Day 2-

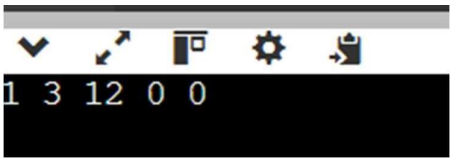
Student Name: Rajan Mishra UID: 22BCS13621
Branch: BE-CSE Sec/Group:22BCS-IOT-615

1. Given an integer array nums, move all 0's to the end of it while maintaining the relative order of the non-zero elements.
Note: that you must do this in-place without making a copy of the array.
Example 1: Input: nums = [0,1,0,3,12] Output: [1,3,12,0,0]
Example 2: Input: nums = [0] Output: [0]
Constraints: • 1 <= nums.length <= 1e4 •-2e31 <= nums[i] <= 2e31 - 1 Follow up: Could you minimize the total number of operations done

CODE:

```
#include <iostream> #include <vector>
using namespace std; void
moveZeroes(vector<int>& nums) {
int n = nums.size();     int lastNonZero =
0;     for (int i = 0; i < n; i++) {         if
(nums[i] != 0) {
            swap(nums[lastNonZero++], nums[i]);
        } }} int main() {     vector<int>
nums = {0, 1, 0, 3, 12};
   moveZeroes(nums);
for (int num : nums) {
        cout << num << " ";
   }
return 0;
}
```

OUTPUT:

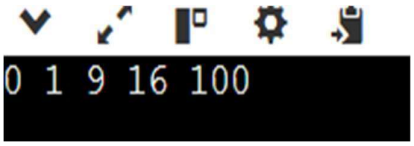


2. Given an integer array nums sorted in non-decreasing order, return an array of the squares of each number sorted in non-decreasing order.
Example 1: Input: nums = [-4,-1,0,3,10] Output: [0,1,9,16,100]
Explanation: After squaring, the array becomes [16,1,0,9,100]. After sorting, it becomes [0,1,9,16,100].
Example 2: Input: nums = [-7,-3,2,3,11] Output: [4,9,9,49,121]
Constraints: • 1 <= nums.length <= 1e4 •-1e4 <= nums[i] <= 1e4 • nums is sorted in nondecreasing order

CODE:

```
#include <iostream>
#include <vector> #include <algorithm> using
namespace std; vector<int>
sortedSquares(vector<int>& nums) {
   int n = nums.size();     vector<int> result(n);
int left = 0, right = n - 1;     int index = n - 1;
while (left <= right) {         if (abs(nums[left]) >
abs(nums[right])) {             result[index--] =
nums[left] * nums[left];            left++;         }
else {
          result[index--] = nums[right] * nums[right];
right--;}}     return result;} int main() {
vector<int> nums = {-4, -1, 0, 3, 10};
vector<int> result = sortedSquares(nums);
   for (int num : result) {
cout << num << " ";}
return 0;}
```

OUTPUT:

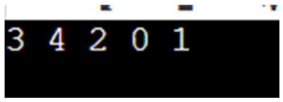


3. Given an array arr[] of size N where every element is in the range from 0 to N-1, rearrange the given array so that the transformed array arr^T[i] becomes arr[arr[i]]. Note: arr and arr^T are the same variables, representing the array before and after transformation, respectively.
Example 1: Input: 2 1 0 Output: 0 1 Explanation: arr[arr[0]] = arr[1] = 0 arr[arr[1]] = arr[0] = 1 So, arr^T becomes {0, 1}

CODE:

```
#include <iostream>
#include <vector> using
namespace std;
void rearrange(vector<int>& arr) {
int n = arr.size();     for (int i = 0; i
< n; i++) {         arr[i] +=
(arr[arr[i]] % n) * n;
   }
   for (int i = 0; i < n; i++) {
arr[i] /= n;}} int main() {
   vector<int> arr = {4, 0, 2, 1, 3};
rearrange(arr);     for (int num :
arr) {         cout << num << " ";}
return 0;}
```

OUTPUT:



4. Given the head of a singly linked list, reverse the list, and return the reversed list. Input: head = [1,2,3,4,5] Output: [5,4,3,2,1] Input: head = [1,2] Output: [2,1]

CODE:

```
#include <iostream> using
namespace std;
struct ListNode {
   int val;
   ListNode* next;
   ListNode(int x) : val(x), next(nullptr) {}};
ListNode* reverseList(ListNode* head) {
   ListNode* prev = nullptr;
ListNode* curr = head;
   while (curr) {
      ListNode* nextTemp = curr->next;
      curr->next = prev;
prev = curr;        curr =
nextTemp;}     return
prev;}
void printList(ListNode* head) {
while (head) {         cout <<
head->val << " ";         head =
head->next;}} int main() {
   ListNode* head = new ListNode(1);     head->next
= new ListNode(2);     head->next->next = new
ListNode(3);     head->next->next->next = new
ListNode(4);     head->next->next->next->next = new
ListNode(5);     ListNode* reversed =
reverseList(head);
   printList(reversed);
return 0;}
```

OUTPUT:

5. Calculate the Factorial of a Number Given the head of a singly linked list, return the middle node of the linked list. If there are two middle nodes, return the second middle node. Input: head = [1,2,3,4,5] Output: [3,4,5] Explanation: The middle node of the list is node 3. Input: head = [1,2,3,4,5,6] Output: [4,5,6] Explanation: Since the list has two middle nodes with values 3 and 4, we return the second one

CODE:

```
#include <iostream> using
namespace std;
struct ListNode {
   int val;
   ListNode* next;
   ListNode(int x) : val(x), next(nullptr) {}};
ListNode* middleNode(ListNode* head) {
   ListNode* slow = head;
ListNode* fast = head;     while
(fast && fast->next) {         slow
= slow->next;         fast = fast-
>next->next;}     return slow;}
void printList(ListNode* head) {
while (head) {         cout <<
head->val << " ";         head =
head->next;}}
int main() {
   ListNode* head = new ListNode(1);     head->next
= new ListNode(2);     head->next->next = new
ListNode(3);     head->next->next->next = new
ListNode(4);     head->next->next->next->next = new
ListNode(5);
   ListNode* middle = middleNode(head);
```

```
    printList(middle);
return 0;}
```

OUTPUT: