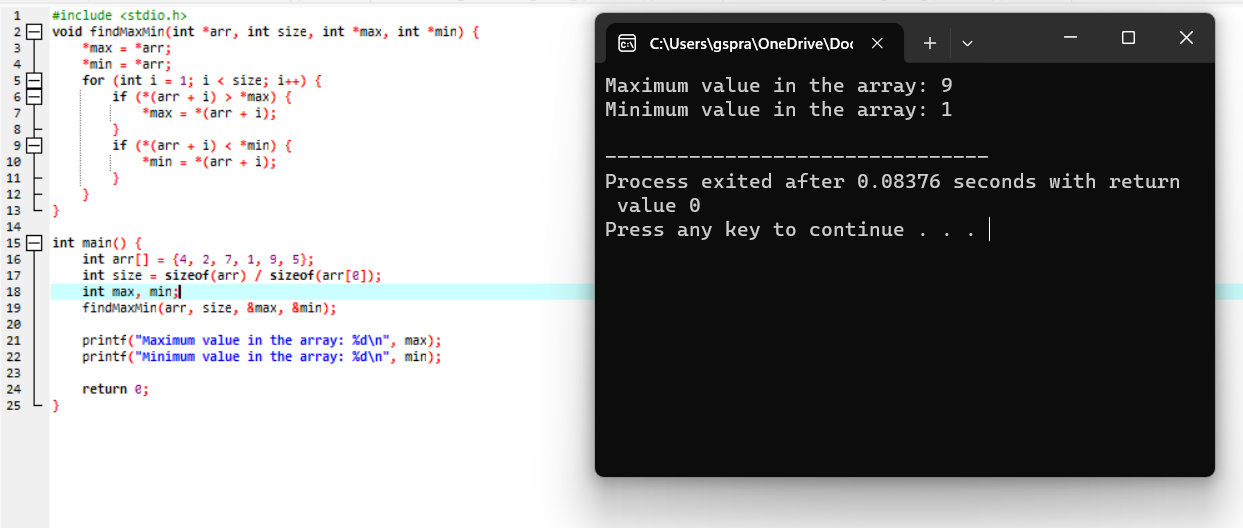
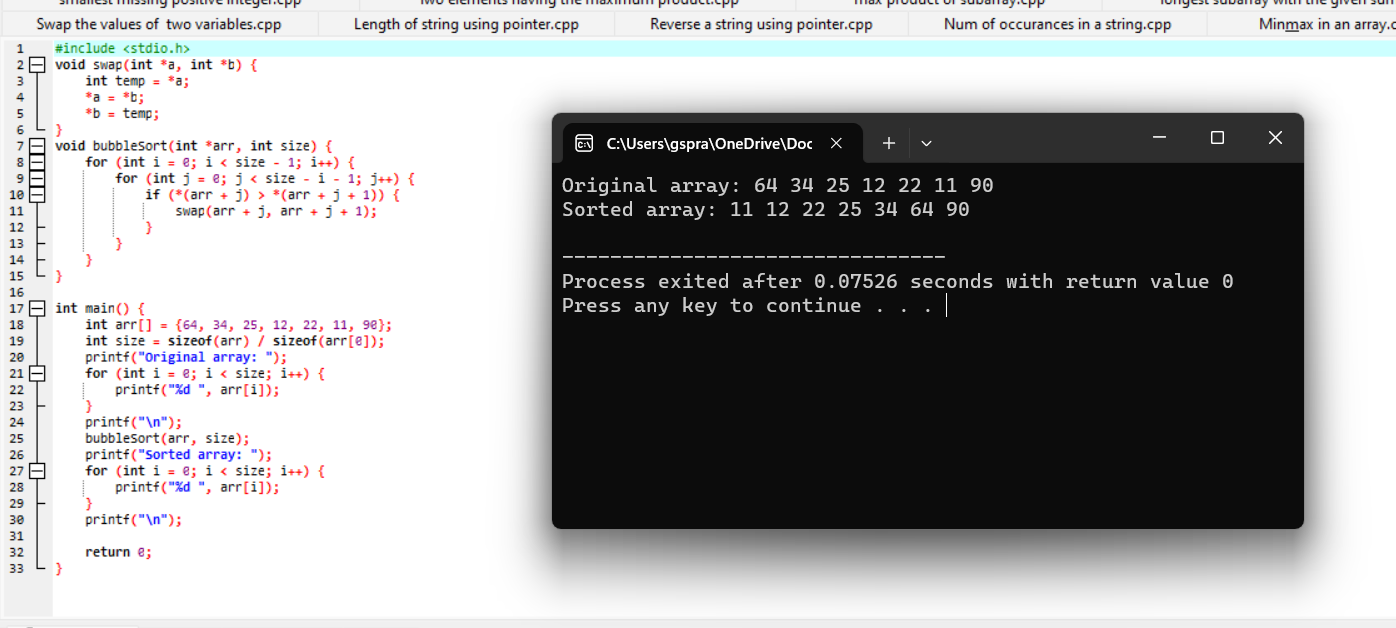
