Homework #1

Due Time: 2023/03/31 00:00

Contact TAs: algota@noj.tw

Introduction and Rules

1. The judge system is located at https://noj.tw, please submit your code by the deadline.
2. Homework Rule
3. Can I refer to resources from the Internet or other sources that are not from textbooks or lecture slides?
   - Yes, but you must include a reference source, such as a website or book title and page number, and attach it as a comment at the top of the code.
   - Although you can refer to external resources, please write your own code after the reference.
   - Remember to specify the references; otherwise we'll view it as cheating.
4. The Non-Programming part is no need to hand in. Just for practicing!

Programming Part

Please go to Normal Online Judge to read the programming problem.

Non-Programming Part

1. Complexity

For each function of the following C program, find out the tightest bound using Θ notation.

(1).

```c
void f(int n) {
    int cnt = 0;
    for ( int i=1; i<=n; i++ ) {
        for ( int j=i+1; j<=n; j++ ) {
            cnt = cnt + 1;
        }
    }
}
```

(2).

```c
void h(int n, int m) {
    int sum = 0;
    for ( int i=0; i<n; i++ ) {
        sum = sum + i;
    }
    for ( int i=0; i<m; i++ ) {
        sum = sum + i;
    }
}
```

(3).

```c
void g(int n) {
    if ( n <= 0 ) {
        return;
    }
    g(n-2);
    g(n-1);
}
```

(4).

```
void j(int n) {
    if ( n == 1 ) {
        return;
    }
    j(n/2);
    j(n/2);
    int total = 0;
    for ( int i=0; i<n; i++ ) {
        total = total + i;
    }
}
```

(5).

```
void f(int n) {
    int cnt = 0;
    for ( int i=1; i<=n; i++ ) {
        for ( int j=i; j<=n; j+=i ) {
            cnt = cnt + 1;
        }
    }
}
```

Hint: ‘Squeeze theorem‘

For each recurrence relation below, find out the tightest bound using Θ notation. Show your work.

(6). $T(n) = 9T(n/3) + n$

(7). $T(n) = 2T(n/2) + n$

(8). $T(n) = 2T(n/2) + n \lg n$

(9). $T(n) = 2T(n/2) + n\sqrt{n}$

2. Fibonacci

The Fibonacci numbers form a sequence, called the Fibonacci sequence, such that each number is the sum of the two preceding ones, starting from 0 and 1.

That is, $F_0 = 0,\ F_1 = 1$, and $F_N = F_{N-1} + F_{N-2},\ for\ N > 1$

The beginning of the sequence is thus: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ..., and it can be described in Matrix form:

$$\begin{pmatrix} F_{k+2} \\ F_{k+1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_{k+1} \\ F_k \end{pmatrix}$$

Thus, we can figure out F_N with $F_0,\ F_1$ by:

$$\begin{pmatrix} F_N \\ F_{N-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{N-1} \begin{pmatrix} F_1 \\ F_0 \end{pmatrix}$$

(1).

To multiple two n-dimension matrix, we can implement by a Naïve algorithm.

```
for ( int i=0; i<n; i++ ) {
    for ( int j=0; j<n; j++ ) {
        c[i][j] = 0;
        for ( int k=0; k<n; k++ ) {
            c[i][j] = c[i][j] + a[i][k]*b[k][j];
        }
    }
}
```

What is the complexity of this algorithm? Using Θ notation with n.

(2).

Let $A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$

To obtain A^{N-1} is quite simple:

```
Power(A):
    ans = I; // Identity Matrix
    for ( int i=0; i<N-1; i++ ) {
        ans *= A; // Matrix Multiplication
    }
    return ans;
```

The complexity is $O(N)$, but we've learned that Divide & Conquer can calculate x^n more efficient.

Please provide a Divide & Conquer algorithm or method, which can obtain A^{N-1} in $o(N)$, and analyze the complexity of your solution by Master Theorem.

You can provide C code, Pseudo code or just explain the method in plain text. However, make sure your method is clearly described.

3. Quick Sort Attack

The following sorting algorithms will make the input sequence ascending if it doesn't mention.

