Data Structures and Practices

|  |  |
| --- | --- |
| **Topic** | Recursion |
| **Learning Objective** | Students can understand the principle of recursion.  Students can build a program using recursion. |
| **Recursion** | * A common method of simplification is to divide a problem into sub-problems of the same type * The function calls itself recursively on a smaller version of the input (n - 1) until reaching the base case |
| **Problem1**  **10points\*2** | Show what is wrong with each function.  **1) int recursive (int n)**  **{**  **if (n<=1) return 0;**  **return n\*recursive(n);**  **}**  **[Answer]**  If the value of n is less than or equal to 1, then 1 shall be returned. So n\*recursive(n) no, n\*recursive(n-1) yes.  **2) int recursive (int n)**  **{**  **printf("recursive(%d) \n", n);**  **return n\*recursive(n-1);**  **}**  **[Answer]**  Because the return function runs indefinitely, So added if (n<=1) return 1; |
| **Problem2**  **10points\*2** | Show the result of each function when it is called as recursive(5). Show what is displayed in the console and the return value.  **1) int recursive(int n)**  **{**  **printf("%d \n", n);**  **if (n<1) return -1;**  **else return (recursive(n-3)+1);**  **}**  **[Answer]**  **5**  **2**  **-1**  **2) int recursive(int n)**  **{**  **if (n!=1) recursive(n-1);**  **printf("%d \n",n);**  **return 0;**  **}**  **[Answer]**  1  2  3  4  5 |
| **Problem 3**  **10points\*2** | Use the following code to solve the problem below.   |  | | --- | | **#include <stdio.h>**  **void func(int\* a, int\* b)**  **{**  **int temp; temp=\*a; \*a=\*b; \*b=temp; return;**  **}**  **void main()**  **{**  **int num1=5; int num2=10;**  **printf("1: Number1 = %d ",num1); printf("Number2 = %d \n",num2);**  **func(&num1,&num2);**  **printf("2: Number1 = %d ",num1); printf("Number2 = %d \n",num2);**  **return;**  **}** |  1. Write down the purpose of **func()**  |  | | --- | | [Answer]  Initially, the values of 5 and 10 are stored in the addresses and converted at the second output with the pointer values to output 10 and 5. |  1. Write the results of the program.  |  | | --- | | [Answer]  1: Number1 = 5  Number2 = 10  2: Number1=10  Number2=5 | |
| **Problem4**  **20 points** | Write a C program to calculates the factorial of a given number.  5! = 5 × 4 × 3 × 2 × 1  **#include <stdio.h>**  **int factorial(unsigned int i)**  **{**  **if (n == 1) return 1;**  **return i\*factorial(i-1);**  **}**  **int main()**  **{**  **int i;**  **printf("Enter a positive integer: \n");**  **scanf("%d",&i);**  **printf("Factorial of %d is %d \n", i, factorial(i));**  **return 0;**  **}** |
| **Problem 5**  **4points\*5** | Express the time complexity of the following codes in Big O notation. Describe why?   |  |  |  | | --- | --- | --- | | 1) | for(i=1; i<n; i\*=2)  ++k; | [Answer] If n is 16. Then 1 2 4 8 16  Because of this approach, we do 2^x calculations  So Big O notation is O(logn) | | 2) | for(i=0; i<n; ++i)  if(i%2 ==0)  ++k; | [Answer] Repeat as i increases by 1 from 0 to n.  ++k is n/2 only when i is an even number.  So Big O notation is O(n/2) = O(n). | | 3) | for(i=0; i<n; ++i)  for(j=0; j <n; ++j)  ++k; | [Answer. The first i grows from 0 to n by 1. The second j grows from 0 to n by 1. So Big O nation is O(n^2) | | 4) | for (i = 1; i <= m; )  {  for (j = i; j <= m; )  j \*= 2;  i \*= 2;  } | [Answer] The log is repeated as the first j is doubled and the second i is doubled, so the is repeated.  So Big O nation is O(). | |  |  |  | |