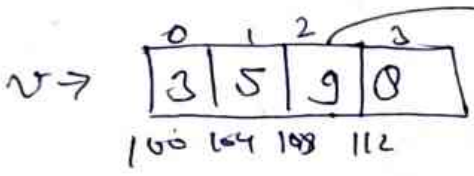
ARRAY

- Array contain similar types of data
- Array stores value in a contiguous memory location



v[0] → 1st location
→ 3
cout << v[0];
→ 3

v[0] = 3
v[3] - 100+3x4 = 112 (0)

initialization

int num[3] = {5, 7, 11}

5	7	11

- int array[10000] = 0; → all array [0, 0, 0 ... 0, 0]
  int array[1000] = 1; → [1, 0, 0, 0, 0 ... 0]
- If arr[15] = {5, 7, 11} → arr is ⇒ [2, 7, 0, 0 ... 0, 0]
- Actual size of array = sizeof(arr) / sizeof int;

if arr[15] = {5, 7, 11}
↓ 15 block ↳ size is 3 only

then in the function (int arr, int size
{ ____
}

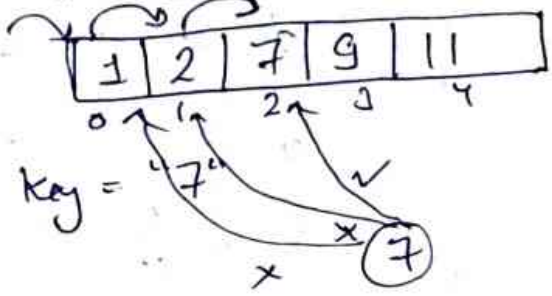
- range of int is $(-2^{31}-1, 2^{31})$
  INT_MIN ↳ INT_MAX
- { Predefined function for finding minimum and maximum.
  ⇒ int maximum = INT_MIN
  maximum = max(maximum, num[i]);
  OR
  if (num[i] > max) {
  max = num[i];
  }

★ In the local function if we update the array then the original array will also be update.

- But in case of integer. if we update value in local function then original value in main function will never be updated. (Call by Value)

LINEAR SEARCH

arr[5] = {1, 2, 7, 9, 11}




```
if (key == arr[i])
  { return 1
  }
else return 0;
```

code

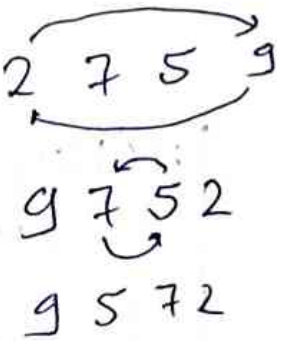
```
bool search(int arr[], int size, int key) {
    for (int i=0; i<size; i++) {
        if (arr[i] == key) {
            return 1;
        }
    }
    return 0;
}
int main() {
    int arr[10] = {5, 7, -2, 10, 22, -4, 95, 22, 13};
    cout << " Enter the element to search for" << endl;
    int key;
    cin >> key;
    bool found = search(arr, 10, key);
    if (found) {
        cout << " key is present " << end;
    }
    else {
        cout << " key is absent" << endl;
    }
    return 0;
}
```

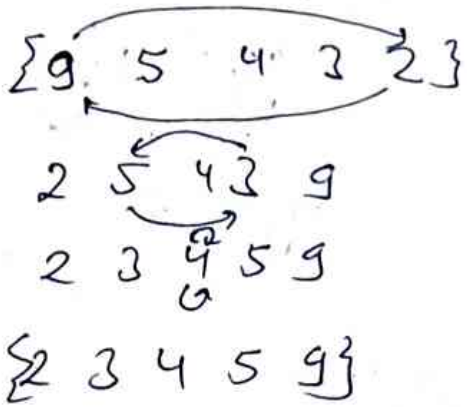
Reverse an Array.

