I. សំណួរជ្រើសរើស

Instruction: *Read each question below carefully, and then* ***circle*** *the best answer choice by A, B, C, or D.*

1. Index of arrays in C programming langauge starts from ____________.

   [✓ A. 0]

   B. 1

   C. either 0 or 1

   D. undefined

   Answer: **A**

2. If there's no base criteria in a recursive program, the program will ____________

   A. not be executed.

   B. execute until all conditions match.

   [✓ C. execute infinitely.]

   D. obtain progressive approach.

   Answer: **C**

3. What will be the output of the following code snippet?

```
void solve() {
        int a[] = {1, 2, 3, 4, 5};
        int i, s = 0;
        for(i = 0; i < 5; i++) {
                if(i % 2 == 0) {
                        s += a[i];
                }
        }
        printf("%d\n", s);
}
```

   A. 5

   B. 15

   [✓ C. 9]

   D. 6

   Answer: **C**

   The code snippet basically calculates the sum of elements at even indices of the array, so we get 1 + 3 + 5 = 9 as the result.

4. What will be the output of the following code snippet?

```
    void solve() {
            int n, l, r, a, mid;
            n = 24; l = 0; r = 100; a = n;
            while(l <= r) {
                    mid = (l + r) / 2;
                    if(mid * mid <= n) {
                            a = mid;
                            l = mid + 1;
                    }
                    else {
                            r = mid - 1;
                    }
            }
            printf("%d\n", a);
}
```

   A. 5

   [✓ B. 4]

   C. 6

   D. 3

   Answer: **B**

   The code snippet basically uses binary search to calculate the floor of the square root of a number. Since the square root is an increasing function, so binary search is applicable here. Here, for n = 24, the answer is 4.

5. The logical or mathematical model of a particular organization of data is called a ____________.

   [✓ A. Data Structure]

   B. Data Arrangement

   C. Data Configuration

   D. Data Formation

   Answer: **A**

II. **សំណួរត្រិះរិះ និងលំហាត់**

6. Why we need to do algorithm analysis?

   A problem can be solved in more than one ways. So, many solution algorithms can be derived for a given problem. We analyze available algorithms to find and implement the best suitable algorithm.

7. What are the criteria of algorithm analysis?

   An algorithm are generally analyzed on two factors – time and space. That is, how much **execution** time and how much **extra space** required by the algorithm.

8. ចូរសរសេរ Recursive Algorithm (អនុគមន៍ភាសា C) ដើម្បីគណនា: $\frac{1}{2}+\frac{2}{3}+\frac{3}{4}+\cdots+\frac{n}{n+1}$

```
float sumf(int n) {
    if(n == 1) return 1.0/2.0;
    else sumf(n-1) + (float)n/(n+1);
```

9. គេឲ្យ algorithm មួយ ដូចខាងក្រោម៖

```
int sum(int n){
    int i, s, p;
    p = 1; s = 0;
    for(i = 1; i <= n; i++) {
            p = p * i;
            s = s + p;
    }
    return s;
}
```

9.1. ចូរសរសេរទម្រង់ផលបូកដែលបានមកពីលទ្ធផលរបស់ algorithm ខាងលើ។

9.2. ចូរសរសេរជា recursive algorithm (អនុគមន៍ភាសា C) មួយ ឲ្យសមមូលនឹង algorithm ដែលគេឲ្យខាងលើ។

9.3. ចូរសរសេរដំណើរការអនុវត្តរបស់ recursive algorithm ដែលបានសរសេរក្នុងសំណួរ 9.2 ក្នុងករណី n=5។

9.4. ចូរកំណត់តម្លៃ Big O នៃ recursive algorithm ដែលបានសរសេរក្នុងសំណួរ 9.2។

9.1. ចូរសរសេរទម្រង់ផលបូកដែលបានមកពីលទ្ធផលរបស់ algorithm ខាងលើ។

S(n) = 1! + 2! + 3! + … + (n-3)! + (n-2)! + (n-1)! + n!

9.2. ចូរសរសេរជា recursive algorithm (អនុគមន៍ភាសា C) មួយ ឲ្យសមមូលនឹង algorithm ដែលគេឲ្យខាងលើ។

9.3. ចូរសរសេរដំណើរការអនុវត្តរបស់ recursive algorithm ដែលបានសរសេរក្នុងសំណួរ 9.2 ក្នុងករណី n=5។

9.4. ចូររក Complexity (រកតម្លៃ Big Oh) នៃ recursive algorithm ដែលបានសរសេរក្នុងសំណួរ 9.2។

```
Here is the recursive function in C language to find 1! + 2! + 3! + 4! +...+ n!:

int factorial(int n) {
    if (n == 0 || n == 1) {
        return 1;
    } else {
        return n * factorial(n-1);
    }
}

int sumOfFactorials(int n) {
    if (n == 0) {
        return 0;
    } else {
        return factorial(n) + sumOfFactorials(n-1);
    }
}

The `factorial` function calculates the factorial of a given number using recursion. The
`sumOfFactorials` function uses this `factorial` function to calculate the sum of factorials from 1
to n using recursion as well.

To find the big O of this function, we need to analyze its time complexity. The `factorial` function
has a time complexity of O(n), since it needs to calculate all the previous factorials before
calculating the current one. The `sumOfFactorials` function calls `factorial` recursively for each
value from 1 to n, so its time complexity is O(n^2).

Therefore, the big O of this function is O(n^2).
```