Cyclic Sort:

When Given range is from 1 to N to sort it, use Cyclic sort

def cyclic\_sort(nums):

i = 0

while i < len(nums):

correct\_index = nums[i]

if correct\_index < len(nums) and nums[i] != nums[correct\_index]:

# Swap the elements at index i and correct\_index

nums[i], nums[correct\_index] = nums[correct\_index], nums[i]

else:

i += 1

# Example usage:

arr = [3, 0, 2, 1, 4]

cyclic\_sort(arr)

print(arr)

OP:

[0, 1, 2, 3, 4]

|  |  |
| --- | --- |
|  | [Missing Number](https://leetcode.com/problems/missing-number/) |

class Solution:

def missingNumber(self, nums: List[int]) -> int:

i=0

while i<len(nums):

j=nums[i]

if nums[i]< len(nums) and nums[i]!=nums[j]:

temp=nums[i]

nums[i]=nums[j]

nums[j]=temp

else:

i+=1

for i in range(len(nums)):

if nums[i] != i :

return i

return len(nums)

[Find All Numbers Disappeared in an Array](https://leetcode.com/problems/find-all-numbers-disappeared-in-an-array/)

class Solution:

def findDisappearedNumbers(self, nums: List[int]) -> List[int]:

res=[]

i=0

while i<len(nums):

idx=nums[i]-1

if idx<len(nums) and nums[idx]!=nums[i]:

nums[idx],nums[i]=nums[i],nums[idx]

else:

i+=1

print(nums)

k=0

while k<len(nums):

if k!=nums[k]-1:

res.append(k+1)

k+=1

return res

TIPS:

If range is 0 to n:

Every element will be at index=value

If range is 1 to n:

Every element will be at index=value-1

[Find the Duplicate Number](https://leetcode.com/problems/find-the-duplicate-number/)

class Solution:

def findDuplicate(self, nums: List[int]) -> int:

i=0

while i<len(nums):

idx=nums[i]-1

if idx<len(nums) and nums[idx]!=nums[i]:

nums[idx],nums[i]=nums[i],nums[idx]

else:

i+=1

k=0

while k<len(nums):

if nums[k]!=k+1:

return nums[k]

k+=1

return nums[-1]

[Find All Duplicates in an Array](https://leetcode.com/problems/find-all-duplicates-in-an-array/)

class Solution:

def findDuplicates(self, nums: List[int]) -> List[int]:

res=[]

for i in range(len(nums)):

idx=abs(nums[i])-1

if (nums[idx]<0):

res.append(idx+1)

else:

nums[idx]=-1\*nums[idx]

return res

**C++**

**Pair**

#include <iostream>

using namespace std;

int main()

{

pair<int,string> p;

// p=make\_pair(2,"abc");

p={2,"abc"};

cout<<p.first<<" "<<p.second<<"\n";

pair<int,int> p\_array[3];

p\_array[0]={1,2};

p\_array[1]={6,3};

p\_array[2]={4,5};

for (int i=0;i<3;i++){

cout<<p\_array[i].first<<" "<<p\_array[i].second<<"\n";

}

return 0;

}  
oop

2 abc

1 2

6 3

4 5

**VECTOR**

#include <iostream>

#include <vector>

using namespace std;

void printvec(vector<int>& v){

cout<<"size is "<<v.size()<<"\n";

for(auto it:v){

cout<< it<<"\n";

}

}

int main()

{

//dynamic array

vector<int> v;

int n;

cin>>n;

for (int i=0;i <n;i++)

{ int x;

cin>>x;

v.push\_back(x);

}

v.push\_back(2);

cout<<"OP"<<"\n";

printvec(v);

cout<<"popping"<<"\n";

v.pop\_back();

printvec(v);

vector<int> v2=v;

v2.push\_back(66);

printvec(v2);

return 0;

}

4

1

2

35

5

OP

size is 5

1

2

35

5

2

popping

size is 4

1

2

35

5

size is 5

1

2

35

5

**66**

TIT BITS:

y = 2; // Initialize y with the value 2

int &x = y; // Declare a reference x that refers to y

x = 3; // Change the value of x to 3, which also changes the value of y

cout << y; // Print the value of y, which is now 3

When you run this code, it will output 3 because the value of y has been changed to 3 by modifying x, which is a reference to y

+---+ +---+

| y | ---> | 3 |

+---+ +---+

^ ^

| |

x is an alias for y