Ch. 1 Basic Concepts 參考答案

Question 1

Let's assume that there are two functions A and B, and the time complexity of each is:

A(n): $2n^2$+4n

B(n): 12n

(1) Does function A run faster when n = 3 ?

(2) Is function B faster when n = 10 ?

(3) What is the Big-O of the two functions?

(4) For overall performance, which function is more efficient?

Ans: Question 1

(1) Yes

(2) Yes

(3) A: O(n^2) , B: O(n)

(4) B

Question 2

Please rank 1~5 for the following cases from the best time complexity (the fastest) to the worst when n is big enough (ex: n = 100000)

- f1 = θ(n) 2
- f2 = θ(2^n) 5
- f3 =θ(log n) 1
- f4 = θ(n^2) 4
- f5 = θ(nlogn) 3

Question 3

Big-O: $f(n)$ = $O(g(n))$ iff there exist c, n_0 > 0 such that $f(n) \leq cg(n)$ for all $n \geq n_0$

Omega: $f(n)$ = $\Omega(g(n))$ iff there exist c, n_0 > 0 such that $f(n) \geq cg(n)$ for all $n \geq n_0$.

Please find possible c and n_0, for each corresponding complexity notations.

- **(a)** $f(n) = 100n^3 + 50n + 6$

$f(n) = O(n^3) = \boldsymbol{\Omega}(n^3)$

O(n^3): c = 1000000, n_0=1

Ω(n^3): c = 0.0000001, n_0=1

- **(b)** $f(n) = 2n + 8n \log n$

$f(n) = O(n \log n)$ **=** $\boldsymbol{\Omega}(n \log n)$

O($n \log n$): c = 1000000, n_0=1

Ω($n \log n$): c = 0.0000001, n_0=1

Question 4

- According to the lecture video, a recursive code usually performs poorly, in memory usage, than a pure iterative code, while the time complexity appears to be the same. Explain why we often still adopt recursions in coding?

1. For small input size, the performance difference is negligible for recursion compared to iteration.
2. The code for recursion is easier to write compared to iteration.

Question 5

(1) Please write down the Pseudo Code of Fibonacci series using the iterative method.

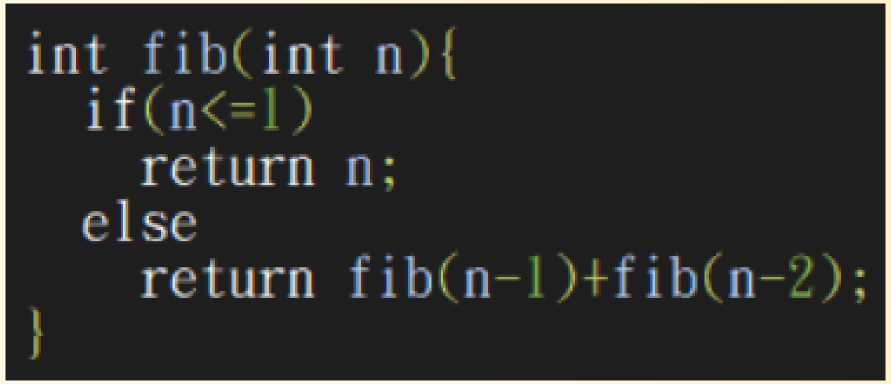
(2) Please write down the Pseudo Code of Fibonacci series using the recursive method.

(3) Please calculate their performance using big-O notation, and which one is more efficient?