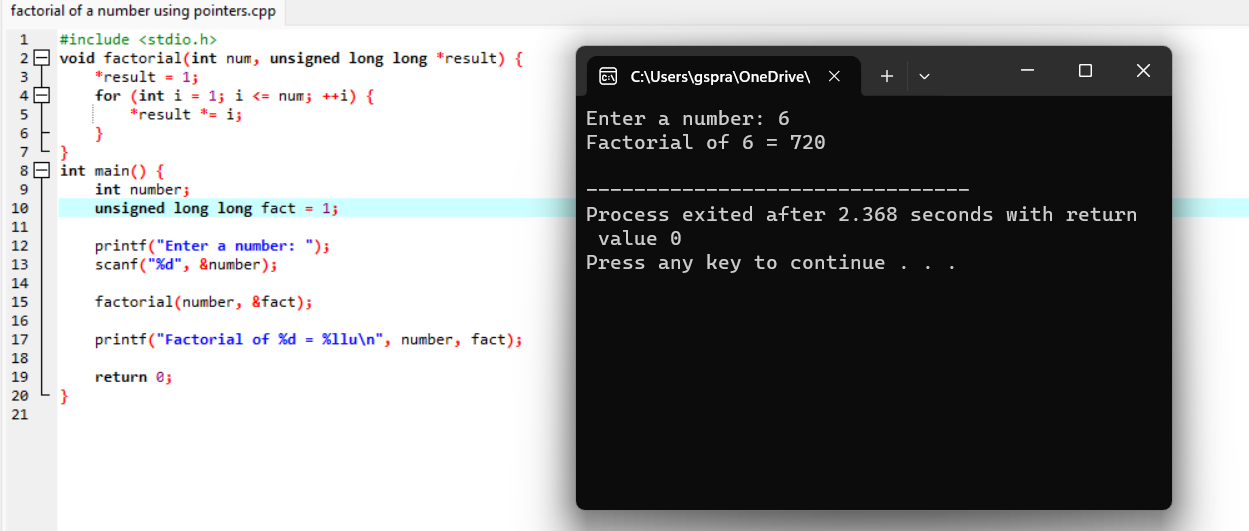
**20.Min & max in an array.**



