**Roll No…………….. Total No. of Pages:……**

**ST-1 (SET-I)**

**6th SEMESTER 2023-24**

**CS192- Advanced Data Structures**

**Time allowed: 90 Minutes Max. Marks: 40**

**General Instructions:**

* **Follow the instructions given in each section.**
* **Make sure that you attempt the questions in order.**

**SECTION-A (10\*1 mark=10 marks)**

***(All questions are compulsory)***

1. The prime sieve algorithm was developed by \_\_\_\_\_\_\_\_\_\_\_.
   1. **Eratosthenes**
   2. Euclid
   3. Fibonacci
   4. Gauss
2. In mathematics, a vector represents \_\_\_\_\_\_\_\_\_\_\_.
   1. A scalar quantity
   2. A 2D shape
   3. **A quantity with both magnitude and direction**
   4. A mathematical constant
3. Binary search can be used to find:
   1. The sum of all elements in an array
   2. The average of all elements in an array
   3. **The median of an array**
   4. The mode of an array
4. What is the worst-case time complexity of binary search?
   1. O(1)
   2. **O(log n)**
   3. O(n)
   4. O(n^2)
5. What is the time complexity of the Knuth-Morris-Pratt (KMP) algorithm for pattern searching in a text of length N and pattern of length M?
   1. O(N)
   2. O(M)
   3. **O(N+M)**
   4. O(N\*M)
6. The KMP algorithm is based on which principle?
   1. Divide and Conquer
   2. Greedy approach
   3. Sliding Window
   4. **Prefix-suffix matching**
7. What is the advantage of using the sliding window technique?
   1. **It reduces the time complexity of the algorithm.**
   2. It guarantees an optimal solution for any problem.
   3. It simplifies the implementation of prefix arrays.
   4. It eliminates the need for auxiliary data structures.
8. Which algorithm uses the sliding window technique to find the longest subarray with a given sum?
   1. Kadane's algorithm
   2. Binary search algorithm
   3. **Two-pointer algorithm**
   4. Prefix sum algorithm
9. Which algorithm uses the concept of a sliding window to solve substring problems efficiently?
   1. BFS Algorithm
   2. DFS Algorithm
   3. **Two Pointer Algorithm**
   4. Quick Sort Algorithm
10. How can we check if two strings are anagrams using the frequency array approach?
    1. Compare their lengths
    2. Sort both strings and compare them
    3. **Count occurrences of characters in both strings and compare the frequency arrays**
    4. Use the two-pointer technique to compare characters

**SECTION-B (5\*2 mark=10 marks)**

***(All questions are compulsory)***

**SECTION-C(Coding Question) (2x5 marks=5 marks)**

Q16) Given an array of integers, find the length of the longest subarray that forms a mountain pattern.

Note: A mountain pattern array is a sequence of numbers that exhibits a specific pattern resembling a mountain.

In this pattern, the elements in the array first increase and then decrease. The increasing part of the array represents the ascending slope of the mountain, while the decreasing part represents the descending slope.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | {2, 3, 4, 5, 2, 1, 0} | {12, 13, 11, 41} | {2, 3, 4, 5} |
| **Output** | Length of the longest mountain subarray: 7 | Length of the longest mountain subarray: 3 | Length of the longest mountain subarray: 0 |

Solution :

**#include <stdio.h>**

**// Function to find the length of the longest mountain subarray**

**int findLongestMountain(int arr[], int n) {**

**int maxLength = 0; // Variable to store the length of the longest mountain subarray**

**// Iterate through the array**

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

**int length = 1; // Initialize the length of the current mountain subarray**

**// Check if the current element is the peak of a mountain**

**if (arr[i] > arr[i - 1] && arr[i] > arr[i + 1]) {**

**// Traverse the left side of the peak**

**int j = i;**

**while (j > 0 && arr[j] > arr[j - 1]) {**

**length++;**

**j--;**

**}**

**// Traverse the right side of the peak**

**int k = i;**

**while (k < n - 1 && arr[k] > arr[k + 1]) {**

**length++;**

**k++;**

**}**

**// Update the maximum length if the current mountain subarray is longer**

**if (length > maxLength) {**

**maxLength = length;**

**}**

**}**

**}**

**return maxLength;**

**}**

**int main() {**

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

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

**int longestMountainLength = findLongestMountain(arr, n);**

**printf("Length of the longest mountain subarray: %d\n", longestMountainLength);**

**return 0;**

**}**

Q17) Given an array of integers, find the most frequent element using a frequency array and print the result.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | {1, 2, 3, 4, 2, 3, 1, 2, 4, 4} | {1, 2, 3, 1, 4} | {11,22,33,44,22,77,32,22} |
| **Output** | Most frequent element: 2 | Most frequent element: 1 | Most frequent element: 22 |

Solution :

**#include <stdio.h>**

**#define MAX\_SIZE 100**

**int findMostFrequent(int arr[], int size) {**

**int freq[MAX\_SIZE] = {0};**

**int i, maxFreq = -1, mostFrequent;**

**// Counting the frequency of each element**

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

**freq[arr[i]]++;**

**if (freq[arr[i]] > maxFreq) {**

**maxFreq = freq[arr[i]];**

**mostFrequent = arr[i];**

**}**

**}**

**return mostFrequent;**

**}**

**int main() {**

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

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

**int mostFrequent = findMostFrequent(arr, size);**

**printf("Most frequent element: %d\n", mostFrequent);**

**return 0;**

**}**

**SECTION-D (Coding Question)(1x10 mark=10 mark)**

Q18) You are given an array of words, and you need to find the longest common prefix among them.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | {"apple", "ape", "april"} | {"ant", "antique", "antra"} | {"banana", "basket", "bandra"} |
| **Output** | The longest common prefix is: ap | The longest common prefix is: ant | The longest common prefix is: ba |

Solution :

**#include <stdio.h>**

**#include <string.h>**

**// Function to find the longest common prefix**

**char\* longestCommonPrefix(char\* ar[], int n) {**

**// If size is 0, return empty string**

**if (n == 0)**

**return "";**

**// If size is 1 then just return that character**

**if (n == 1)**

**return ar[0];**

**// Sort the given array (lexicographically)**

**// so that the common prefix will be in the first and last string**

**for (int i = 0; i < n - 1; i++) {**

**for (int j = i + 1; j < n; j++) {**

**if (strcmp(ar[i], ar[j]) > 0) {**

**char\* temp = ar[i];**

**ar[i] = ar[j];**

**ar[j] = temp;**

**}**

**}**

**}**

**// Find the minimum length from first and last string**

**int en = strlen(ar[0]);**

**if (strlen(ar[n - 1]) < en)**

**en = strlen(ar[n - 1]);**

**// Now the common prefix in first and last string is the longest common prefix**

**int i = 0;**

**while (i < en && ar[0][i] == ar[n - 1][i])**

**i++;**

**char\* pre = (char\*)malloc((i + 1) \* sizeof(char));**

**strncpy(pre, ar[0], i);**

**pre[i] = '\0';**

**return pre;**

**}**

**int main() {**

**char\* ar[] = {"apple", "ape", "april"};**

**int n = sizeof(ar) / sizeof(ar[0]);**

**char\* longestCP= longestCommonPrefix(ar, n);**

**printf("The longest common prefix is: %s\n", longestCP);**

**free(longestCP);**

**return 0;**

**}**