, currWater);
            }
        }
        return maxWater;
    }
    public static void main (String args[]) {
        ArrayList<Integer> height = new ArrayList<>();
        //1 8 6 2 5 4 8 3 7
        height.add(1);
        height.add(8);
        height.add(6);
        height.add(2);
        height.add(5);
        height.add(4);
        height.add(8);
        height.add(3);
        height.add(7);
        System.out.println(storeWater(height));
    }
}
```

Output : 49

★ PAIR SUM - 1

Find if any pair in a Sorted Arraylist has a target sum.

⇒ List = [1, 2, 3, 4, 5, 6], ⇒ target = 5

2+3
4+1

* Firstly, we will do it with BRUTE FORCE, i.e. means all possible pairs & time complexity $O(n^2)$

⇒
```
public static boolean pairSum1 (ArrayList<Integer> list, int target) {
    for (int i=0; i<list.size(); i++) {
        for(int j=i+1; j<list.size(); j++) {
            if (list.get(i) + list.get(j) == target) {
                return true;
            }
        }
    }
    return false;
}
public static void main (String args[]) {
    ArrayList<Integer> list = new ArrayList<>();
    // 1, 2, 3, 4, 5, 6
    list.add(1);
    list.add(2);
    list.add(3);
    list.add(4);
    list.add(5);
    list.add(6);
    int target = 5;
    System.out.println(pairSum1(list, target));
}
```

DATE

230

* 2 Pointer Approach

case 1: (Lp.number + Rp.number == target)
return true;

case 2: (Lp.number + Rp.number < target)
Lp++;

case 3: (Lp.number + Rp.number > target)

⇒
```
public static boolean pairSum1 (ArrayList<Integer> list, int target) {
    int lp = 0;
    int rp = list.size() - 1;
    while (lp != rp) {
        //case 1
        if (list.get(lp) + list.get(rp) == target) {
            return true;
        }
        //case 2
        if (list.get(lp) + list.get(rp) < target) {
            lp++;
        } else {
            //case 3
            rp--;
        }
    }
    return false;
}
```

→ Some Code

★ PAIR SUM -2

Find if any pair in a Sorted & Rotated Array List has a target sum.

→ List = [11, 15, 6, 8, 9, 10] , → target = 16

* If we want to try BRUTE FORCE APPROACH, then we can write same code of Pair Sum 1 as it is.

→ Find out the pivot point in Array list.

* जहाँ से Array rotated हुआ हो वो Point जैसे कि 10 in [11, 15, 6, 8, 9, 10].

But

* Code में Pivot point मतलब Breaking point, जहाँ SORTING BREAK हो जाता है।

11, 15, | 6, 8, 9, 10

Breaking point

* Lp → left pointer → smallest number → i+1
* Rp → right pointer → largest number → i

→
```
while (lp != rp) {
    lp = (lp + 1) % n      //case 1
    rp = (n + rp - 1) % n  //case 2
}
```

* We will use MODULAR ARITHMETIC OPERATOR.

⇒
```
public static boolean pairSum2 (ArrayList<Integer> list, int target) {
  int bp = -1;
  int n = list.size();
  for (int i=0; i<list.size(); i++) {
    if (list.get(i) > list.get(i+1)) { //breaking point
      bp = i;
      break;
    }
  }
  int lp = bp+1; //smallest
  int rp = bp; //largest
  while (lp != rp) {
    //case 1
    if (list.get(lp) + list.get(rp) == target) {
      return true;
    }
    //case 2
    if (list.get(lp) + list.get(rp) < target) {
      lp = (lp+1) % n;
    } else {
      //case 3
      rp = (n+rp-1) % n;
    }
  }
  return false;
}
public static void main (String args[]) {
  ArrayList<Integer> list = new ArrayList<>();
  list.add(11); list.add(15); list.add(6); list.add(8);
  list.add(9); list.add(10);
  int target = 16;
  System.out.println(pairSum2(list, target));
}
```

QUES 1. Monotonic ArrayList - An Arraylist is monotonic if it is ~~so~~ either monotone increasing or monotone decreasing.

* An ArrayList nums is monotone increasing if for all $i <= j$, nums.get(i) <= nums.get(j). An Arraylist nums is monotone decreasing if for all $i <= j$, nums.get(i) >= nums.get(j).
* Given an integer Arraylist nums, return true if the given list is monotonic or false otherwise.

⇒ Sample Input: nums = [1, 2, 2, 3]
Sample Output: true
⇒ Sample Input: nums = [6, 5, 4, 4]
Sample Output: true
⇒ Sample Input: nums = [1, 3, 2]
Sample Output: false

* Constraints
→ 1 <= nums.size() <= 105
means that the size of an array nums must be between 1 & 105 inclusive. In other words, the array must contain at least 1 element & at most 105 elements.
→ -105 <= nums.get(i) <= 105
means that each element in the array nums must be between -105 & 105 inclusive. So, every value in the array should fall within this range.

