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

**ST-4 (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. What is the classic example of a "Counting" type 1-Dimensional DP problem?
2. Fibonacci Sequence
3. Longest Common Subsequence
4. **Coin Change**
5. Knapsack Problem
6. In a grid DP problem, what does the "overlapping subproblems" property imply?
   1. Subproblems are entirely unrelated.
   2. **The same subproblems are solved multiple times.**
   3. Subproblems have no common elements.
   4. Subproblems always have unique solutions.
7. Which algorithm is used to solve the Fractional Knapsack problem optimally?
   1. **Greedy algorithm**
   2. Dynamic programming
   3. Backtracking
   4. Branch and bound
8. What is the primary advantage of using dynamic programming over a naive recursive approach?
   1. **Faster execution**
   2. Smaller code
   3. Easier implementation
   4. Higher memory efficiency
9. In Dynamic Programming on Trees, what does the term "bottom-up" approach refer to?
   1. **Starting from the leaves and moving up to the root**
   2. Starting from the root and moving down to the leaves
   3. Moving from one branch to another within the tree
   4. Traversing the tree level by level
10. Which greedy algorithm is used for task scheduling to minimize waiting time?
    1. Huffman coding
    2. Dijkstra's algorithm
    3. **Shortest Job First (SJF)**
    4. Kruskal's algorithm
11. How do you set the i-th bit of an integer x to 1?
    1. **x | (1 << i)**
    2. x & (1 << i)
    3. x ^ (1 << i)
    4. x >> i
12. If φ(10) is the Euler's Totient Function of 10, what is its value?
    1. 1
    2. 2
    3. **4**
    4. 5
13. How many onto (or surjective) functions are there from an n-element (n >= 2) set to a 2-element set?
    1. 2(2^n - 2)
    2. **2^n - 2**
    3. 2^n - 1
    4. 2^n
14. What is the possible number of reflexive relations on a set of 5 elements?
    1. 2^25
    2. **2^20**
    3. 2^15
    4. 2^10

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

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

11) What is the output of the below code: for n=5 and arr[] = {3, 10, 2, 1, 20};

int function(int arr[], int n)

{

int lis[n];

lis[0] = 1;

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

lis[i] = 1;

for (int j = 0; j < i; j++)

if (arr[i] > arr[j] && lis[i] < lis[j] + 1)

lis[i] = lis[j] + 1;

}

return \*max\_element(lis, lis + n);

}

a) 2

b) 10

c) 1

**d) 3**

12) The Partition Problem can be reduced to which well-known problem?

a) Knapsack Problem

b) Traveling Salesman Problem

**c) Subset Sum Problem**

d) Graph Coloring Problem

13) Define Rn to be the maximum amount earned by cutting a rod of length n meters into one or more pieces of integer length and selling them. For i>0, let p[i] denote the selling price of a rod whose length is i meters. Consider the array of prices:

p[1]=1, p[2]=5, p[3]=8, p[4]=9, p[5]=10, p[6]=17, p[7]=18

Which of the following statements is/are correct about R7?

**a) R7=18**

b) R7=19

**c) R7 is achieved by three different solutions**

d) R7 cannot be achieved by a solution consisting of three pieces

14) What is the return value of following function for arr[] = {9, 12, 2, 11, 2, 2, 10, 9, 12, 10, 9, 11, 2} and n is size of this array.

int fun(int arr[], int n)

{

int x = arr[0];

for (int i = 1; i < n; i++)

x = x ^ arr[i];

return x;

}

a) 0

**b) 9**

c) 12

d) 2

15) What is the output of the following C++ code?

#include <iostream>

#include <algorithm>

using namespace std;

int main() {

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

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

sort(arr, arr + n, greater<int>());

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

cout << arr[i] << " ";

}

return 0;

}

a) 1 2 3 5 6

**b) 6 5 3 2 1**

c) 1 3 2 5 6

d) 6 5 2 3 1

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

Q16) Evaluate the value of an arithmetic expression in Reverse Polish Notation (postfix notation).

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | ["2", "1", "+", "3", "\*"] | ["6", "3", "2", "4", "+","-","\*"] | ["6", "3","/"] |
| **Output** | 9 | -18 | 2 |

Solution :

**#include <iostream>**

**#include <stack>**

**#include <vector>**

**using namespace std;**

**// Function to evaluate Reverse Polish Notation (RPN) expression**

**int evalRPN(vector<string>& tokens) {**

**stack<int> st; // Initialize a stack to hold operands**

**// Iterate through each token in the expression**

**for (string token : tokens) {**

**if (token == "+" || token == "-" || token == "\*" || token == "/") {**

**// If the token is an operator, pop the top two elements from the stack**

**int num2 = st.top(); st.pop();**

**int num1 = st.top(); st.pop();**

**// Perform the operation based on the operator**

**if (token == "+") st.push(num1 + num2);**

**else if (token == "-") st.push(num1 - num2);**

**else if (token == "\*") st.push(num1 \* num2);**

**else if (token == "/") st.push(num1 / num2);**

**} else {**

**// If the token is an operand, convert it to an integer and push it onto the stack**

**st.push(stoi(token));**

**}**

**}**

**// The final result will be left on the top of the stack**

**return st.top();**

**}**

**int main() {**

**vector<string> tokens = {"6", "3", "2", "4", "+","-","\*"};**

**cout << "Result: " << evalRPN(tokens) << endl; // Evaluate the RPN expression and display the result**

**return 0;**

**}**

Q17) Write a CPP program to print the top view of binary tree. Top view of a binary tree is the set of nodes visible when the tree is viewed from the top.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | 10  / \  20 30  / \ / \  40 60 90 100 | 1  / \  2 3  \  4  \  5  \  6 | 10  / \  20 30 |
| **Output** | 40 20 10 30 100 | 2 1 3 6 | 20 10 30 |

Solution :

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

**using namespace std;**

**// Structure to represent a tree node**

**struct Node {**

**int data;**

**Node\* left;**

**Node\* right;**

**Node(int val) {**

**data = val;**

**left = right = NULL;**

**}**

**};**

**// Function to print the top view of a binary tree**

**void topView(Node\* root) {**

**if (!root)**

**return;**

**map<int, int> verticalMap; // Map to store vertical level and node data**

**queue<pair<Node\*, int>> q; // Queue for BFS traversal**

**q.push({root, 0});**

**while (!q.empty()) {**

**Node\* node = q.front().first;**

**int verticalLevel = q.front().second;**

**q.pop();**

**// Insert the node's data if not present in the map**

**if (verticalMap.find(verticalLevel) == verticalMap.end()) {**

**verticalMap[verticalLevel] = node->data;**

**}**

**// Push left child with decreased vertical level**

**if (node->left) {**

**q.push({node->left, verticalLevel - 1});**

**}**

**// Push right child with increased vertical level**

**if (node->right) {**

**q.push({node->right, verticalLevel + 1});**

**}**

**}**

**// Print the nodes in the top view**

**for (const auto& entry : verticalMap) {**

**cout << entry.second << " ";**

**}**

**}**

**int main() {**

**Node\* root = new Node(10);**

**root->left = new Node(20);**

**root->right = new Node(30);**

**root->left->right = new Node(60);**

**root->left->left = new Node(40);**

**root->right->left = new Node(90);**

**root->right->right = new Node(100);**

**cout << "Top view of the binary tree: ";**

**topView(root);**

**return 0;**

**}**

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

Q18) Write a C++ program to insert new element to MAX-Heap.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | heap={ 10, 5, 3, 2, 4 }  key=15 | heap={ 12,6,4,3,5 }  key=21 | heap={ 30,15,19 }  key=2 |
| **Output** | 15 5 10 2 4 3 | 21 6 12 3 5 4 | 30 15 19 2 |

Solution :

**#include <iostream>**

**using namespace std;**

**#define MAX 1000 // Max size of Heap**

**// Function to heapify ith node in a Heap**

**// of size n following a Bottom-up approach**

**void heapify(int arr[], int n, int i)**

**{**

**// Find parent**

**int parent = (i - 1) / 2;**

**if (arr[parent] > 0) {**

**// For Max-Heap**

**// If current node is greater than its parent**

**// Swap both of them and call heapify again**

**// for the parent**

**if (arr[i] > arr[parent]) {**

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

**// Recursively heapify the parent node**

**heapify(arr, n, parent);**

**}**

**}**

**}**

**// Function to insert a new node to the Heap**

**void insertNode(int arr[], int& n, int Key)**

**{**

**// Increase the size of Heap by 1**

**n = n + 1;**

**// Insert the element at end of Heap**

**arr[n - 1] = Key;**

**// Heapify the new node following a**

**// Bottom-up approach**

**heapify(arr, n, n - 1);**

**}**

**// A utility function to print array of size n**

**void printArray(int arr[], int n)**

**{**

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

**cout << arr[i] << " ";**

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

**}**

**int main()**

**{**

**// Array representation of Max-Heap**

**// 10**

**// / \**

**// 5 3**

**// / \**

**// 2 4**

**int arr[MAX] = { 10, 5, 3, 2, 4 };**

**int n = 5;**

**int key = 15;**

**insertNode(arr, n, key);**

**printArray(arr, n);**

**// Final Heap will be:**

**// 15**

**// / \**

**// 5 10**

**// / \ /**

**// 2 4 3**

**return 0;**

**}**