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

**ST-3 (SET-III)**

**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 average and worst-case time complexity of the pop() operation in a stack is:
   1. **O(1)**
   2. O(log n)
   3. O(n)
   4. O(n log n)
2. Which data structure is commonly used to implement undo functionality in applications?
   1. Queue
   2. **Stack**
   3. Linked List
   4. Hash Table
3. Which type of binary tree has the height balanced such that the difference between the heights of the left and right subtrees is at most one?
   1. Perfect Binary Tree
   2. Complete Binary Tree
   3. **AVL Tree**
   4. B-Tree
4. Balanced binary tree with n items allows the lookup of an item in \_\_\_\_ worst-case time.
   1. **O(log n)**
   2. O(nlog 2)
   3. O(n)
   4. O(1)
5. AVL tree is an example of a \_\_\_\_\_.
   1. Search tree
   2. Traversal tree
   3. Priority tree
   4. **Balanced tree**
6. The expression "((2+3)\*(5-4))/(2+3)" can be evaluated using which stack-based approach?
   1. Postfix evaluation
   2. **Infix evaluation**
   3. Prefix evaluation
   4. Parentheses balancing
7. What is the load factor of a hash table?
   1. The number of elements stored in the table.
   2. The size of the table.
   3. **The ratio of occupied slots to total slots in the table.**
   4. The number of collisions in the table.
8. Which of the following hash functions is most likely to cause clustering in a hash table?
   1. **h(k) = k % m**
   2. h(k) = floor(m \* (kA mod 1))
   3. h(k) = k
   4. h(k) = ((k / m) + k \* m) + k % m
9. While inserting a new element in a binary max-heap, if the new element violates the heap property, what action is taken?
   1. The new element is removed from the heap.
   2. The new element is placed at the root position.
   3. **The new element is swapped with its largest child until the heap property is satisfied.**
   4. The heap is restructured from scratch.
10. After performing a single left rotation during AVL tree deletion, the height balance of the affected nodes becomes:
11. **(0, 0)**
12. (-1, 1)
13. (1, 0)
14. (0, 1)

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

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

11) Consider the following pseudocode that uses a stack. What is output for input "letsfindc"?

declare a stack of characters

while ( there are more characters in the word to read )

{

read a character

push the character on the stack

}

while ( the stack is not empty )

{

pop a character off the stack

write the character to the screen

}

a) letsfindcletsfindc

**b) cdnifstel**

c) letsfindc

d) cdnifstelcdnifstel

12) What will be the output of the following code?

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* left;

Node\* right;

Node(int val) : data(val), left(nullptr), right(nullptr) {}

};

int sumOfNodes(Node\* root) {

if (root == nullptr) return 0;

return root->data + sumOfNodes(root->left) + sumOfNodes(root->right);

}

int main() {

Node\* root = new Node(1);

root->left = new Node(2);

root->right = new Node(3);

root->left->left = new Node(4);

root->left->right = new Node(5);

cout << sumOfNodes(root);

return 0;

}

a) 10

**b) 15**

c) 16

d) 20

13) bool isBST(Node\* root, int minValue, int maxValue) {

if (root == NULL)

return true;

if (root->data <= minValue || root->data >= maxValue)

return false;

return isBST(root->left, minValue, root->data) &&

isBST(root->right, root->data, maxValue);

}

int main() {

Node\* root = newNode(5);

root->left = newNode(3);

root->right = newNode(7);

root->left->left = newNode(2);

root->left->right = newNode(6);

cout << (isBST(root, INT\_MIN, INT\_MAX) ? "Yes" : "No") << endl;

return 0;

}

a) Yes

**b) No**

c) Compilation error

d) Runtime error

14) Consider the following AVL tree deletion pseudo-code:

AVL\_Delete(root, key):

// Deletion logic here

return new\_root

What will be the value of 'new\_root' if the key is not found in the AVL tree?

a) NULL

b) 0

c) -1

**d) root**

15) int inline hash1(int value){

return value%TABLE\_SIZE;

}

int inline hash2(int value){

return PRIME - (value%PRIME);

}

Above functions are used in which type of hashing?

a) Linear probing

**b) Double hashing**

c) Quadratic probing

d) Chaining

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

Q16) Given an array of integers, the task is to find the maximum absolute difference between the nearest left and the right smaller element of every element in the array.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | arr[] = {2, 1, 8} | arr[] = {2, 4, 8, 7, 7, 9, 3} | arr[] = {12, 8, 3, 1} |
| **Output** | 1 | 4 | 8 |

Solution :