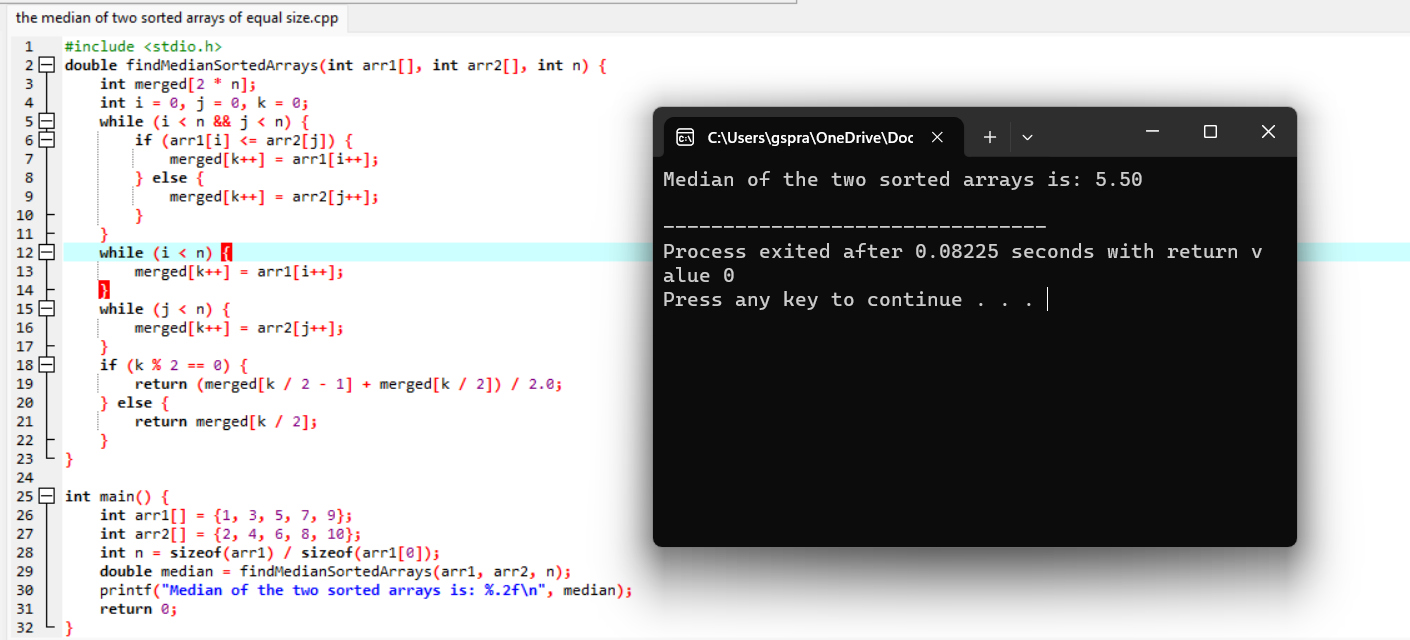
**21.Sort an array of integers.**



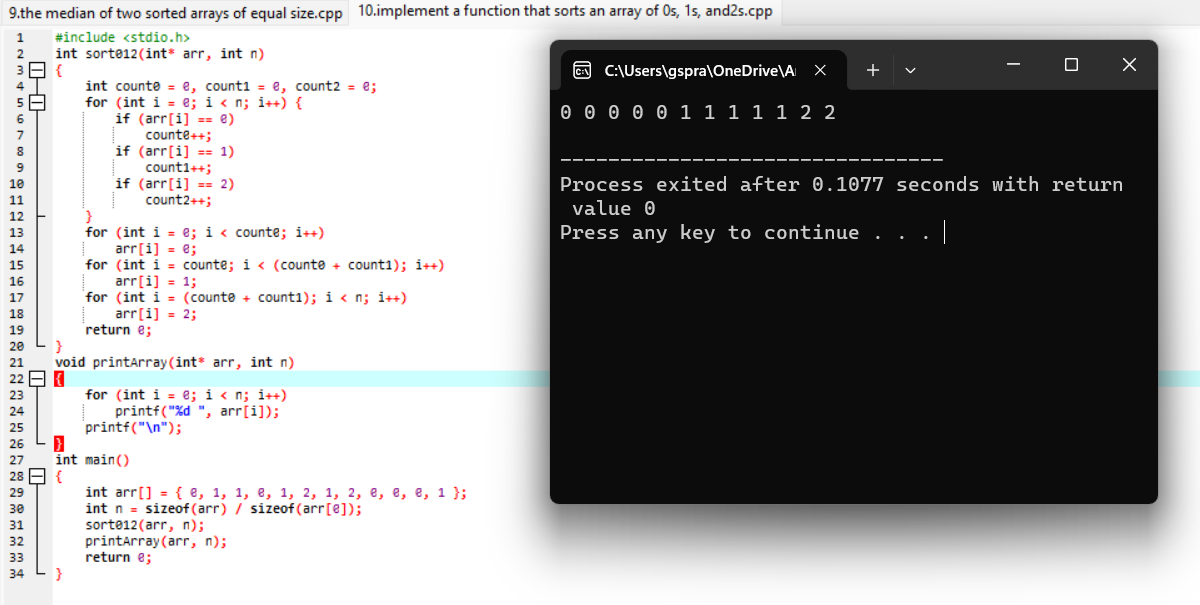
**22. Factorial of a number**.

**Analytical programs:**

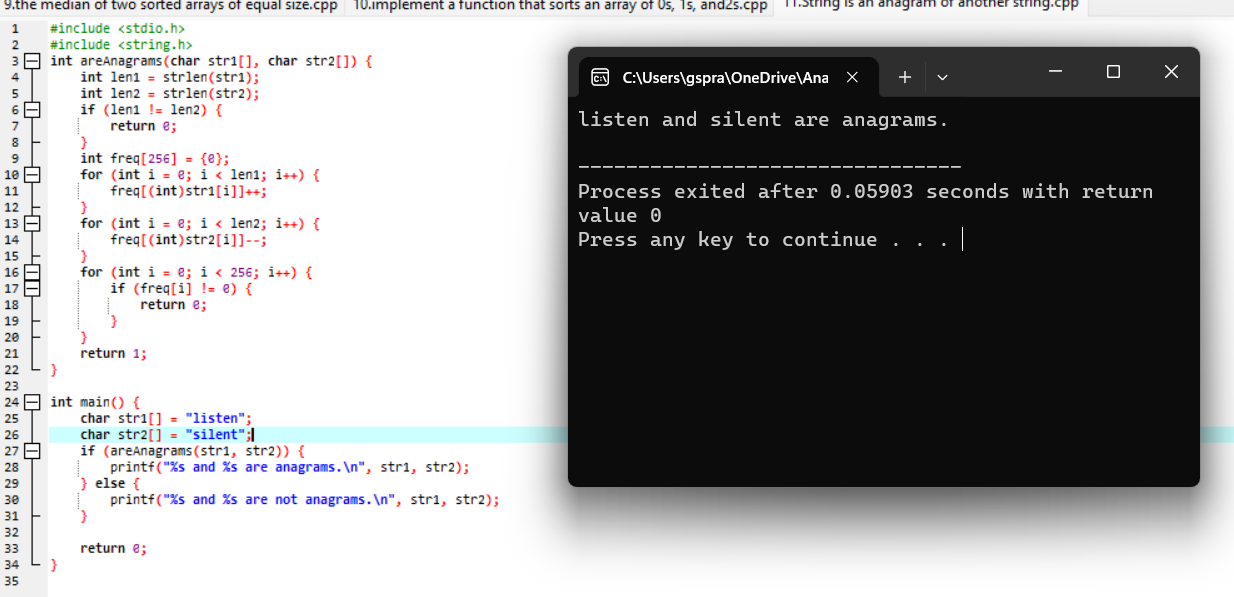
**9. The median of two sorted arrays of equal size:**

****

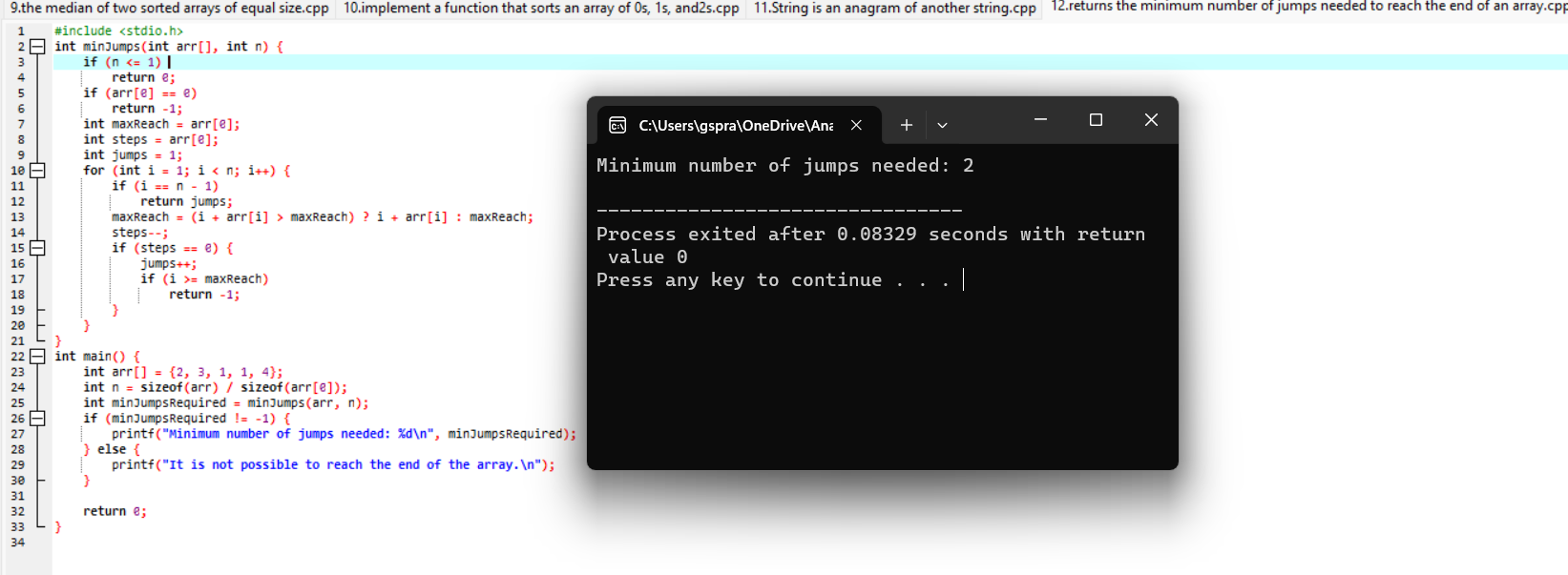
**10.** **Implement a function that sorts an array of 0s, 1s, and 2s**:

****

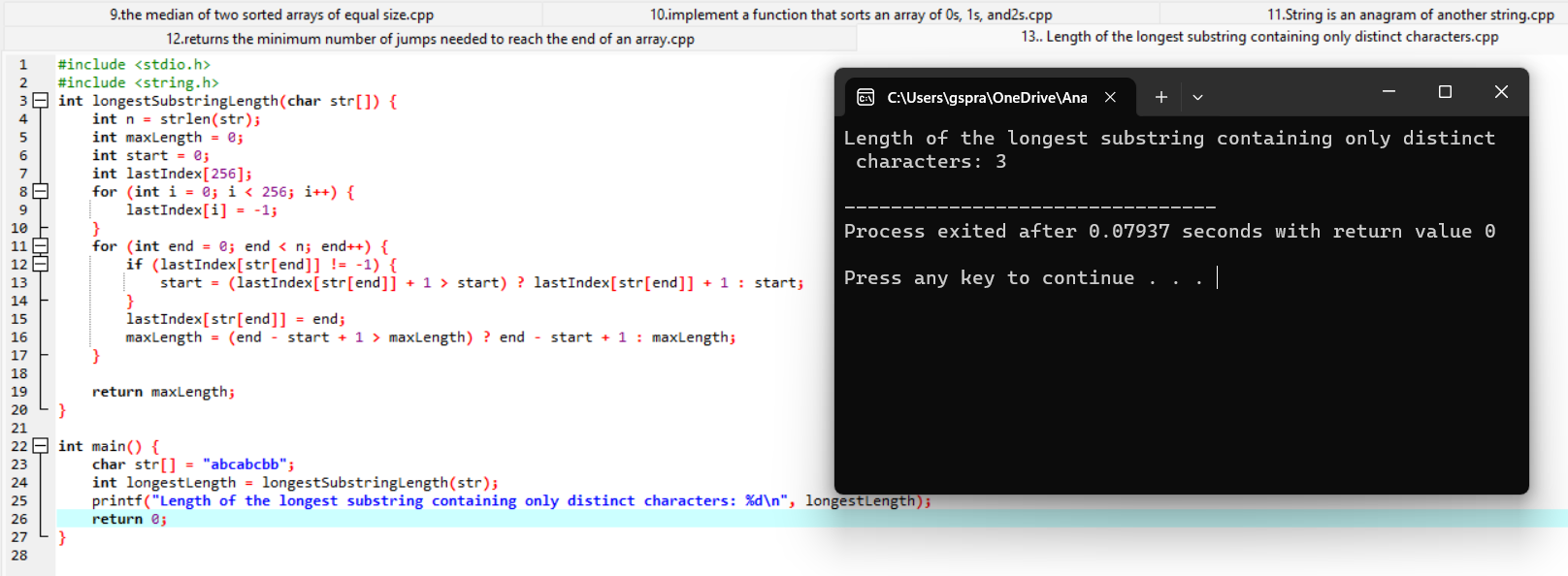
**11.** **String is an anagram of another string:**

****

**12.** **Returns the minimum number of jumps needed to reach the end of an array:**

****

**13. Length of the longest substring containing only distinct characters:**

****

**14. Finds the kth largest element in an unsorted array:**

#include <stdio.h>

void swap(int\* a, int\* b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int partition(int arr[], int low, int high) {

int pivot = arr[high];

int i = low - 1;

for (int j = low; j < high; j++) {

if (arr[j] <= pivot) {

i++;