```
⇒ public boolean isMonotonic (ArrayList<Integer> A) {
    boolean inc = true;
    boolean dec = true;
    for (int i=0; i<A.size()-1; i++) {
      if (A.get(i) > A.get(i+1))
        inc = false;
      if (A.get(i) < A.get(i+1))
        dec = false;
    }
    return inc || dec;
  }
```

QUES 2. Lonely Numbers in ArrayList - You are given an integer arraylist nums. A number x is lonely when it appears only once, & no adjacent numbers (x+1 & x-1) appear in the arraylist.

* Return all lonely numbers in nums. You may return the answer in any order.

⇒ Sample Input: nums = [10,6,5,8] , Sample Output: [10, 8]

⇒ Sample Input: nums = [1,3,5,3] , Sample Output: [1, 5]

```
⇒ public ArrayList<Integer> findLonely (ArrayList<Integer> nums) {
    Collections.sort(nums);
    Arraylist<Integer> list = new Arraylist<>();
    for (int i=1; i<nums.size()-1; i++) {
      if (nums.get(i-1) + 1 < nums.get(i) && nums.get(i) +
          1 < nums.get(i+1)) {
        list.add(num.get(i));
      }
    }
    if (nums.size() == 1) {
      list.add(nums.get(0));
    }
```

```
if (nums.size() > 1) {
    if (num.get(0) + 1 < nums.get(1)) {
        list.gadd (nums.get(0));
    }
    if (nums.get(nums.size()-2) + 1 < nums.get
        (nums.size()-1)) {
        list.add (nums.get(nums.size()-1));
    }
}
return list;
}
```

QUES 3. Most Frequent Number Following Key - You are given an integer ArrayList nums. You are also given an integer key, which is present in nums.

* For every unique integer target in nums, count the number of times target immediately follows an occurrence of key in nums. In other words, count the number of indices i such that :

  0 <= i <= nums.size() - 2,
  nums.get(i) == key &
  nums.get(i+1) == target.

Return the target with the maximum count.

Sample Input 1 : nums = [1, 100, 200, 1, 100], key = 1.
Sample Output 1 : 100

Sample Input 2 : nums = [2, 2, 2, 2, 3], key = 2
Sample Output 2 : 2.

Constraints: 2 <= nums.size() <= 1000
1 <= nums.get(i) <= 1000
Assume that the answer is unique.

⇒
```
public int mostFrequent (ArrayList<Integer> nums, int key) {
    int [] result = new int [1000];
    for (int i=0; i<nums.size()-1; i++) {
        if (nums.get (i) == key) {
            result [nums.get (i+1) -1] ++;
        }
    }
    int max = Integer. MIN_VALUE;
    int ans = 0;
    For (int i=0; i<1000; i++) {
        if (result [i] > max) {
            max = result [i];
            ans = i+1;
        }
    }
    return ans;
}
```

QUES 4. Beautiful ArrayList - An arraylist nums of size n is beautiful if: nums is a permutation of the integers in the range [1, n].
For every 0 <= i < j < n, there is no index k with i < k < j where 2 * nums.get (k) == nums.get (i) + nums.get (j).
Given the integer n, return any beautiful arraylist nums of size n. There will be at least one valid answer of the given n.

Sample Input: n = 4 ; n = 5
Sample Output: [2, 1, 4, 3] ; [3, 1, 2, 5, 4]

Constraints: 1 <= n <= 1000

⇒
```
public ArrayList<Integer> beautifulArray(int n) {
    ArrayList<Integer> res = new ArrayList<>();
    divideConque(1, 1, res, n);
    return res;
}
public void divideConque(int start, int increment, ArrayList<Integer> res, int n){
    if (start + increment > n) {
        res.add(start);
        return;
    }
    divideConque(start, 2*increment, res, n);
    divideConque(start+increment, 2*increment, res, n);
}
```