Homework Week 7

1) Find the Big O

if $n = 2^m$

```
for (int i=n; i>0; i--) {            // loop 3
  for (int j=1; j<n; j*=2) {         // loop 2
    for (int k=0; k<j; k++) {        // loop 1
      ... // constant number of operations
}}}
```

loop 1 → runs j times ⇒ $O(j)$

loop 2 → j doubles each time until $< n$.
this is a logarithmic loop $\log_2(n)$ ⇒ $O(\log n)$

j	iteration	work done by inner loop 1.
1	1	1
2	2	2
4	3	4
8	4	8
...	...	...
n/2	$\log_2(n)$	n/2

This is a geometric series

Total $= 1+2+4+8+ \dots + n/2 = 2n - 1 \approx O(n)$

So loop 1 + loop 2 = $O(n)$

loop 3 ⇒ runs n times from n to 1.
Each time it runs, it triggers the full B+C loop which does $O(n)$ work.

$$\boxed{\Rightarrow O(n \times n) = O(n^2)}$$

loop 3 (n iterations) × loops {1+2 ($O(n)$) work each

Problem 2: $f(n) = \Theta(f(n/2))$ Is this always true.

i.e. If I cut the input in half, will the time/steps stay about the same (up to a constant multiplier)?

Solution:

let $n = 8$

f-n	f(n)	f(n/2)	Same growth?
n	8	4	yes
n^2	64	16	yes
$\log n$	≈ 3	≈ 2	yes
2^n	256	16	no.

For $f(n) = 2^n$, halving n makes the result much, much smaller.

so $f(n) \neq \Theta(f(n/2))$.

It works for some functions (linear or quadratic) but not for faster growing ones, like exponential.

Problem 1a: Find Big O (worst case time complexity).

```
int myTest (int n){                       // stops when n ≤ 0
    if (n<= 0) return 0
    else {
        int i = random (n-1);
        return myTest (i) + myTest (n-1-i);   // recursive 2 calls each time
                 call 1          call 2
    }
}.
```

Solution:

- This f-n stops when $n \leq 0$
  else
  - → picks a random number $0 \geq i \leq n-1$.
  - → recursively calls
    my Test (i)
    my Test $(n-1-i)$.

→ Always makes 2 recursive calls

Ex: $n = 3$

my Test (3)
- my Test (0) → 0
- my Test (2)
  - my Test (0)
  - my Test (1)
    - my Test (0)
    - my Test (0)

- This is like a binary tree, where each node creates 2 children.

Even though the input size gets smaller, the number of calls grows exponentially ⇒ $O(2^n)$

$\boxed{3-3}$ Rank the following f-ns by order of growth:

- $\lg(\lg^* n)$ - logarithm of the iterated logarithm - an extremely slow-growing f-n.

  $2^{\lg^* n}$ - $O((\lg n)^k)$ for any $k > 0$.

  $(\sqrt{2})^{\lg n}$ - $\Theta(\sqrt{n})$

  n^2 → $\Theta(n^2)$

  $n!$ → factorial f-n.

  $(\lg n)!$ - factorial of log. of n

  2^{2^n} - double exponential f-n, grows incredibly fast, faster than $n!$

-