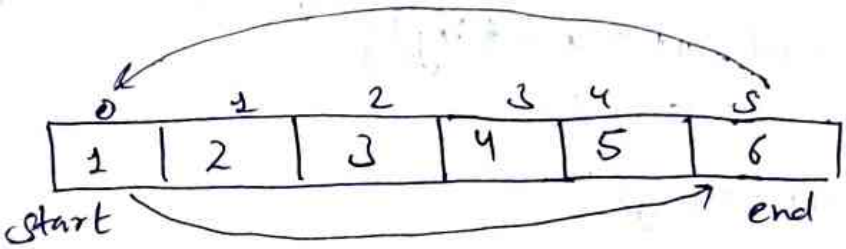
i/p ⇒ {2, 7, 5, 9}

o/p ⇒ {9, 5, 7, 2}

Logic →







Algo → swap → (start value swap end value)

→ start++, end--;

if (start > end) → stop

code

```
void reverse (int arr[], int n) {
    int start = 0;
    int end = n-1;
    while (start <= end)
        swap (arr[start], arr[end]);
        start++;
        end--;
    }
}
void printArray (int arr[], int n) {
    for (int i=0; i<n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}
```

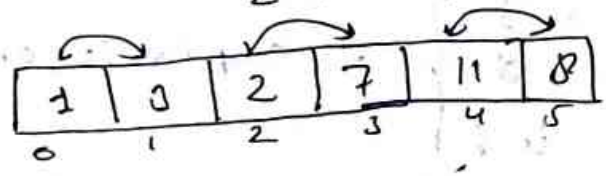
```
int main() {
    int arr[6] = {1, 4, 0, 5, -2, 15};
    int brr[5] = {2, 6, 3, 9, 4}
    reverse (arr, 6);
    reverse (brr, 5);
    printArray (arr, 6);
    printArray (brr, 5);
    return 0;
}
```

8 Swap Alternate

i/p → arr[5] = {1, 2, 7, 8, 5}

o/p → = {2, 1, 8, 7, 5}

Algo




```
For (i → 0-5)          (i = i+2)
{
    if (i+1 < size
    {
        swap (arr[i], arr[i+1]
    }
}
```

Code

```
void printArray (int arr[], int n) {
    for (int i=0; i<n; i++)
    {
        cout << arr[i] << " ";
    }
    cout << endl;
}
void swapAlternate (int arr[], int size) {
    for (int i=0; i<size; i+=2) {
        if(i+1 < size) {
            swap (arr[i], arr[i+1]);
        }
    }
}
```

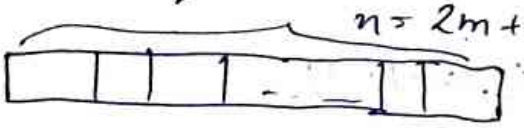
```
int main()
{
    int even[8] = {5, 2, 9, 4, 7, 6, 1, 0};
    int odd[5] = {11, 33, 9, 76, 43};

    swapAlternate (even, 8);
    printArray (even, 8);
    cout << endl;
    swapalternate (odd, 5);
    printArray (odd, 5)
    return 0;
}
```

★ How to swap values without using the swap function

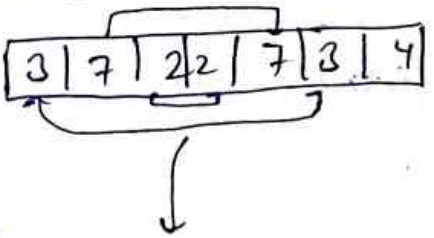
Sol
$$\left\{\begin{array}{l} temp = arr[1] \\ arr[1] = arr[0] \\ arr[0] = temp \end{array}\right\} = swap(arr[0], arr[1])$$

Q find uniques numbers in Array. (Coding Ninjas)



n → odd.
"m" num → twice
1 num → appear 1

Eg m=3



4 came only 1 time find that number

O/P → 4

Algorithm

we know that $x \wedge x = 0$

$0 \wedge x = x$

So $3 \wedge 7 \wedge 2 \wedge 2 \wedge 7 \wedge 3 \wedge 4 = 4$

code

```
int findUnique (int *arr, int size)
{   int ans = 0;
    for (int i=0; i<size; i++) {
        ans = ans^arr[i];
    }
  return ans;
}
```

Q Leetcode (Unique no of occurence)

i/p ⇒ {1, 2, 2, 1, 1, 3}

1 → 3 times
2 → 2 times
3 → 1 time

o/p ⇒ true

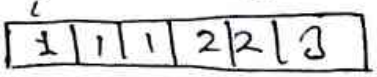
i/p = {1, 2, 2, 1, 1, 3, 3}

1 → 3 times
2 → 2 times
3 → 2 times

o/p ⇒ false

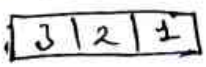
Approach

Step① Sort the array.



count = 1
i = count+1

step2 from ans array.