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i + 1], &arr[high]);

return i + 1;

}

int quickSelect(int arr[], int low, int high, int k) {

if (low <= high) {

int pivotIndex = partition(arr, low, high);

if (pivotIndex == k) {

return arr[pivotIndex];

} else if (pivotIndex < k) {

return quickSelect(arr, pivotIndex + 1, high, k);

} else {

return quickSelect(arr, low, pivotIndex - 1, k);

}

}

return -1;

}

int findKthLargest(int arr[], int size, int k) {

if (k > 0 && k <= size) {

return quickSelect(arr, 0, size - 1, size - k);

}

return -1;

}

int main() {

int arr[] = {3, 2, 1, 5, 6, 4};

int size = sizeof(arr) / sizeof(arr[0]);

int k = 2;

int kthLargest = findKthLargest(arr, size, k);

if (kthLargest != -1) {

printf("The %dth largest element is: %d\n", k, kthLargest);

} else {

printf("Invalid value of k.\n");

}

return 0;

}

**Output:**

The 2th largest element is: 5

**15.** **Maximum sum subarray within a circular array:**

#include <stdio.h>

int max(int a, int b) {

return (a > b) ? a : b;

}

int kadane(int arr[], int size) {

int maxEndingHere = arr[0];

int maxSoFar = arr[0];

for (int i = 1; i < size; i++) {

maxEndingHere = max(arr[i], maxEndingHere + arr[i]);

maxSoFar = max(maxSoFar, maxEndingHere);

}

return maxSoFar;

}

int maxCircularSum(int arr[], int size) {

int maxKadane = kadane(arr, size);

int totalSum = 0;

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

totalSum += arr[i];

arr[i] = -arr[i];

}

int minKadane = kadane(arr, size);

int maxCircular = totalSum + minKadane;

if (maxKadane <= 0) {

return maxKadane;

}

return max(maxKadane, maxCircular);

}

int main() {

int arr[] = {8, -4, 3, -5, 4};

int size = sizeof(arr) / sizeof(arr[0]);

int maxCircularSumArray = maxCircularSum(arr, size);

printf("Maximum sum circular subarray: %d\n", maxCircularSumArray);

return 0;

}

**Output:**

Maximum sum circular subarray: 12

**16. Longest palindromic substring in a given string:**

#include <stdio.h>

#include <string.h>

int isPalindrome(char \*str, int start, int end) {

while (start < end) {

if (str[start] != str[end])

return 0;

start++;

end--;

}

return 1;

}

void longestPalindromicSubstring(char \*str) {

int n = strlen(str);

int maxLength = 1;

int start = 0;

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

for (int j = 0; i - j >= 0 && i + j < n; j++) {

if (str[i - j] != str[i + j])

break;

int length = 2 \* j + 1;

if (length > maxLength) {

maxLength = length;

start = i - j;

}

}

for (int j = 0; i - j >= 0 && i + 1 + j < n; j++) {

if (str[i - j] != str[i + 1 + j])

break;

int length = 2 \* j + 2;

if (length > maxLength) {

maxLength = length;

start = i - j;

}

}

}

printf("Longest palindromic substring is: ");

for (int i = start; i < start + maxLength; i++)

printf("%c", str[i]);

printf("\n");

}

int main() {

char str[100];

printf("Enter a string: ");

scanf("%s", str);

longestPalindromicSubstring(str);

return 0;

}

**Output:**

Enter a string: saireddy

Longest palindromic substring is: dd

**17.** **Number of subarrays with sum less than a given value.**

#include <stdio.h>

int countSubarraysWithSumLessThan(int arr[], int n, int target) {

int count = 0;

int sum = 0;

int start = 0;

for (int end = 0; end < n; end++) {

sum += arr[end];

while (sum >= target && start <= end) {

sum -= arr[start];

start++;

}

count += end - start + 1;

}

count -= n;

return count;

}

int main() {

int arr[] = {1, 4, 2, 3};

int n = sizeof(arr) / sizeof(arr[0]);

int target = 6;

int numSubarrays = countSubarraysWithSumLessThan(arr, n, target);

printf("Number of subarrays with sum less than %d is: %d\n", target, numSubarrays);

return 0;

}

**Output:**

Number of subarrays with sum less than 6 is: 2

**18.** **Count of distinct substrings of a given string**.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_LEN 100

struct TrieNode {

struct TrieNode\* children[26];

int count;