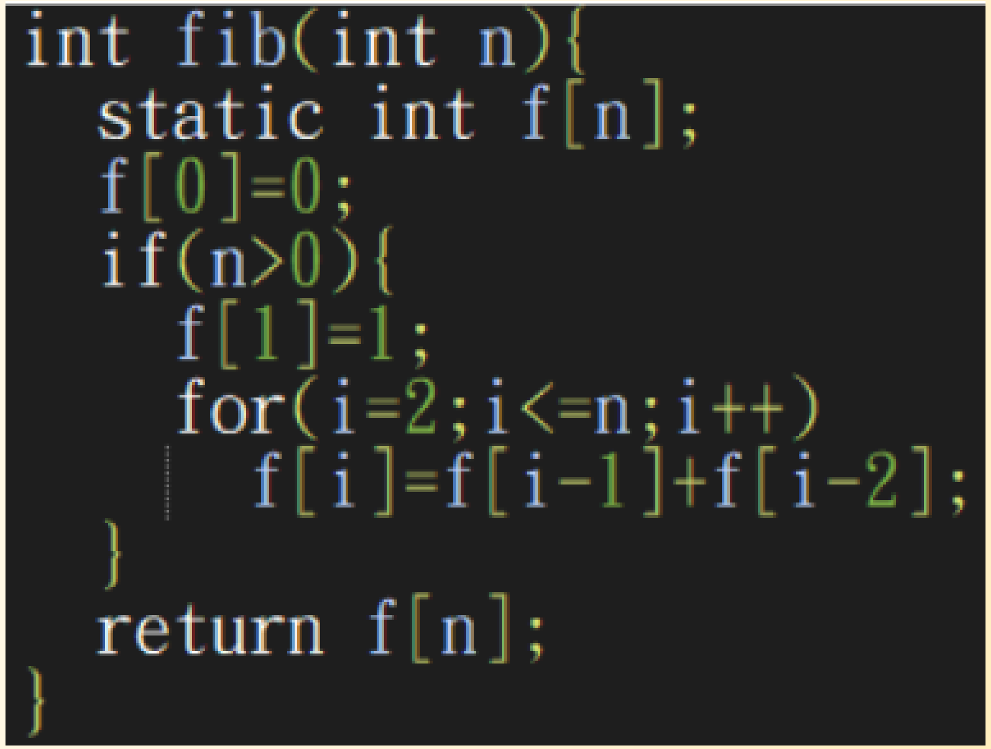
Ans: Question 5

參考: https://www.geeksforgeeks.org/program-for-nth-fibonacci-number/

- Recursion法為 O(2^n)
- Iteration法為O(n)



```
int fib(int n){
  if(n<=1)
    return n;
  else
    return fib(n-1)+fib(n-2);
}
```



```
int fib(int n){
  static int f[n];
  f[0]=0;
  if(n>0){
    f[1]=1;
    for(i=2;i<=n;i++)
      f[i]=f[i-1]+f[i-2];
  }
  return f[n];
}
```

Question 6

Which of the following statements is correct and why the other is wrong?

$A.$ $O(n!)$ **is the most efficient among the following time complexity functions:** $O(n^2), O(2n\ ), O(nlogn), O(n!)$**.**

