Week3 lab

Name:ojas apratim

Roll number:55

Registration number:230957612

L3.1: Find Maximum Element in an Unsorted Array

#include <iostream>

using namespace std;

int findMax(int arr[], int n) {

int maxElement = arr[0]; // Start with the first element

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

if (arr[i] > maxElement) {

maxElement = arr[i];

}

}

return maxElement;

}

int main() {

int arr[] = {3, 1, 4, 1, 5, 9, 2, 6, 5};

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

cout << "Maximum element: " << findMax(arr, n) << endl;

return 0;

}

Time Complexity: O(n)

---

L3.2: Find Unique Element in an Array Where All Other Elements Appear Twice

#include <iostream>

using namespace std;

int findUnique(int arr[], int n) {

int unique = 0;

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

unique ^= arr[i]; // XOR all elements

}

return unique;

}

int main() {

int arr[] = {2, 3, 5, 3, 2, 5, 7};

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

cout << "Unique element: " << findUnique(arr, n) << endl;

return 0;

}

Time Complexity: O(n)

---

L3.3: Multiply Two Square Matrices of Order n Using Standard Approach

#include <iostream>

using namespace std;

void multiplyMatrices(int A[][3], int B[][3], int result[][3], int n) {

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

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

result[i][j] = 0;

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

result[i][j] += A[i][k] \* B[k][j];

}

}

}

}

int main() {

int A[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};

int B[3][3] = {{9, 8, 7}, {6, 5, 4}, {3, 2, 1}};

int result[3][3];

multiplyMatrices(A, B, result, 3);

// Output the result matrix

cout << "Result of matrix multiplication:" << endl;

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

cout << result[i][j] << " ";

}

cout << endl;

}

return 0;

}

Time Complexity: O(n^3)

---

L3.4: Compute the Number of Binary Digits of a Decimal Integer

#include <iostream>

using namespace std;

int countBinaryDigits(int n) {

int count = 0;

while (n > 0) {

n >>= 1; // Right shift by 1 bit

count++;

}

return count;

}

int main() {

int n = 29; // Binary representation: 11101

cout << "Number of binary digits: " << countBinaryDigits(n) << endl;

return 0;

}

Time Complexity: O(log n)

---

Recursive Algorithms:

L3.5: Compute Factorial of a Non-Negative Integer n

#include <iostream>

using namespace std;

int factorial(int n) {

if (n <= 1) {

return 1;

} else {

return n \* factorial(n - 1);

}

}

int main() {

int n = 5;

cout << "Factorial of " << n << " is: " << factorial(n) << endl;

return 0;

}

Time Complexity: O(n)

---

L3.6: Solve the Tower of Hanoi Problem

#include <iostream>

using namespace std;

void towerOfHanoi(int n, char from, char to, char aux) {

if (n == 1) {

cout << "Move disk 1 from " << from << " to " << to << endl;

return;

}

towerOfHanoi(n - 1, from, aux, to); // Move n-1 disks from 'from' to 'aux'

cout << "Move disk " << n << " from " << from << " to " << to << endl;

towerOfHanoi(n - 1, aux, to, from); // Move n-1 disks from 'aux' to 'to'

}

int main() {

int n = 3;

towerOfHanoi(n, 'A', 'C', 'B');

return 0;

}

Time Complexity: O(2^n)

---

L3.7: Compute Number of Binary Digits of a Decimal Integer n (Recursive)

#include <iostream>

using namespace std;

int countBinaryDigitsRecursive(int n) {

if (n == 0) {

return 0; // Base case: binary digits of 0 is 0

} else {

return 1 + countBinaryDigitsRecursive(n / 2); // Divide n by 2 and count digits recursively

}

}

int main() {

int n = 29;

cout << "Number of binary digits: " << countBinaryDigitsRecursive(n) << endl;

return 0;

}

Time Complexity: O(log n)

---

Summary of Time Complexities:

- L3.1: O(n)

- L3.2: O(n)

- L3.3: O(n^3)

- L3.4: O(log n)

- L3.5: O(n)

- L3.6: O(2^n)

- L3.7: O(log n)