1. (a) True

(b) True

(c) False

(d) True

(e) False

(f) True

(g) False

(h) True

(i) True

(j) False

2. $$\lim_{n\to\infty} \left| \frac{\log(n^2)}{\log(n)} \right| = \lim_{n\to\infty} \left| \frac{2\log(n)}{\log(n)} \right| = 2 \geq 0$$

$$\Rightarrow \log(n^2) \in O(\log n) \quad \&\& \quad \log(n^2) \in \Theta(\log n)$$

$$\lim_{n\to\infty} \left| \frac{\log(n)}{\log(n^2)} \right| = \lim_{n\to\infty} \left| \frac{\log(n)}{2\log(n)} \right| = \frac{1}{2} \geq 0$$

$$\Rightarrow \log(n^2) \in \Omega(\log n) \quad \&\& \quad \log(n^2) \in \Theta(\log n)$$

(a)(c)(e) are true

3. $\lim_{n\to\infty} \left| \frac{(n+1)!}{n!} \right| = \lim_{n\to\infty} \left| \frac{n! \cdot (n+1)}{n!} \right| = \lim_{n\to\infty} |n+1| \to \infty$

do not have limit, so the statement is False

4. 1° a, b, c together form a right triangle (c has obstacle)

the sum of two sides of a triangle is bigger than the third side

$\Rightarrow a+b > c$

as a right triangle $\Rightarrow c^2 = a^2+b^2$

$\because (a-b)^2 = a^2+b^2-2ab \geq 0$

$\therefore a^2+b^2 \geq 2ab$

$\therefore a^2+b^2+a^2+b^2 \geq a^2+b^2+2ab$

$2a^2+2b^2 \geq (a+b)^2$

$2(a^2+b^2) \geq (a+b)^2$

$\because a^2+b^2 = c^2$

$\therefore 2c^2 \geq (a+b)^2$

$\Rightarrow \sqrt{2}c \geq a+b$

put together $\Rightarrow c < a+b \leq \sqrt{2}c$

2° two approaches go the same way (no obstacle)

$c = a+b$

$\Rightarrow c = a+b < \sqrt{2}c$

combine Case 1° and Case 2° together:

$c \leq a+b \leq \sqrt{2}c$

now we can prove that $(a+b) \in \theta(c)$ since we

have found two positive real numbers :

$$k_1 \cdot c \leq a+b \leq k_2 \cdot c \quad (k_1 = 1,\ k_2 = \sqrt{2})$$

so in terms of total distance traveled, the two approaches are computationally equivalent.

5.(a) we follow these steps to append n objects to the end of an array of size 1 :

for the ith object $(1 \leq i \leq n)$
  if no empty space in the array ①
    the current size of array is k, then allocate a new array of size $k+1$
    copy original $(i-1)$ elements from current array to the new array
  put the ith object into the ith position ②
  store the next object

for each object we take two constant operations :
① : determine whether out of space
② : write the ith object into the ith position

since the size of array is increased by 1 when out of space, to store n objects, we need resize $n-1$ times.
for each resize step, we will take 1 operation to allocate a new array and $i-1$ operations for copy. so in total is $i-1+1 = i$ operations.

~~complexity : $T(n) = (1+2+\cdots+(n-1)) +$~~

(a) complexity : $T(n) = \underbrace{(2+3+4+\cdots+(n-1)+n)}_{n-1 \text{ terms}} + 2n$

$$= \frac{(n-1)(n+2)}{2} + 2n$$

$$= \frac{1}{2}n^2 + \frac{5}{2}n - 1$$

$$\in O(n^2)$$

since the complexity is $O(n^2)$, this approach won't give us constant time read / write

(b) we follow these steps to append n objects to the end of an array of size 1 :

for the ith object $(1 \leq i \leq n)$
  if no empty space in the array ①
    the current size of array is k, then allocate a new array of size $k+100$
    copy original $(i-1)$ elements from current array to the new array
  put the ith object into the ith position ②
store the next object

for each object we take two constant operations :
① : determine whether out of space
② : write the ith object into the ith position

since the size of array is increased by 100 when out of space, to store n objects, we need resize when the object number is a multiple of 100 plus 2 :

(b) $2, 102, 202, \cdots\cdots, 100k+2$

so the last object number that need resize is n', which is the biggest number of a multiple of 100 plus 2 that less than n:

$$n' = 100k + 2 < n \qquad (k \text{ is integer})$$

since $n - n' = c$ (c is constant), ~~$O(n) \quad O(n-c)$~~

$\Rightarrow \quad n' = (n-c) \in O(n)$

we need resize $\frac{n'-2}{100}+1$ times in total.
for each resize step, we will take 1 operation to allocate a new array and $i-1$ operations for copy, so intotal is $i-1+1 = i$ operations.

complexity: $T(n) = \underbrace{(2+102+202+\cdots+n')}_{1+\frac{n'-2}{100} \text{ terms}} + 2n$

$$= \frac{(1+\frac{(n'-2)}{100})(n'+2)}{2} + 2n$$

$$= \frac{1}{200}n'^2 + 2n - \frac{1}{50} + \frac{1}{2}n' + 1$$

$$\in O(n^2)$$

since the complexity is $O(n^2)$, this approach won't give us constant time read / write

(c) We follow these steps to append n objects to the end of an array of size 1:

for the ith object ($1 \le i \le n$)
  if no empty space in the array ①
    the current size of array is k, then allocate a new array of size $2k$
    copy original $(i-1)$ elements from current array to the new array
  put the ith object into the ith position ②
  store the next object

for each object we take two constant operations:
① : determine whether out of space
② : write the ith object into the ith position

Since the size of array is doubled when out of space, to store n objects, we need resize when the object number is a power of 2 plus 1:

$$2, 3, 5, 9, \ldots\ldots, 2^k+1$$

So the last object number that need resize is n', which is the biggest number of a power of 2 plus 1 that less than n:

$$n' = 2^k + 1 < n \quad (k \text{ is integer})$$

Since $n - n' = c$ (c is constant),
$\Rightarrow$ $n' = (n-c) \in O(n)$

We need resize $\log_2^{(n'-1)} + 1$ times in total.