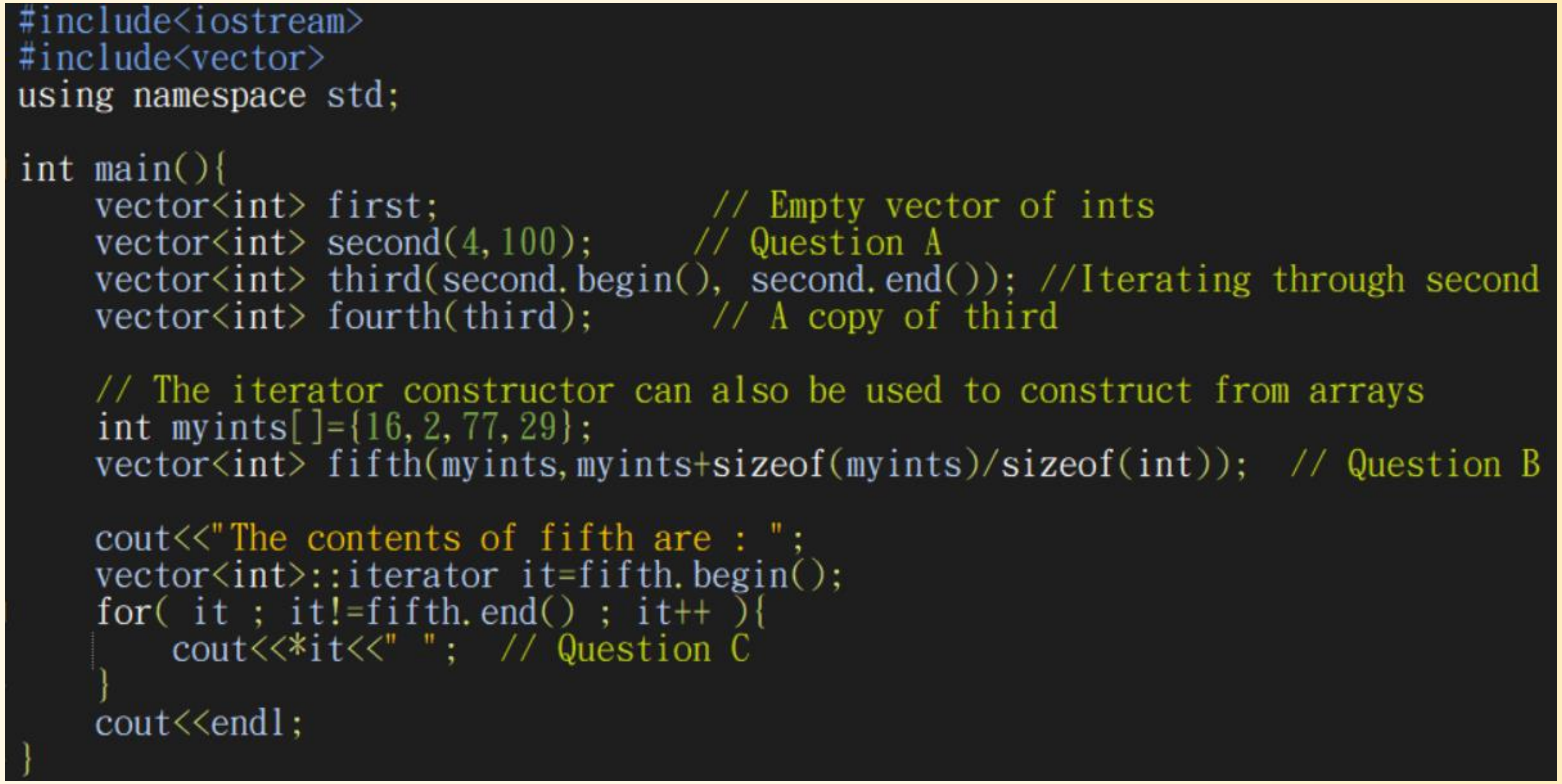
B. Suppose that A, B, C are matrices of size $n * n$**, the time complexity for the matrix multiplication C=A*B is** $O(n^3)$**. (Using the most general algorithm)**

C. A pair of iterators define a sequence which starts from the first element and ends at the last element.

D. A feasible recursive algorism may have increased parameter values.

B is the most accurate.
(however, there are some faster algorithm: https://stackoverflow.com/questions/8546756/matrix-multiplication-algorithm-time-complexity)
A: $O(n!)$ is the slowest one.
C: end() of a vector points to the element next to the last element. (https://www.geeksforgeeks.org/vectorbegin-vectorend-c-stl/)
D: Usually the parameters have to decrease after every function call (or it will never stop.)

Question 7



```
#include<iostream>
#include<vector>
using namespace std;

int main(){
    vector<int> first;                  // Empty vector of ints
    vector<int> second(4,100);         // Question A
    vector<int> third(second.begin(), second.end()); //Iterating through second
    vector<int> fourth(third);          // A copy of third

    // The iterator constructor can also be used to construct from arrays
    int myints[]={16,2,77,29};
    vector<int> fifth(myints,myints+sizeof(myints)/sizeof(int));  // Question B

    cout<<"The contents of fifth are : ";
    vector<int>::iterator it=fifth.begin();
    for( it ; it!=fifth.end() ; it++ ){
        cout<<*it<<" ";  // Question C
    }
    cout<<endl;
}
```

A. What does this vector constructor “**vector<int> second(4,100);**” do?
B. What does this vector constructor “**vector<int> fifth(myints,myints+sizeof(myints)/sizeof(int));**” do?
C. Why do we need to dereference “it” for cout?

Ans:

A. vector<int> second(4,100):

Initialize the vector with capacity=4, and with all element=100.

