No.

Date . .

Homework for chap2

2.2-2

```
Selection-Sort(A)
1  for i = 1 to A.length-1        // n-1次外循环
2      min = i
3      for j = i+1 to A.length    // 从i~尾找最小元素坐标
4          if A[j] < A[min]
5              min = j
6      temp = A[i]                // 交换
7      A[i] = A[min]
8      A[min] = temp
```

~~int~~ Loop invariant:

At the start of each iteration of the for loop ~~in~~ of lines 1-8, the subarray $A[1 \cdots i-1]$ consists of the smallest $i-1$ elements in Array A, and in sorted order.

Why dose it need to run for only the first $n-1$ elements?

A: after $n-1$ times of loop, according to our loop invariant, the subarray $A[1 \cdots n-1]$ consists of the smallest $n-1$ elements, therefore, the last one is the biggest.

Running times: best case: $\Theta(n^2)$

worst case: $\Theta(n^2)$

No.

Date . .

2.

Fibonacci (n) // return nth number in Fibonacci array

	代价	次数
let F[1..n] be a new array	C_1	1
F[1] = 1	C_2	1
F[2] = 1	C_3	1
for i = 3 to n	C_4	n-1
F[i] = F[i-1] + F[i-2]	C_5	n-2
return F[n]	C_6	1

$T(n) = C_1 + C_2 + C_3 + nC_4 - C_4 + nC_5 - 2C_5 + C_6$

$\Rightarrow an + b$

$T(n) = \Theta(n)$

3. 2.3-7 $\Theta(n \lg n)$

first, use merge sort $\Theta(n \lg n)$, make set S ordered

next, for each element in S, s_i, $i = 1 \cdots n$.

search $S[i+1 \cdots n]$ with binary search, which is $\Theta(\lg n)$ for $x - s_i$

if finally find $x - s_i$, return

Θ true, eles return false

$\Theta(n \lg n) + n \cdot \Theta(\lg n) = \Theta(n \lg n)$

Homework

- Answer the following problem and justify your answer.

```
Input: integer n
Output: number of line 5 that is executed
1)  count=0
2)  for i=1 to n
3)      m=[n/i]
4)      for j=1 to m
5)        count=count+1`
6)      end for
7)  end for
8)  return count
```

第5行执行的次数：
[n/1]+[n/2]+...+[n/n]

[n]表示向下取整

No.

Date

Problem 3-4

a. Disprove, $n = O(n^2)$ while $n^2 \neq O(n)$

b. Disprove $f(n) = n^2$ $g(n) = n$ $f(n) + g(n) = \Theta(n^2)$

c. Prove: $f(n) = O(g(n))$

$\therefore \exists c, n_0, \forall n \geq n_0, 0 \leq f(n) \leq cg(n)$

$0 \leq \lg(f(n)) \leq \lg c + \lg(g(n))$ $(n \geq n_0)$

make another constant $c_1 = \frac{\lg c}{\lg(g(n))} + 1$. $c_1 \leq \lg c + 1$

So, $0 \leq \lg(f(n)) \leq (\lg c + 1)\lg(g(n))$

$= \lg c \cdot \lg(g(n)) + \lg(g(n))$

$\lg(c) + \lg(g(n)) \Rightarrow \lg(f(n)) = O(\lg(g(n)))$ (Since $\lg(g(n)) \geq 1$)

d. Disprove: $2n = O(n)$ while $2^{2n} = 4^n \neq O(2^n)$

e. Disprove: $f(n) < 1$

(However, consider in Algorithm running time situation, hardly can $f(n)$ be such functions. so the statement may be right?)

f. $0 \leq f(n) \leq cg(n) \rightarrow 0 \leq \frac{1}{c} f(n) \leq g(n) \rightarrow g(n) = \Omega(f(n))$

g. $f(n) = \Theta(f(n/2))$ Disprove: make $f(n) = 2^n$

$c_1 2^{n/2} \leq 2^n \leq c_2 2^{\frac{n}{2}}$ for $\forall n \geq n_0$. impossible

h. ~~$\Theta f(n)$~~ make $g(n) = O(f(n))$. $\exists c, n_0. \forall n \geq n_0, 0 \leq g(n) < cf(n)$

then $0 \leq c_1 f(n) \leq f(n) + g(n) \leq c_2 f(n)$ is true

with $c_1 = 1$, $c_2 = c + 1$

No.

Date . .

Homework on chap3

3.1-4

Is $2^{n+1} = O(2^n)$? Is $2^{2n} = O(2^n)$?

$2^{n+1} = 2 \times 2^n$

① $O(g(n)) = \{ f(n)$: there exist positive constants C and no such that $0 \le f(n) \le cg(n)$ for all $n \ge no\}$

make $C = 3, no = 0$ then

$0 \le 2 \times 2^n \le 3 \times 2^n$ for all $n \ge no$, then $2^{n+1} = O(2^n)$

② $2^{2n} = (2^n)^2 = 4^n$ no such C and no can make

$0 \le 4^n \le C \times 2^n$ for all $n \ge no$

if so, there'll be $0 \le 2^n \le C$ for all $n \ge no$,

but C is a constant. that is impossible.

2.

for a sequence $a_1 .. a_n$ real numbers find $(i, j) \to \sum_{k=i}^{j} a_k$ max

Solution 1: calculate all possible outcomes, find the biggest

let RES[n][n] be a new array

```
for i=1 to n
    for j=i to n
        for k= i to j
            calculate a_i + ... + a_j
        RES[i][j] = res
Find the maximum of RES matrix
    return the index
```

performance analysis:

$1n + 2(n-1) + 3(n-2) + \cdots + n \cdot 1$

$= \frac{n^3 - 3n^2 + 2n}{6} = O(n^3)$

No.

Date . .

Solution 2 (5)

```
let max[n] be a new array    // max[i]表示以S[i]结尾中最大和
max[1] = S[1]
for i=2 to n
    max[i] = Math.max(max[i-1]+S[i], S[i])
~~for j=1 to n~~
maxval = 0 - int.MAX_VAL
for j=1 to n                 // 找到最大和
    maxval = Math.max(max[j], maxval)
backindex = 0
for j=1 to n                 // 找到最右坐标
    if max[j]==maxval
        backindex = j
        break
~~while maxval!=0~~
frontindex = backindex
while maxval!=0              // 找到最左坐标
    maxval -= S[frontindex]
    frontindex -= 1
return [frontindex+1, backindex]
```

performance analysis: O(n)