1. Write a program to find the reverse of a given number using recursive.

#include <stdio.h>

int reverseNumber(int num, int rev)

{

if (num == 0)

return rev;

else

return reverseNumber(num / 10, rev \* 10 + num % 10);

}

int main()

{

int number, reversedNumber;

printf("Enter an integer: ");

scanf("%d", &number);

reversedNumber = reverseNumber(number, 0);

printf("Reverse of the number: %d\n", reversedNumber);

return 0;

}

1. Write a program to find the perfect number.

#include<stdio.h>

int main()

{

int num, rem, sum = 0, i;

printf("Enter a number\n");

scanf("%d", &num);

for(i = 1; i < num; i++) {

rem = num % i;

if (rem == 0){

sum = sum + i;

} }

if (sum == num)

printf(" %d is a Perfect Number",num);

else

printf("\n %d is not a Perfect Number",num);

return 0;

}

1. Write C program that demonstrates the usage of these notations by analyzing the time complexity of some example algorithms.

#include <stdio.h>

int main() {

int n = 10;

printf("Constant Time Complexity (O(1))\n");

printf("Linear Time Complexity (O(n))\n");

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

printf("%d ", i);

}

printf("\n");

printf("Quadratic Time Complexity (O(n^2))\n");

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

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

printf("(%d, %d) ", i, j);

}

printf("\n");

}

return 0;

}

1. Write C programs that demonstrate the mathematical analysis of non-recursive and recursive algorithms.

#include <stdio.h>

int nonRecursiveFactorial(int n)

{

int result = 1;

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

{

result \*= i;

}

return result;

}

int recursiveFactorial(int n)

{

if (n == 0)

{

return 1;

} else

{

return n \* recursiveFactorial(n - 1);

}

}

int main()

{

int num = 5;

int nonRecursiveResult = nonRecursiveFactorial(num);

printf("Non-Recursive Factorial of %d is: %d\n", num, nonRecursiveResult);

int recursiveResult = recursiveFactorial(num);

printf("Recursive Factorial of %d is: %d\n", num, recursiveResult);

return 0;

}

1. Write C programs for solving recurrence relations using the Master Theorem, Substitution Method, and Iteration Method will demonstrate how to calculate the time complexity of an example recurrence relation using the specified technique.

#include <stdio.h>

int masterTheorem(int n) {

if (n == 1) {

return 1;

} else {

return 2 \* masterTheorem(n / 2) + n;

}

}

int main() {

int n = 8;

printf("Result using Master Theorem: %d\n", masterTheorem(n));

return 0;

}

#include <stdio.h>

int substitutionMethod(int n) {

if (n == 1) {

return 1;

} else {

return 2 \* substitutionMethod(n - 1) + n;

}

}

int main() {

int n = 8;

printf("Result using Substitution Method: %d\n", substitutionMethod(n));

return 0;

}

#include <stdio.h>

int iterationMethod(int n) {

int result = 0;

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

result += 2 \* i + 1;

}

return result;

}

int main() {

int n = 8;

printf("Result using Iteration Method: %d\n", iterationMethod(n));

return 0;

}

1. Given two integer arrays nums1 and nums2, return an array of their Intersection. Each element in the result must be unique and you may return the result in any order.

int\* intersection(int\* nums1, int nums1Size, int\* nums2, int nums2Size, int\* returnSize){

int i, j = 0;

int size = nums1Size < nums2Size ? nums1Size : nums2Size;

int hash [1000] = {0};

int \*res = (int \*) malloc(size\*sizeof(int));

for (i = 0; i < nums1Size; i++)

hash[nums1[i]] = 1;

for (i = 0; i< nums2Size; i++) {

if (hash[nums2[i]] == 1) {

res[j++] = nums2[i];

hash[nums2[i]] = 2;

}

}

\*returnSize = j;

return res;

}

1. Given two integer arrays nums1 and nums2, return an array of their intersection. Each element in the result must appear as many times as it shows in both arrays and you may return the result in any order.

#include <stdio.h>

void arrayIntersection(int X[], int Y[], int m, int n) {

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

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

if (X[i] == Y[j]) {

printf("%d ", X[i]);

break;

}

}

}

}

int main() {

int arr1[] = {1, 2, 3, 4, 6};

int arr2[] = {1, 2, 5, 6, 7};

int m = sizeof(arr1) / sizeof(arr1[0]);

int n = sizeof(arr2) / sizeof(arr2[0]);

printf("Intersection is: ");

arrayIntersection(arr1, arr2, m, n);

return 0;

}

9. Given an array of integers nums, half of the integers in nums are odd, and the other half are even Sort the array so that whenever nums[i] is odd, i is odd, and whenever nums[i] is even, i is even. Return any answer array that satisfies this condition.

Sent by you: 9. Given an array of integers nums, half of the integers in nums are odd, and the other half are even Sort the array so that whenever nums[i] is odd, i is odd, and whenever nums[i] is even, i is even. Return any answer array that satisfies this condition.

class Solution {

public:

vector<int> sortArrayByParityII(vector<int>& nums) {

int n = nums.size();

int i = 0, j = 1;

while (i < n && j < n) {

if (nums[i] % 2 == 0)

i += 2;

else if (nums[j] % 2 == 1)

j += 2;

else

swap(nums[i], nums[j]);

}

return nums;

}

};