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

**using namespace std;**

**// Function to fill left smaller element for every**

**// element of arr[0..n-1]. These values are filled**

**// in SE[0..n-1]**

**void leftSmaller(int arr[], int n, int SE[]) {**

**// Create an empty stack**

**stack<int> S;**

**// Traverse all array elements**

**// compute nearest smaller elements of every element**

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

**// Keep removing top element from S while the top**

**// element is greater than or equal to arr[i]**

**while (!S.empty() && S.top() >= arr[i])**

**S.pop();**

**// Store the smaller element of the current element**

**if (!S.empty())**

**SE[i] = S.top();**

**// If all elements in S were greater than arr[i]**

**else**

**SE[i] = 0;**

**// Push this element**

**S.push(arr[i]);**

**}**

**}**

**// Function returns the maximum difference between**

**// Left & right smaller elements**

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

**int LS[n]; // To store left smaller elements**

**// Find left smaller element of every element**

**leftSmaller(arr, n, LS);**

**// Find right smaller element of every element**

**// First reverse the array and do the same process**

**int RRS[n]; // To store right smaller elements in reverse array**

**reverse(arr, arr + n);**

**leftSmaller(arr, n, RRS);**

**// Find the maximum absolute difference between LS & RRS**

**// In the reversed array, right smaller for arr[i] is stored at RRS[n-i-1]**

**int result = -1;**

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

**result = max(result, abs(LS[i] - RRS[n - 1 - i]));**

**// Return the maximum difference between LS & RRS**

**return result;**

**}**

**int main() {**

**int arr[] = {12, 8, 3, 1};**

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

**cout << "Maximum diff : " << findMaxDiff(arr, n) << endl;**

**return 0;**

**}**

Q17) Write a function to determine if two trees are identical or not: Two trees are identical when they have the same data and the arrangement of data is also the same.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | 1 1  / \ / \  2 3 2 3  / /  4 4 | 1 9  / \ / \  2 3 8 13 | 1 1  / \ / \  2 3 2 3 |
| **Output** | Both trees are identical | Given trees are not identical. | Both trees are identical |

Solution :

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

**using namespace std;**

**/\* A binary tree node \*/**

**class Node**

**{**

**public:**

**int data;**

**Node\* left;**

**Node\* right;**

**Node(int d)**

**{**

**this->data = d;**

**this->left = NULL;**

**this->right = NULL;**

**}**

**};**

**void preorder(Node\* root,vector<int>&v)**

**{**

**if(root==NULL)**

**{**

**// v.push\_back(0);**

**return;**

**}**

**v.push\_back(root->data);**

**if(root->left)**

**preorder(root->left,v);**

**if(!root->left)**

**v.push\_back(0);**

**if(root->right)**

**preorder(root->right,v);**

**if(!root->right)**

**v.push\_back(0);**

**}**

**bool isIdentical(Node\* root1,Node\*root2)**

**{**

**vector<int>v,x;**

**preorder(root1,v);**

**preorder(root2,x);**

**// for(auto it:v)cout<<it<<" ";**

**// cout<<endl;**

**return v==x;**

**}**

**int main()**

**{**

**Node\* root1 = new Node(1);**

**root1->left = new Node(2);**

**root1->right = new Node(3);**

**root1->left->left = new Node(4);**

**root1->left->right = new Node(5);**

**Node\* root2 = new Node(1);**

**root2->left = new Node(2);**

**root2->right = new Node(3);**

**root2->left->left = new Node(4);**

**root2->left->right = new Node(5);**

**// Function call**

**if (isIdentical(root1, root2))**

**{**

**cout << "Both the trees are identical." << endl;**

**}**

**else**

**{**

**cout << "Given trees are not identical." << endl;**

**}**

**}**

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

Q18) Implement a C++ program for Heap Sort, a comparison-based sorting algorithm that utilizes a binary heap data structure to arrange elements in ascending order.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | arr={ 60 ,20 ,40 ,70, 30, 10} | arr={12,9,15,70,91,45} | arr={30,60,65,100} |
| **Output** | After heapifying array is  70 60 40 20 30 10  Sorted array is  10 20 30 40 60 70 | After heapifying array is  91 70 45 12 9 15  Sorted array is  9 12 15 45 70 91 | After heapifying array is  100 60 65 30  Sorted array is  30 60 65 100 |

Solution :

**#include <iostream>**

**using namespace std;**

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

**{**

**int largest = i; // Initialize largest as root Since we are using 0 based indexing**

**int l = 2 \* i + 1; // left = 2\*i + 1**

**int r = 2 \* i + 2; // right = 2\*i + 2**

**// If left child is larger than root**

**if (l < n && arr[l] > arr[largest])**

**largest = l;**

**// If right child is larger than largest so far**

**if (r < n && arr[r] > arr[largest])**

**largest = r;**

**// If largest is not root**

**if (largest != i) {**

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

**// Recursively heapify the affected sub-tree**

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

**}**

**}**

**// main function to do heap sort**

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

**{**

**// Build heap (rearrange array)**

**for (int i = n / 2 - 1; i >= 0; i--)**

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

**// One by one extract an element from heap**

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

**// Move current root to end**

**swap(arr[0], arr[i]);**

**// call max heapify on the reduced heap**

**heapify(arr, i, 0);**

**}**

**}**

**/\* 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()**

**{**

**int arr[] = { 60 ,20 ,40 ,70, 30, 10};**

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

**for(int i=n/2 -1;i>=0;i--){**

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

**}**

**cout << "After heapifying array is \n";**

**printArray(arr, n);**

**heapSort(arr, n);**

**cout << "Sorted array is \n";**

**printArray(arr, n);**

**return 0;**

**}**