A. vector<int> fifth(myints,myints+sizeof(myints)/sizeof(int)):

Initialize the vector with array “myints[]”, and import elements from the array indexed from 0 to sizeof(myints)/sizeof(int). (ALL elements)

A. Why do we need to dereference “it” for cout?

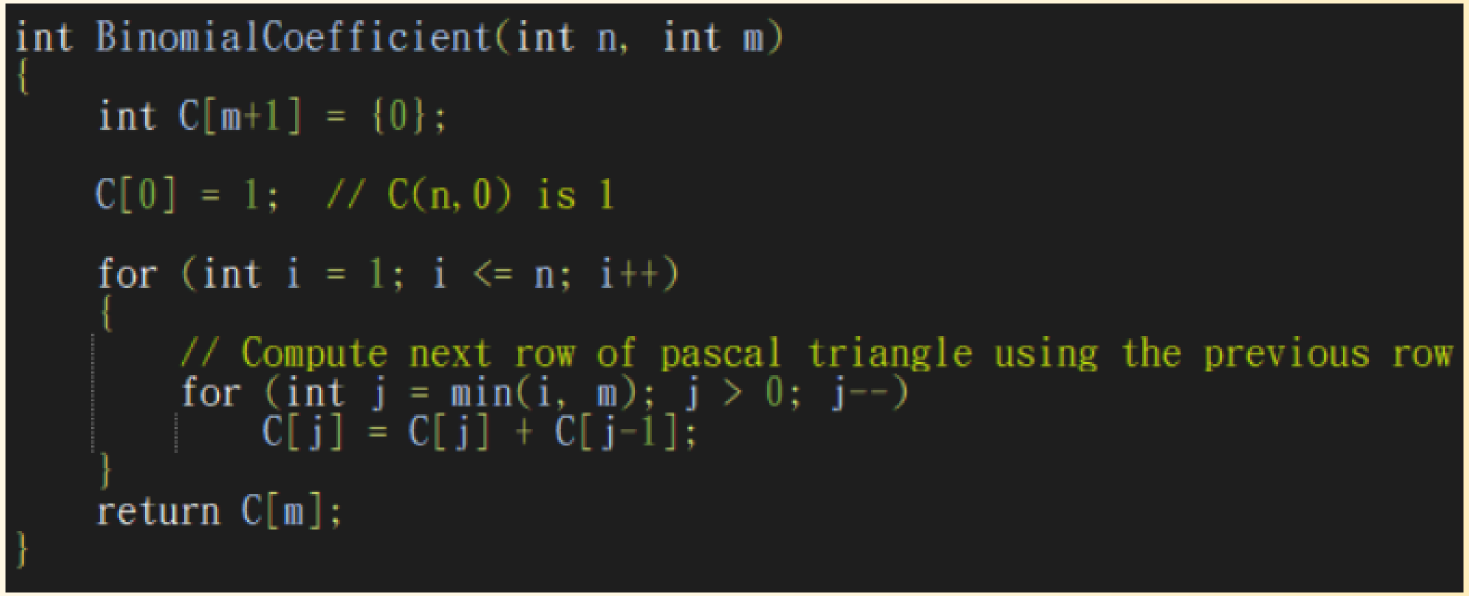
Iterators function as a pointer, which point to a “memory address”. So we need to dereference an iterator to get the actual value it points to.

Question 8

Please design an algorithm to compute Binomial Coefficient C(n,m), and analyze the space complexity and time complexity of your algorithm.

C(n, m) = C(n-1, m-1) + C(n-1, m)
C(n, 0) = C(n, n) = 1

"Dynamic Programming"
Space: O(m) (A row in Pascal Triangle)
Time: O(n*m)



