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

**ST-1 (SET-V)**

**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 is used to generate \_\_\_\_\_\_\_\_\_\_\_.
   1. **Prime numbers up to a given limit**
   2. Composite numbers up to a given limit
   3. All numbers up to a given limit
   4. Perfect numbers up to a given limit
2. The time complexity of the prime sieve algorithm is \_\_\_\_\_\_\_\_\_\_\_.
   1. **O(n)**
   2. O(n^2)
   3. O(log n)
3. The Rabin-Karp algorithm is based on which principle?
   1. Prefix-suffix matching
   2. **Hashing**
   3. Divide and Conquer
   4. Dynamic Programming
4. The Rabin-Karp algorithm is primarily used for:
   1. **Exact pattern matching**
   2. Pattern matching with wildcards
   3. Pattern searching in sorted arrays
   4. Longest common substring
5. Binary search can be used to solve which of the following problems efficiently?
   1. Sorting a linked list
   2. Finding the kth largest element in an unsorted array
   3. Finding the longest increasing subsequence in an array
   4. **All of the above**
6. The number of iterations required for binary search depends on:
   1. **The size of the array**
   2. The value being searched
   3. The position of the value in the array
   4. The search algorithm used
7. Given two strings, "ABCD" and "DCBA," what will be the output of their longest common subsequence length?
   1. 0
   2. 1
   3. 3
   4. **4**
8. The two-pointer technique can be applied to find a substring within a string in:
   1. O(1) time complexity
   2. **O(n) time complexity**
   3. O(log n) time complexity
   4. O(n^2) time complexity
9. What is the main advantage of using the sliding window technique?
10. **It reduces time complexity by eliminating the need for nested loops.**
11. It allows for constant-time access to individual array elements.
12. It simplifies array manipulation by maintaining a fixed-size window.
13. It guarantees sorted subarrays in ascending order.
14. In the sliding window technique, how is the window size determined?
15. **It is fixed and specified in the problem statement.**
16. It is calculated based on the size of the input array.
17. It expands dynamically as needed during the computation.
18. It is determined randomly during the execution of the program.

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

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

1. What is the correct output of the given code snippets?

#include <iostream>

#include <deque>

using namespace std;

int main()

{

deque<int> d;

d.push\_back(10);

d.push\_front(20);

d.pop\_back();

d.push\_back(40);

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

cout << d[i] << " ";

}

return 0;

}

* 1. 10 20 40
  2. 10 40
  3. **20 40**
  4. Syntax error

1. What will be the output of the following C++ code?

#include <iostream>

#include <string>

using namespace std;

int main() {

string str {"Steve jobs"};

unsigned long int found = str.find\_first\_of("aeiou");

while (found != string :: npos)

{

str[found] = '\*';

found = str.find\_first\_of("aeiou", found + 1);

}

cout << str << "\n";

return 0;

}

* 1. Steve
  2. jobs
  3. **St\*v\* j\*bs**
  4. St\*v\*

1. What is the correct output of given code snippets?

#include <iostream>

#include <vector>

using namespace std;

int main()

{

vector<int> v1;

v1.push\_back(1);

v1.push\_back(2);

v1.push\_back(3);

v1.push\_back(4);

for (int i = 0; i < v1.size(); i++)

cout << \*(v1 + i) << " ";

return 0;

}

* 1. 1 2 3 4
  2. Garbage value
  3. **Syntax error**
  4. Runtime error

1. Find the length of the longest increasing subsequence for the given sequence:

{-10, 24, -9, 35, -21, 55, -41, 76, 84}

* 1. 5
  2. 4
  3. 3
  4. **6**

1. Which of the following standard algorithms is not Dynamic Programming based?
   1. Bellman–Ford Algorithm for single source shortest path
   2. Floyd Warshall Algorithm for all pairs shortest paths
   3. 0-1 Knapsack problem
   4. **Prim's Minimum Spanning Tree**

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

Q16) Given a string S, the task is to find the bracket numbers.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | S = "(aa(bdc))p(dee)" | S = "(((()(" | S = “(sam(sung))” |
| **Output** | 1 2 2 1 3 3 | 1 2 3 4 4 5 | 1 2 2 1 |

Solution :

