23/02/24

Given an array of integers, find the smallest missing positive integer.

```
#include <stdio.h>
void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
int firstMissingPositive(int nums[], int numsSize) {
    for (int i = 0; i < numsSize; i++) {
        while (nums[i] > 0 && nums[i] <= numsSize && nums[nums[i] - 1] != nums[i]) {
            swap(&nums[nums[i] - 1], &nums[i]);
        }
    }
    for (int i = 0; i < numsSize; i++) {
        if (nums[i] != i + 1) {
            return i + 1;
        }
    }
    return numsSize + 1;
}
int main() {
    int nums[100];
    int numsSize;
    printf("Enter the size of the array: ");
    scanf("%d", &numsSize);
     printf("Enter the elements of the array:\n");
    for (int i = 0; i < numsSize; i++) {
        scanf("%d", &nums[i]);
    }
    int result = firstMissingPositive(nums, numsSize);
    printf("The smallest missing positive integer is: %d\n", result);
    return 0;
}
```

OUTPUT:

```
Enter the size of the array: 3
Enter the elements of the array:
-3
```

```
-6
-9
The smallest missing positive integer is: 1
```

Given an array of integers, find the two elements that have the maximum product.

```c
#include <stdio.h>

void findMaxProduct(int arr[], int n) {
    if (n < 2) {
        printf("Array must contain at least two elements.\n");
        return;
    }

    int max1 = arr[0], max2 = arr[1];
    int min1 = arr[0], min2 = arr[1];

    for (int i = 2; i < n; i++) {
        if (arr[i] > max1) {
            max2 = max1;
            max1 = arr[i];
        } else if (arr[i] > max2) {
            max2 = arr[i];
        }

        if (arr[i] < min1) {
            min2 = min1;
            min1 = arr[i];
        } else if (arr[i] < min2) {
            min2 = arr[i];
        }
    }

    if ((max1 * max2) > (min1 * min2)) {
        printf("Two elements with maximum product: %d and %d\n", max1, max2);
    } else {
        printf("Two elements with maximum product: %d and %d\n", min1, min2);
    }
}

int main() {
    int arr[] = {1, 4, 3, 6, 7, 0};
    int n = sizeof(arr) / sizeof(arr[0]);
    findMaxProduct(arr, n);
    return 0;
}
```

OUTPUT:

```
Two elements with maximum product: 7 and 6
```

Given an array of integers, find the subarray with the maximum product.

```
#include <stdio.h>
int max(int a, int b) {
    return (a > b) ? a : b;
}
int min(int a, int b) {
    return (a < b) ? a : b;
}
void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
int maxProductSubarray(int arr[], int n) {
    int max_product = arr[0];
    int min_product = arr[0];
    int max_result = arr[0];
    for (int i = 1; i < n; i++) {
        if (arr[i] < 0)
            swap(&max_product, &min_product);
        max_product = max(arr[i], max_product * arr[i]);
        min_product = min(arr[i], min_product * arr[i]);
        max_result = max(max_result, max_product);
    }
    return max_result;
}
int main() {
    int arr[] = {2, 3, -2, 4};
    int n = sizeof(arr) / sizeof(arr[0]);
    int result = maxProductSubarray(arr, n);
    printf("Maximum product subarray is: %d\n", result);
    return 0;
}
```

OUTPUT:

```
Maximum product subarray is: 6
```

Given an array of integers, find the longest subarray with the given sum.

```
#include <stdio.h>
void longestSubarrayWithSum(int arr[], int n, int targetSum) {
    int start = 0;
    int maxLength = 0;
    int currentSum = 0;
    for (int end = 0; end < n; end++) {
        currentSum += arr[end];
        while (currentSum > targetSum) {
            currentSum -= arr[start];
            start++;
        }
        maxLength = (end - start + 1 > maxLength) ? (end - start + 1) : maxLength;
    }
    printf("Length of the longest subarray with sum %d is: %d\n", targetSum, maxLength);
}
int main() {
    int arr[] = {5, 6, -5, 5, 3, 5, 3, -2, 0};
    int n = sizeof(arr) / sizeof(arr[0]);
    int targetSum = 8;
    longestSubarrayWithSum(arr, n, targetSum);
    return 0;
}
```