true

code

```
bool uniqueOccurence (vector<int> & arr) {
    vector<int> ans;
    int size = arr.size();
    sort (arr.begin(), arr.end());
    int i = 0;
    while (i<size)
        int count = 1
        for (int j=i+1; j<size; j++) {
            if (arr[i] == arr[j]) {
                count++;
            }
```

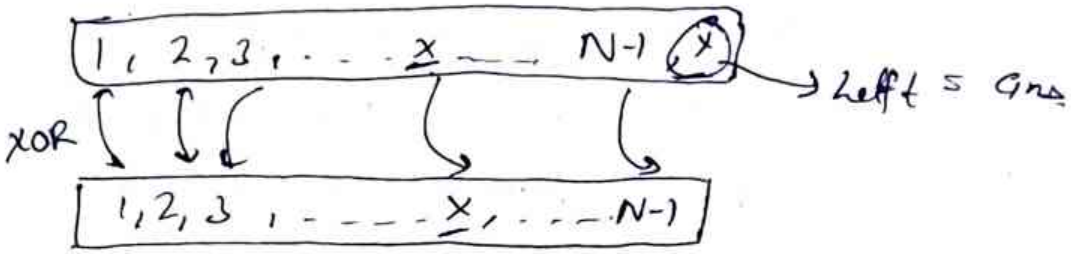
```
            else {
                break;
            }
        }
        ans.push_back(count);
        i = it.count;
    }
    size = ans.size();
    sort(ans.begin(), ans.end());
    for(int i=0; i<size-1; i++){
            if(ans[i] == ans[i+1]){
                return false;
            }
    }
    return true
}
```

Q Find Duplicate in Array (code studio)

i/p → | 2 | 1 | 3 | 4 | 5 | 2 | → 1 to N-1

o/p → 2

Logic



code

```
int findDuplicate(vector<int>& arr)
{
    int ans = 0;
    for(int i=0; i<arr.size(); i++) {
        ans = ans ^ arr[i];
    }
    for(int i=1; i<arr.size(); i++) {
        ans = ans ^ i;
    }
    return ans;
}
```

---

442. Find all Duplicate in an Array (leetcode)

condition ⇒ $1 \le a[i] \le n$

Approach

0	1	2	3	4	5	6	7
-4	-3	-2	-7	8	2	-3	-1

i=0	i=1	i=2	i=3	i=4	i=5	i=6	i=7
arr[i] = 4	arr[i] = 3	arr[i] = 2	7-1=6	8-1=7	2-1=1	3-1=2	,
4-1=3	3-1=2	2-1=1	go to 6				
go to 3	go to 2	go to 1	3 → -3	1 → -1	-3 already there in ans	-2 already in ans	
7 → -7	2 → -2	3 → -3					

ans ← [2, 3]

~~Inter~~ code

```
public List<Integer> findDuplicates(int[] nums) {
    List<Integer> resultSet = new ArrayList<>();

    for (int i = 0; i < nums.length; i++) {
     // Get the index, the element corresponds to
     int index = Math.abs(nums[i]) - 1;
    // if num is already negative, it means we are
    // encountering it twice
    if (nums[index] < 0)
        resultSet.add(index + 1);
    // Flip the number at the index to negative
     nums[index] = nums[index] * -1
    }
    return resultSet;
}
```

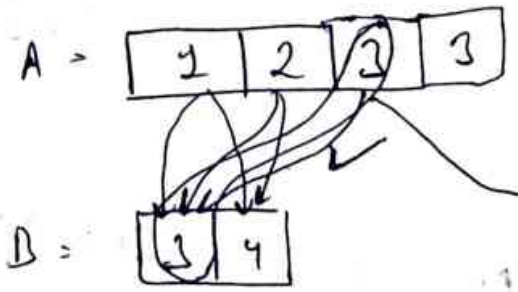
Intersection of two Arrays. (Code studio) (imp)

→ if intersection not present return -1

sorted [non-dec order]

i/p → A = {1, 2, 3}
B = {3, 4}
} Ans (3)

Logic 1



3 → aage nahi kaam update value (when no match again)
so break;

TC = O(mn)

Code (Ghatia Approach (TLE mox)

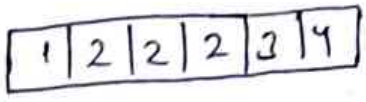
```
vector<int> findArrayIntersection(vector<int> &arr1, int n,
                                  vector<int>
{
    vector<int> ans;
    for(int i=0; i<n; i++){
        int element = arr1[i]
        for(int j=0; j<m; j++){
            if(element == arr2[j]){
                ans.push_back(element);
                arr2[j] = -11223;   → INT_MIN
                break;
            }
        }
    }
    return ans;
```

- If we add like in b/w (more optimize than previous)

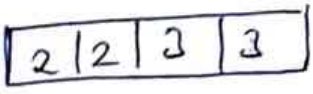
```
if(element <= arr2[j])
    break;
```

- Again TLE is comming.

★ approch 3 (Two pointer loop)



ar[i] < ar[j]
i++



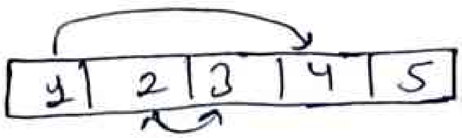
ar[i] == ar[j]
print / vector
↓
i++, j++

ar[i] > ar[j] ;
j++;

Code (Optimize code No TLE) T.C = O(n, m)

```
vector<int> findArrayIntersection (vector<int> &arr1, int n,
                                   vector<int> ...
{   int i = 0, j = 0
    vector<int> ans;
    while (i<n && j<m) {
      if (arr1[i] == arr2[j])
      {
          ans.push_back (arr1[i]);
          i++;  j++;
      }
      else if (arr1[i] < arr2[j]) {
          i++;
      }
      else
      {
          j++;
      }
    } return ans;
}
```

Q Pair Sum (code studio)

i/p 

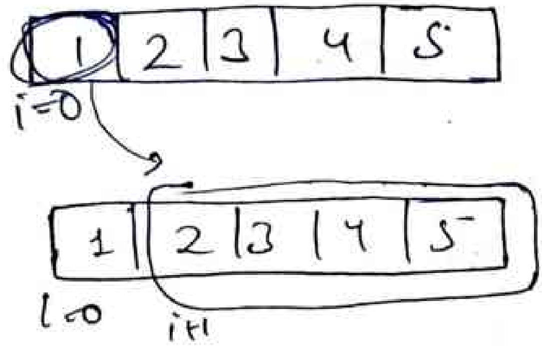
i/p S = 5

O/P : (short)

1	4
2	3

(short)

Logic



for (0 → n-1)
↳ for (i+1 → n-1)

Code

```
vector<vector<int>> pairSum (vector<int> &arr, int s) {

    vector<vector<int>> ans;

    for (int i=0 ; i<arr.size() ; i++)
    {
        for (int j= i+1;  j<arr.size(); j++) {
            if (arr[i] + arr[j] == s)

            {   vector<int> temp;
                temp.push_back(min(arr[i], arr[j]));
                temp.push_back(max(arr[i], arr[j]));
                ans.push_back(temp);
            }
        }
    }
    sort(ans.begin(), ans.end());
    return ans;
}
```

Q Triplet with given sum (3sum) (code studio) (50)

size ↑ sum ↑

```
vector<vector<int>> findTriplets(vector<int> arr, int n, int k)
{
    vector<vector<int>> ans;
    // Sorting the vector
    sort(arr.begin(), arr.end());

    for(int i=0; i<n; i++) {
        int target = k - arr[i]
        int front = i+1
        int back = n-1;
        while(front < back) {

            int sum = arr[front] + arr[back];

            // finding answer which starts from arr[i].
            if(sum < target) {
                front++;
            }
            else if (sum > target)
                back--;
            }
            else {
                int x = arr[front];
                int y = arr[back];
                ans.push_back({arr[i], arr[front], arr[back]});
// Increment front pointer until we reach a diff num
                while(front < back && arr[front] == x){
                    front++;
                }
```

```
// Decrement last pointer until we reach a diff num.
        while ( front < back && arr[back] == y) {
            back--;
        }
    }
}

// Ensure that we don't encounter duplicate value for arr[i
        while ( i+1 < n && arr[i] == arr[i+1]) {
            i++;
        }
    }
    return ans;
}
```