(c) for each resize step, we will take 1 operation to allocate a new array and $i-1$ operations for copy, so in total is $i-1+1 = i$ operations.

complexity : $T(n) = \underbrace{(2+3+5+\cdots+n')}_{1+\log_2(n'-1) \text{ terms}} + 2n$

$$= (1+2+4+\cdots+(n'-1)) + 1 + \log_2(n'-1) + 2n$$

$$= \frac{1-2^{(\log_2(n'-1)+1)}}{1-2} + 2n + \log_2(n'-1) + 1$$

$$= 2 \cdot 2^{\log_2(n'-1)} + 2n + \log_2(n'-1)$$

$$= 2n' + 2n + \log_2(n'-1) - 2$$

$$\in O(n)$$

since the complexity is $O(n)$, this approach will give us constant time read / write.

6.(a) assume my car is in the nth location, if my algorithm is perfect and take me straight to my car, the minimum number of cars I have to pass is $O(n)$.

(b) walk in one direction:

best case: go straight to my car. If my car is in the nth location, the number of cars I need pass is $O(n)$

worst case: go the opposite direction to where my car locates. It will take me infinity time to keep going and I will not find my car forever.

So this method doesn't always work. If I choose the wrong direction, then I can never find my car.

(c) In the first round, I would check one car to one side, then back to origin. check one car to another side, and finally back to origin. So in this round, I have passed each car twice (go and back), so the number of cars I have passed in total is: $2 \times 2 \times 1 = 4$ cars.
Similarly, in the second round, I would pass 8 cars, and in the third round is 12 cars. If my car is in the nth location, then I need to keep searching till the nth round, in which I would pass $4n$ cars.

complexity: $T(n) = 4 \times (1+2+3+\cdots+n)$

$$= 4 \times \frac{n(n+1)}{2}$$

$$= 2n^2 + 2n \in O(n^2)$$

so I need to pass $O(n^2)$ cars to find my car.

(d) In the first round, I check 1 car to each side. Then in the second round. I check $2\times1=2$ cars to each side. Then in the third round. I check $2\times2=4$ cars to each side. At the end of each round, I would double the number of cars that I would check in the next round.

(e) In my algorithm, I would check cars on both sides. And In each round, I would check every car in the specified range. Assume my car is in the nth location, we can always find a number, n', which is the smallest power of 2 that bigger than, n:

or equal to $n' = 2^k \geq n > 2^{k-1}$ (k is integer)

so in the round that I intend to check n' cars on both sides, I would definitely find my car.

(f) as mentioned in (e), I would keep searching until the round that I would check n' cars to each side. So the total number of rounds I would go through is $\log_2 n' + 1$

since $n' - n = c$ (c is constant)

$n' = n + c \in O(n)$

complexity: $T(n) = 4 \times c \underbrace{(1 + 2 + 4 + \cdots + \log_2 n')}_{\log_2 n' + 1 \text{ terms}}$

$$* = 4 \times \frac{1 - 2^{(\log_2 n' + 1)}}{1 - 2}$$

$$= 8n' - 4 \in O(n)$$

need to pass $O(n)$ cars to find my car, is a better algorithm.

7. (a) First, list all the farms from large to small and give them number from 1 (largest) to n (smallest)

Initially, set the left boundary, left, to be 1, and set the right boundary, right, to be n, and start search.

Search (left, right):
If right − left = 1 (only two farms in the range), go to the right farm
If the dragon is there, catch it
otherwise, wait at the right farm, and the dragon could be caught in the next round.

If right − left = 2 (only three farms in the range), go to the farm with number $\frac{left + right}{2}$
If the dragon is there, catch it
Else, if the farm has not been destroyed, wait there and the dragon could be caught in the next round
Else, if the farm has already been destroyed, go to the farm with number $\frac{left + right}{2} + 2$ in the next round, and the dragon could be caught there (not beyond n) < ①

Else, go to the farm with number $\frac{left + right}{2}$
If the dragon is there, catch it
Else, if the farm has not been destroyed, set
left = left + 1, right = $\frac{left + right}{2} + 1$ (not beyond n)
and start the next round of search

Else, if the farm has already been destroyed, set left = $\frac{left + right}{2} + 1$, right = right + 1 (not beyond n) and start the next round of search

① If we finally arrived at farm of number n and it has already been destroyed, that means the dragon has already destroyed all the farms

(b) as meintioned in (a), the worst case is that we keep searching untill we arrive at farm of number n and found it has already been destroyed, which means that the dragon destroyed all the farms and we failed to catch it.

Since we devide the size of the search domain in half and move it one farm right at each iteration

$$T(n) \rightarrow T(\frac{n}{2})$$

it is actually kind of a binary search algorithm, so its complexity would be $O(\log n)$

So the dragon would destroy $O(\log n)$ farms before we catch it.