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

**using namespace std;**

**class Solution**

**{**

**public:**

**// Function to assign bracket numbers to each opening and closing parenthesis in the input string**

**vector<int> bracketNumbers(string S)**

**{**

**int a = 0; // Counter to keep track of the bracket numbers**

**vector<int> res; // Vector to store the result, i.e., the bracket numbers**

**stack<int> stack; // Stack to keep track of opening parenthesis positions**

**for (auto x : S)**

**{**

**if (x == '(')**

**{**

**stack.push(++a); // Increment the counter and push the current bracket number to the stack**

**res.push\_back(stack.top()); // Store the current bracket number as the result**

**}**

**if (x == ')')**

**{**

**res.push\_back(stack.top()); // For closing parenthesis, assign the top bracket number from the stack**

**stack.pop(); // Pop the bracket number from the stack as it has been used for the current closing parenthesis**

**}**

**}**

**return res; // Return the vector containing the bracket numbers**

**}**

**};**

**int main()**

**{**

**string s;**

**getline(cin, s); // Input the string containing parentheses**

**Solution ob;**

**vector<int> ans = ob.bracketNumbers(s); // Get the bracket numbers for the input string**

**for (auto i : ans)**

**cout << i << " "; // Print the bracket numbers separated by spaces**

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

**return 0;**

**}**

Q17) Given an array of integers, calculate the sum of elements in a given range using the prefix sum array and print the result.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | {1, 2, 3, 4, 5}  range= 1-3 | {21,31,4,5,7}  range= 2-4 | {-3,6,-7,8}  range= 0-2 |
| **Output** | Sum of elements from index 1 to 3: 9 | Sum of elements from index 2 to 4: 16 | Sum of elements from index 0 to 2: -4 |

Solution :

**#include <stdio.h>**

**#define MAX\_SIZE 100**

**int rangeSum(int prefix[], int left, int right) {**

**if (left == 0) {**

**return prefix[right];**

**} else {**

**return prefix[right] - prefix[left - 1];**

**}**

**}**

**int main() {**

**int arr[] = {-3,6,-7,8};**

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

**int prefix[MAX\_SIZE];**

**prefix[0] = arr[0];**

**// Calculating the prefix sum**

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

**prefix[i] = prefix[i - 1] + arr[i];**

**}**

**int left = 0; // Range start index**

**int right = 2; // Range end index**

**int sum = rangeSum(prefix, left, right);**

**printf("Sum of elements from index %d to %d: %d\n", left, right, sum);**

**return 0;**

**}**

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

Q18) Given an array containing only 0s, 1s, and 2s, sort the array in ascending order using the Dutch National Flag Algorithm. The algorithm should rearrange the elements in the array so that all 0s come before 1s, and all 1s come before 2s.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | [0, 1, 2, 0, 1, 2, 1, 0] | [1, 2, 0, 1, 0] | [0,1,2,1,2,0] |
| **Output** | [0, 0, 0, 1, 1, 1, 2, 2] | [0, 0, 1, 1, 2] | [0,0,1,1,2,2] |

Solution :

**#include <stdio.h>**

**// Dutch National Flag Algorithm**

**// This function takes an array 'arr' and its size 'n' as input.**

**// It sorts the array in-place in ascending order, where the elements are colors represented as integers (0, 1, or 2).**

**// The algorithm rearranges the elements so that all 0s come before 1s, and all 1s come before 2s.**

**void sortColors(int arr[], int n) {**

**int low = 0, mid = 0, high = n - 1;**

**int temp;**

**// The algorithm uses three pointers: 'low', 'mid', and 'high'.**

**// 'low' points to the boundary between 0s and 1s.**

**// 'mid' points to the boundary between 1s and unprocessed elements.**

**// 'high' points to the boundary between unprocessed elements and 2s.**

**while (mid <= high) {**

**switch (arr[mid]) {**

**case 0:{**

**// If the element at 'mid' is 0, swap it with the element at 'low'.**

**// Move 'low' and 'mid' one step forward.**

**temp = arr[low];**

**arr[low] = arr[mid];**

**arr[mid] = temp;**

**low++;**

**mid++;**

**break;**

**}**

**case 1:{**

**// If the element at 'mid' is 1, move 'mid' one step forward.**

**mid++;**

**break;**

**}**

**case 2:{**

**// If the element at 'mid' is 2, swap it with the element at 'high'.**

**// Move 'high' one step backward.**

**temp = arr[mid];**

**arr[mid] = arr[high];**

**arr[high] = temp;**

**high--;**

**break;**

**}**

**}**

**}**

**}**

**int main() {**

**// Sample array of colors represented as integers (0, 1, or 2)**

**int arr[] = {0, 1, 2, 0, 1, 2, 1, 0};**

**// Calculate the size of the array**

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

**// Print the unsorted array**

**printf("Unsorted array: ");**

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

**printf("%d ", arr[i]);**

**// Call the sortColors function to sort the array in-place**

**sortColors(arr, n);**

**// Print the sorted array**

**printf("\nSorted array: ");**

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

**printf("%d ", arr[i]);**

**return 0;**

**}**