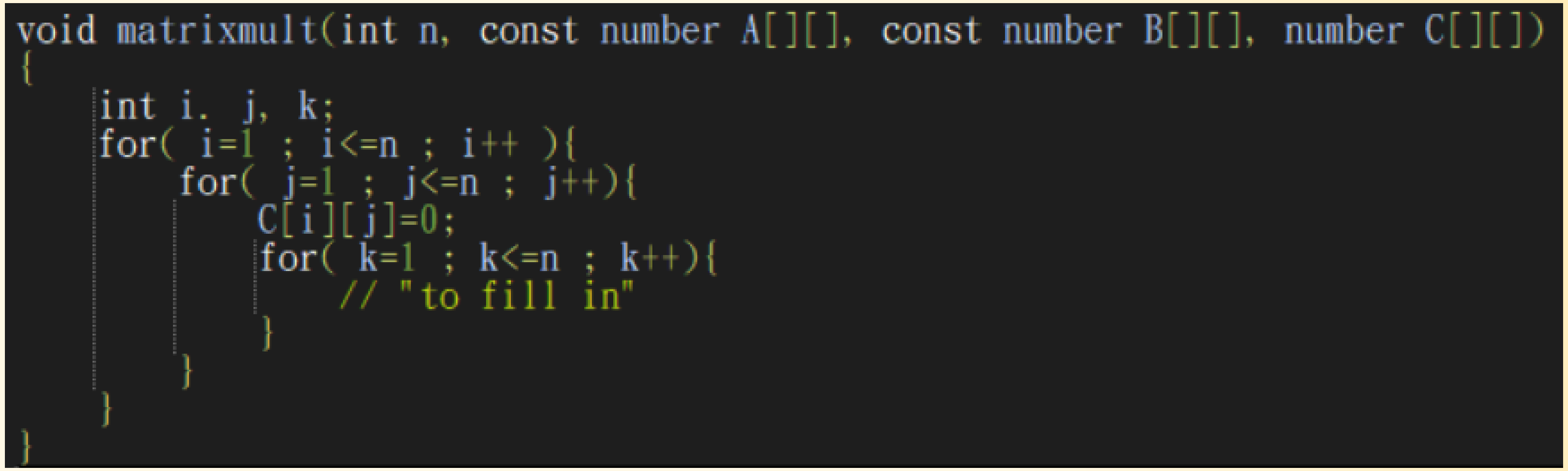
```
int BinomialCoefficient(int n, int m)
{
    int C[m+1] = {0};

    C[0] = 1;  // C(n,0) is 1

    for (int i = 1; i <= n; i++)
    {
        // Compute next row of pascal triangle using the previous row
        for (int j = min(i, m); j > 0; j--)
            C[j] = C[j] + C[j-1];
    }
    return C[m];
}
```

參考: https://www.geeksforgeeks.org/binomial-coefficient-dp-9/

Question 9



```
void matrixmult(int n, const number A[][], const number B[][], number C[][])
{
    int i. j, k;
    for( i=1 ; i<=n ; i++ ){
        for( j=1 ; j<=n ; j++){
            C[i][j]=0;
            for( k=1 ; k<=n ; k++){
                // "to fill in"
            }
        }
    }
}
```

A two-dimensional matrix multiplication is given below. A and B are two n*n two-dimensional matrices, and C is the output of A*B.

A. Please fill in the part marked “to fill in”.

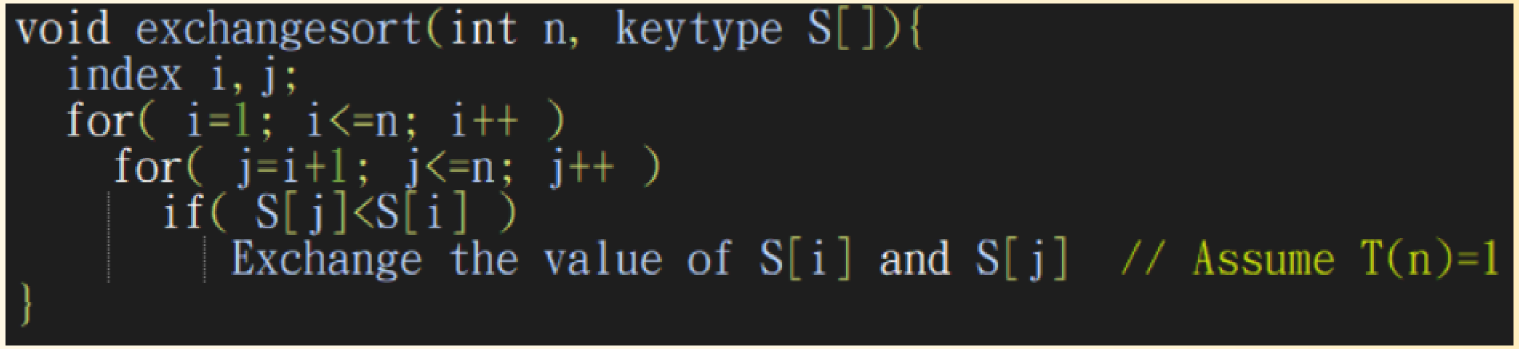
B. Please compute time complexity in Big-O notation and explain why.

