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

**ST-1 (SET-II)**

**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. Which of the following is a valid application of vectors in the context of prime numbers?
   1. Finding the prime factors of a number
   2. Checking if a number is prime
   3. Generating prime numbers up to a given limit
   4. **All of the above**
2. The use of vectors in the prime sieve algorithm helps in \_\_\_\_\_\_\_\_\_\_\_.
   1. **Optimizing memory usage**
   2. Improving computational speed
   3. Ensuring accurate results
   4. Simplifying the algorithm
3. Which sorting algorithm has the best average-case time complexity for large datasets?
   1. Bubble Sort
   2. Selection Sort
   3. Heap Sort
   4. **Merge Sort**
4. Which sorting algorithm is not stable?
   1. Merge Sort
   2. Counting Sort
   3. **Quick Sort**
   4. Bubble Sort
5. The Z function algorithm finds all occurrences of the pattern in the text in:
   1. O(N)
   2. O(M)
   3. **O(N+M)**
   4. O(N\*M)
6. The KMP algorithm is considered more efficient than the Rabin-Karp algorithm for large texts and patterns.
   1. True
   2. **False**
7. What is the time complexity of checking if a string is a palindrome using the two-pointer technique?
   1. O(log n)
   2. **O(n)**
   3. O(n^2)
   4. O(1)
8. Which of the following algorithms uses a frequency array to count the occurrences of characters in a string?
   1. Selection Sort
   2. **Counting Sort**
   3. Bubble Sort
   4. Merge Sort
9. Which of the following is NOT a step in implementing the sliding window technique?
10. Initialize the two pointers to the start of the array.
11. Determine the window size based on the problem requirements.
12. Expand or contract the window based on a specific condition.
13. **Repeat the process until the window covers the entire array.**
14. How can the sliding window technique be applied to find the longest substring with unique characters?
15. By expanding the window until a repeated character is found and contracting the window until no repeated characters are present.
16. **By contracting the window until a repeated character is found and expanding the window until no repeated characters are present.**
17. By expanding the window until no repeated characters are present and contracting the window until a repeated character is found.
18. By contracting the window until no repeated characters are present and expanding the window until a repeated character is found.

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

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

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

Q16) We are given an array consisting of n elements. At each operation you can select any one element and increase rest of n-1 elements by 1.

You have to make all elements equal performing such operation as many times you wish. Find the minimum number of operations needed for this.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | arr = {1,2,3} | arr = {6,5,3,7} | arr = {1,1,1,2} |
| **Output** | Minimum Operation = 3 | Minimum Operation = 9 | Minimum Operation = 1 |

Solution :

**#include<bits/stdc++.h>**

**using namespace std;**

**// function for finding min operation**

**int minOp (int arr[], int n)**

**{**

**// find array sum**

**int sum = accumulate(arr,arr+n,0);**

**// find the smallest element from array**

**int small = \*min\_element(arr,arr+n);**

**// calculate min operation required**

**int minOperation = sum - (n \* small);**

**// return result**

**return minOperation;**

**}**

**int main()**

**{**

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

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

**cout << "Minimum Operation = " << minOp (arr, n);**

**return 0;**

**}**

Q17) Write a C program to check if two given strings are anagrams of each other.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | Enter the first string: silent  Enter the second string: listen | Enter the first string: raw  Enter the second string: war | Enter the first string: risk  Enter the second string: kick |
| **Output** | The two strings are anagrams. | The two strings are anagrams. | The two strings are not anagrams. |

Solution :

**#include <stdio.h>**

**#include <string.h>**

**// Function to check if two strings are anagrams**

**int areAnagrams(char str1[], char str2[]) {**

**int count[128] = {0};**

**// If the lengths of the strings are different, they cannot be anagrams**

**if (strlen(str1) != strlen(str2))**

**return 0;**

**for (int i = 0; i < strlen(str1); i++) {**

**count[str1[i]]++;**

**count[str2[i]]--;**

**}**

**for (int i = 0; i < 128; i++) {**

**if (count[i] != 0)**

**return 0;**

**}**

**return 1;**

**}**

**int main() {**

**char str1[100], str2[100];**

**printf("Enter the first string: ");**

**gets(str1);**

**printf("Enter the second string: ");**

**gets(str2);**

**if (areAnagrams(str1, str2))**

**printf("The two strings are anagrams.");**

**else**

**printf("The two strings are not anagrams.");**

**return 0;**

**}**

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

Q18) Implement the next permutation, which rearranges the list of numbers into Lexicographically next greater permutation of list of numbers.

If such arrangement is not possible, it must be rearranged to the lowest possible order i.e. sorted in an ascending order. You are given an list of numbers arr[ ] of size N.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | N = 6  arr = {1, 2, 3, 6, 5, 4} | N= 5  arr = {1, 4, 7, 9, 2} | N= 7  arr = {1, 5, 8, 9, 12, 32, 2} |
| **Output** | {1, 2, 4, 3, 5, 6} | {1, 4, 9, 2, 7} | 1 5 8 9 32 2 12 |

Solution :

**#include <bits/stdc++.h>**

**using namespace std;**

**class Solution {**

**public:**

**// Function to find the next lexicographically greater permutation of the input array.**

**vector<int> nextPermutation(int n, vector<int> arr) {**

**int index = -1;**

**// Step 1: Find the first element from the right that is smaller than its next element.**

**for (int i = n - 2; i >= 0; i--) {**

**if (arr[i] < arr[i + 1]) {**

**index = i;**

**break;**

**}**

**}**

**// Step 2: If no such element is found, the array is in decreasing order, so reverse the array and return.**

**if (index == -1) {**

**reverse(arr.begin(), arr.end());**

**return arr;**

**}**

**// Step 3: Find the smallest element from the right that is greater than the element at 'index'.**

**for (int i = n - 1; i > index; i--) {**

**if (arr[index] < arr[i]) {**

**swap(arr[i], arr[index]);**

**break;**

**}**

**}**

**// Step 4: Reverse the elements to the right of 'index'.**

**reverse(arr.begin() + index + 1, arr.end());**

**return arr; // Return the next lexicographically greater permutation.**

**}**

**};**

**int main() {**

**int N;**

**cin >> N; // Input the size of the array.**

**vector<int> arr(N); // Declare a vector to store the elements of the array.**

**// Input the elements of the array.**

**for (int i = 0; i < N; i++) {**

**cin >> arr[i];**

**}**

**Solution ob; // Create an object of the Solution class.**

**// Call the nextPermutation function to find the next lexicographically greater permutation.**

**vector<int> ans = ob.nextPermutation(N, arr);**

**// Output the elements of the next permutation.**

**for (int u : ans) {**

**cout << u << " ";**

**}**

**cout << "\n";**

**return 0;**

**}**