The pseudo code of quick sort algorithm:

```
Quick Sort(A, p, r)
    if p < r
        q = PARTITION(A, p, r)
        QUICKSORT(A, p, q - 1)
        QUICKSORT(A, q + 1, r)
```

```
PARTITION(A, p, r)
    x = A[r]
    i = p - 1
    for j = p to r - 1
        if A[j] <= x
            i = i + 1
            exchange A[i] with A[j]
    exchange A[i+1] with A[r]
    return i + 1
```

(1).

Find out the worst, the best, and the average time complexity (using Θ notation) of the above quick sort algorithm, and prove or explain why your answer is correct.

(2).

Please complete following C-code function to generate a sequence to make the above algorithm always sort it with the worst time complexity, and briefly explain why your code can do this.

```c
int *attack(int n){
    int *arr = malloc(n * sizeof(int));
    // your code here
    return arr;
}
```

4. Bubble Sort

Given a sequence $A = [a_1, a_2, a_3..., a_n]$, we could say (i, j) an inversion if $1 \leq i \leq j \leq n$ and $a_i > a_j$. Let $I(A)$ mean the number of inversions.

```
BubbleSort(A)
    for i = 1 to A.length - 1
        for j = A.length downto i + 1
            if A[j] < A[j - 1]
                exchange A[j] with A[j-1]
```

Above is the pseudo code of bubble sort.

(1).

(a)

Suppose a_i is the $k - th$ small element, which index is it at after the whole array is sorted by bubble sort? (assuming index started from 1)

(b)

Let x be the number of elements which are larger than a_i and are on his left-hand side in the initial array (before sorting), and let y be the number of elements which are smaller than a_i and are on his right-hand side. How many swapping does a_i encounter during the bubble sort procedure?

(c)

Following the previous question. How many inversions are formed by a_i ? That is, the number of inversions containing a_i.

(2).

Please try to explain or prove why the number of exchanges equal to $I(A)$, the pairs of inversion.

(a)

Please prove why after one swapping, the $I(A)$ decrease exactly 1.

(b)

Using the conclusion from (a), please prove or explain why $I(A)$ equals to the numbers of bubble sort swap.

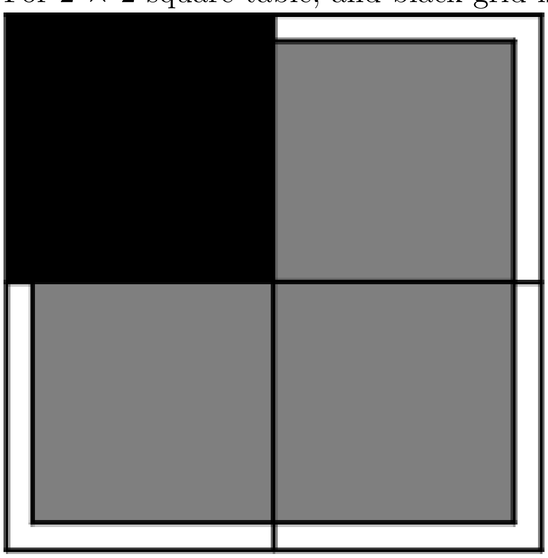
Hint

$A = [1, 5, 2, 4, 3]$ $I(A) = 4$
because (5, 2), (5, 3), (5, 4), (4,3) are all inversions.

5. Chess Covered

Given a 2^k length square table with one black grid in it. Our goal is using L-shape bricks to fill up square table without bricks overlapping.

For 2×2 square table, and black grid is on (0, 0).



(1).

Please draw the filled 4×4 square table with black grid on (1, 1).

(2).

In above, we can view one 4×4 square table as four 2×2 square tables. In order to solve easily, we wants to divide 4×4 square table to 4 four 2×2 square table and conquer it. It is say that we can transform 4×4 square table to four 2×2 squares contain one black grid. Can you find a strategy to simplify 4×4 square table equals to four 2×2 square tables?

(3).

In class, we learned about **Divide & Conquer method**. Please give a recursive solution and analyze the time complexity by **Master Theorem**.