Ans: Question 9

A. C[i][j] += A[i][k] * B[k][j];

B. O(n^3). We have n*n numbers to calculate to obtain C. Each number in C needs to multiply a row from A and a column of B which require n multiplications and n additions.

Question 10



```
void exchangesort(int n, keytype S[]){
  index i,j;
  for( i=1; i<=n; i++ )
    for( j=i+1; j<=n; j++ )
      if( S[j]<S[i] )
        Exchange the value of S[i] and S[j]  // Assume T(n)=1
}
```

A. Prove that time complexity T(n) = $\frac{(n-1)n}{2}$

B. Prove that T(n) = $\frac{(n-1)n}{2}$ = $\Omega(n^2)$

[Hint : For question B, you may start from the definition of Ω]

A. Total number of "if" operation = $\sum_{i=1}^{n}[\sum_{j=i}^{n}(1)] = \frac{(n-1)n}{2}$, which means the time complexity T(n) should be O($\frac{(n-1)n}{2}$).

B. $\frac{n^2}{4} = \frac{n^2}{2} - \frac{n^2}{4} \leq \frac{n^2}{2} - \frac{n}{2} = \frac{(n-1)n}{2}$, for $n \geq 10000000.$

so we can choose c= $\frac{1}{4}$, n_0=10000000 such that $\frac{(n-1)n}{2}$ ≥ c n^2 for all n ≥ n_0

Question 11

Answer TRUE or FALSE to each question and explain why:

1. $n \log n = O(n^2)$ True
2. $n^2 = O(n \log^2 n)$ False
3. $O(n^{0.1}) + O(\log n) = O(n^{0.1})$ True

Question 12

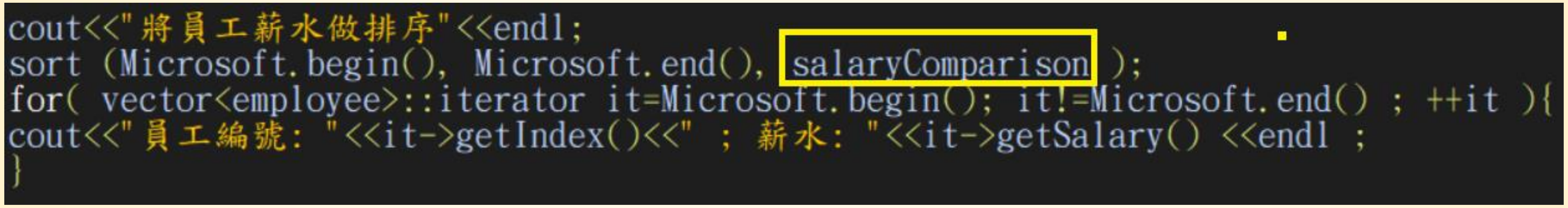
- Rank the following functions by the order of growth rate:

log(n!) 、 n*(2^{n}) 、 (logn)! 、 $2^{2\log \mathrm{n}}$ 、 n! 、 2^{n} 、 log(logn) 、 n

log(logn) < $2^{2\log \mathrm{n}}$ < n < log(n!) < (logn)! < 2^{n} < n*(2^{n}) < n!

- $2^{2\log \mathrm{n}}$可看成$4^{\log \mathrm{n}} = n^{\log 4}$約 $= n^{0.6\sim}$
- 階乘之近似: https://zh.wikipedia.org/wiki/%E5%8F%B2%E7%89%B9%E9%9D%88%E5%85%AC%E5%BC%8F

Question 13


```
cout<<"將員工薪水做排序"<<endl;
sort (Microsoft.begin(), Microsoft.end(), salaryComparison );
for( vector<employee>::iterator it=Microsoft.begin(); it!=Microsoft.end() ; ++it ){
cout<<"員工編號: "<<it->getIndex()<<" ; 薪水: "<<it->getSalary() <<endl ;
}
```

- bool salaryComparison(employee x1, employee x2) {return x1.getSalary() < x2.getSalary(); }

NOTE: salaryComparison should be a global function.