OUTPUT:

```
Length of the longest subarray with sum 8 is: 4
```

ANALYTICAL

Program to right rotate an array

```
#include <stdio.h>
#define SIZE 10
void printArray(int arr[]);
void rotateByOne(int arr[]);
int main()
{
    int i, N;
    int arr[SIZE];
printf("Enter 10 elements array: ");
    for(i=0; i<SIZE; i++)
    {
        scanf("%d", &arr[i]);
    }
    printf("Enter number of times to right rotate: ");
    scanf("%d", &N);
    N = N % SIZE;
    printf("Array before rotationn");
    printArray(arr);
    for(i=1; i<=N; i++)
    {
        rotateByOne(arr);
    }
    printf("\n\nArray after rotation\n");
    printArray(arr);
return 0;
}
void rotateByOne(int arr[])
{
    int i, last;
    last = arr[SIZE - 1];

    for(i=SIZE-1; i>0; i--)
    {
        arr[i] = arr[i - 1];
    }
    arr[0] = last;
}
void printArray(int arr[])
{
    int i;
    for(i=0; i<SIZE; i++)
    {
        printf("%d ", arr[i]);
    }
}
```

OUTPUT:

```
Enter 10 elements array: 56
64
87
58
84
98
37
472
9
9
Enter number of times to right rotate: 5
Array before rotation 56 64 87 58 84 98 37 47 9 9

Array after rotation
98 37 47 9 9 56 64 87 58 84
```

Find the longest common prefix among an array of strings

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
char* longestCommonPrefix(char** strs, int strsSize) {
    if (strsSize == 0) return "";
    int prefixLen = strlen(strs[0]);
    for (int i = 0; i < prefixLen; i++) {
        for (int j = 1; j < strsSize; j++) {
            if (strs[j][i] != strs[0][i] || strs[j][i] == '\0') {
                prefixLen = i;
                break;
            }
        }
    }
    char* prefix = (char*)malloc((prefixLen + 1) * sizeof(char));
    if (prefix == NULL) {
        fprintf(stderr, "Memory allocation failed\n");
        exit(1);
    }
    strncpy(prefix, strs[0], prefixLen);
    prefix[prefixLen] = '\0'; // Null-terminate the string
return prefix;
}
int main() {
    char* arr[] = {"flower", "flow", "flight"};
    int n = sizeof(arr) / sizeof(arr[0]);
    char* result = longestCommonPrefix(arr, n);
    printf("Longest common prefix: %s\n", result);
    free(result);
    return 0;
}
```

OUTPUT:

```
Longest common prefix: fl
```

Finding all distinct combinations of length k from a given array of integers

```
#include <stdio.h>
#include <stdlib.h>
void generateCombinations(int arr[], int n, int k, int start, int* buffer, int index) {
    if (index == k) {
        printf("(");
        for (int i = 0; i < k; i++) {
            printf("%d", buffer[i]);
            if (i != k - 1)
                printf(", ");
        }
        printf(")\n");
        return;
    }
    for (int i = start; i < n; i++) {
        buffer[index] = arr[i];
        generateCombinations(arr, n, k, i + 1, buffer, index + 1);
    }
}
int main() {
    int arr[] = {1, 2, 3, 4};
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 2;
    int* buffer = (int*)malloc(k * sizeof(int));
    if (buffer == NULL) {
        fprintf(stderr, "Memory allocation failed\n");
        return 1;
    }
    generateCombinations(arr, n, k, 0, buffer, 0);
    free(buffer);
    return 0;
}
```

OUTPUT:

```
(1, 2)
(1, 3)
(1, 4)
(2, 3)
(2, 4)
(3, 4)
```

implement a function to remove duplicates from a sorted array in C

```
#include <stdio.h>
int removeDuplicates(int arr[], int n) {
    if (n == 0 || n == 1)
        return n;
        int index = 0;
    for (int i = 1; i < n; i++) {
        if (arr[i] != arr[index]) {
            index++;
            arr[index] = arr[i];
        }
    }
    return index + 1;
}
int main() {
    int arr[] = {1, 1, 2, 2, 2, 3, 4, 4, 5};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("Original array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    int newLength = removeDuplicates(arr, n);
    printf("Array after removing duplicates: ");
    for (int i = 0; i < newLength; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    return 0;
}
```

OUTPUT:

```
Original array: 1 1 2 2 2 3 4 4 5
Array after removing duplicates: 1 2 3 4 5
```

implement a function to return the index of the first occurrence of a target value in a sorted array using binary search.

```
#include <stdio.h>
int firstOccurrence(int arr[], int n, int target) {
    int left = 0;
    int right = n - 1;
    int result = -1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == target) {
            result = mid;
            right = mid - 1;
        } else if (arr[mid] < target) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    return result;
}
int main() {
    int arr[] = {1, 2, 2, 2, 3, 4, 4, 5};
    int n = sizeof(arr) / sizeof(arr[0]);
    int target = 2;
    int index = firstOccurrence(arr, n, target);
    if (index != -1) {
        printf("The index of the first occurrence of %d is: %d\n", target, index);
    } else {
        printf("%d is not found in the array.\n", target);
    }
    return 0;
}
```

OUTPUT:

```
The index of the first occurrence of 2 is: 1
```