};

struct TrieNode\* createNode() {

struct TrieNode\* newNode = (struct TrieNode\*)malloc(sizeof(struct TrieNode));

newNode->count = 0;

for (int i = 0; i < 26; i++) {

newNode->children[i] = NULL;

}

return newNode;

}

void insert(struct TrieNode\* root, char\* str) {

struct TrieNode\* current = root;

int len = strlen(str);

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

int index = str[i] - 'a';

if (current->children[index] == NULL) {

current->children[index] = createNode();

}

current = current->children[index];

}

current->count++;

}

int countDistinctSubstrings(char\* str) {

int n = strlen(str);

struct TrieNode\* root = createNode();

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

for (int len = 1; len <= n - i; len++) {

char\* substr = (char\*)malloc((len + 1) \* sizeof(char));

strncpy(substr, &str[i], len);

substr[len] = '\0';

insert(root, substr);

free(substr);

}

}

int distinctCount = 0;

for (int i = 0; i < 26; i++) {

if (root->children[i] != NULL) {

distinctCount += root->children[i]->count;

}

}

return distinctCount;

}

int main() {

char str[MAX\_LEN];

printf("Enter a string: ");

fgets(str, sizeof(str), stdin);

str[strcspn(str, "\n")] = '\0';

int distinctSubstrings = countDistinctSubstrings(str);

printf("Count of distinct substrings: %d\n", distinctSubstrings);

return 0;

}

**Output:**

Enter a string: secret

Count of distinct substrings: 6

**19.** **Minimum window substring**  **:**

#include <stdio.h>

#include <string.h>

#include <limits.h>

#define MAX\_CHARS 256

int allCharactersFound(int arr[]) {

for (int i = 0; i < MAX\_CHARS; i++) {

if (arr[i] > 0) {

return 0;

}

}

return 1;

}

char\* minWindowSubstring(char\* str, char\* pattern) {

int patternCount[MAX\_CHARS] = {0};

int strCount[MAX\_CHARS] = {0};

int patternLength = strlen(pattern);

int strLength = strlen(str);

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

patternCount[pattern[i]]++;

}

int minWindowSize = INT\_MAX;

char\* minWindowStart = NULL;

int left = 0, right = 0;

int count = 0;

int startPos = -1;

while (right < strLength) {

if (patternCount[str[right]] > 0) {

strCount[str[right]]++;

if (strCount[str[right]] <= patternCount[str[right]]) {

count++;

}

}

if (count == patternLength) {

while (patternCount[str[left]] == 0 || strCount[str[left]] > patternCount[str[left]]) {

if (strCount[str[left]] > patternCount[str[left]]) {

strCount[str[left]]--;

}

left++;

}

if (right - left + 1 < minWindowSize) {

minWindowSize = right - left + 1;

minWindowStart = &str[left];

startPos = left;

}

}

right++;

}

if (startPos != -1) {

minWindowStart[minWindowSize] = '\0';

}

return minWindowStart;

}

int main() {

char str[] = "ADOBECODEBANC";

char pattern[] = "ABC";

char\* minWindow = minWindowSubstring(str, pattern);

if (minWindow != NULL) {

printf("Minimum window substring containing all characters of pattern: %s\n", minWindow);

} else {

printf("No window found\n");

}

return 0;

}

**Output:**

Minimum window substring containing all characters of pattern: BANC

**20.** **Length of the longest contiguous increasing subarray:**

#include <stdio.h>

int longestIncreasingSubarray(int arr[], int size) {

if (size == 0) {

return 0;

}

int maxLength = 1;

int currentLength = 1;

for (int i = 1; i < size; i++) {

if (arr[i] > arr[i - 1]) {

currentLength++;

} else {

if (currentLength > maxLength) {

maxLength = currentLength;

}

currentLength = 1;

}

}

if (currentLength > maxLength) {

maxLength = currentLength;

}

return maxLength;

}

int main() {

int arr[] = {1, 2, 3, 2, 5, 7, 8, 9, 10};

int size = sizeof(arr) / sizeof(arr[0]);

int result = longestIncreasingSubarray(arr, size);

printf("Length of the longest contiguous increasing subarray: %d\n", result);

return 0;

}

**Output:**

Length of the longest contiguous increasing subarray: 6