Dry run

arr ⇒ | 7 | -36 | 5 | 3 | 41 | 5 |

Sum = 10 = k

sort → | -36 | 3 | 5 | 5 | 7 | 41 |
idx: 0 1 2 3 4 5

target ⇒ 10 − (−36) ⇒ 26

f < b

Sum = 44

5 > t

| -36 | 3 | 5 | 5 | 7 | 41 |
f b

Sum = 10

5 > t

| -36 | 3 | 5 | 5 | 7 | 41 |
f b

Sum = 8

5 > t

| -36 | 3 | 5 | 5 | 7 | 41 |
f b

5 > t

| -36 | 3 | 5 | 5 | 7 | 41 |
f b

9 Sort 01

i/p | 0 | 1 | 1 | 0 | 0 | 1 |

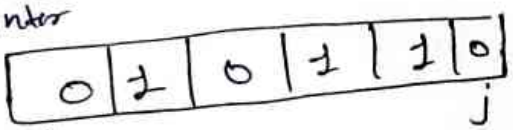
o/p | 0 | 0 | 0 | 1 | 1 | 1 |

Solution

counting	sort	two-pointer approach
0 → 3 1 → 3 Traverse 0/1 2 Traverse ✓	(n log n)	single traverse O(n)

Algorithm 2 pointer

0 → left
1 → right



arr[i] == 0
i++;

arr[j] == 1
j--

arr[i] == 1 & arr[j] == 0
swap (arr[i], arr[j])
i++; j--

i >= j → loop stop

code

```
void printArray(int arr[], int n){
  for(int i=0; i<n; i++){
    cout << arr[i] << " ";
  }
  cout << endl;
}
```

```
void sortOne (int arr[], int n) {
  int left = 0, right n-1;
  while (left < right) {
    while (arr[left] == 0 && left < right) {
      left++;
    }
    while (arr[right] == 1 && left < right) {
      right--;
    }
    // agar yha pohuch gye to iska matlab
    // arr[left] == 1 and arr[right] == 0.
    if (left < right)
    {
      swap (arr[left], arr[right]);
      left++; right--;
    }
  }
}
int main () {
  int arr[8] = {1,1,0,0,0,0,1,0}
  sortOne (arr, 8);
  printArray (arr, 8);
  return 0;
}
```

Similar Question sort 0 1 2

i/p = | 0 | 2 | 2 | 1 | 0 | 1 | 1 | 0 | 2 |

o/p = | 0 | 0 | 0 | 1 | 1 | 1 | 2 | 2 | 2 |

Approach 1 (Use any sorting algorithm)

TC → O(Nlog(N))

we are using inbuilt sort algorithm.

Code

```
void sort012 ( int *arr , int n)
{
    sort(arr, arr+n);
}
```

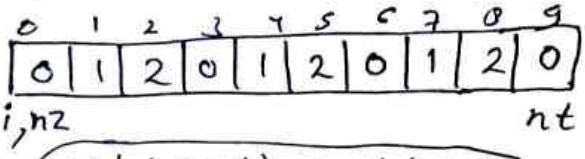
Approach 2 (count 0s, 1s, 2's then fill the array)

Code

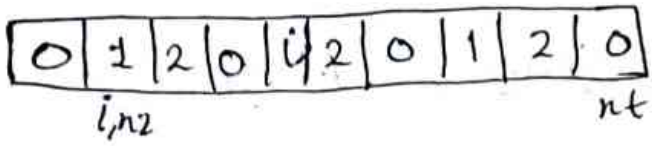
```
void sort012 (int *arr, int n)
{
    int i=0;
    int count[3] = {0,0,0}
    // Storing count of 0s, 1s and 2s
    for(i=0; i<n; i++)
    {   count [arr[i]]++ ;
    }
    i=0;
    while (count [0]--)          // filling 0 to array
    {   arr [i++]=0;
    }
    while ( count[1]--)          // filling 1
    {   arr [i++]=1;
    }
    while (count [2]--)          // filling 2
    {   arr [i++]=2;
    }
}
```

Approach 3 (3 pointer swap approach)

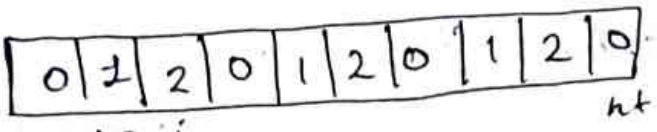
Initially



if(i > nt) → call swap



If i = 0
swap (ar[i], ar[nz])
i++; nz++



if i = 1
then i++

if i = 2
swap (ar[i], ar[nxt])
nt--;

Code

```
void sort012 (int *ar, int n)
{
    int i = 0;
    int nz = 0
    int nt = n-1;
    while( i <= nt)
    {   if (ar[i] == 0)
        {   swap ( ar[i], ar[nextZero]
            i++;
            nz++;
        }
        else if (ar[i] == 2)
        {
            swap(ar[i], ar[nexttwo]);
            nt--;
        }
        else
            {   i++;
            }
    }
}
```