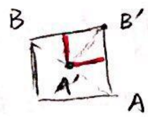
HW#4.

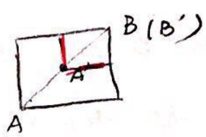
Dong Boyuan
1547489

Problem 1.

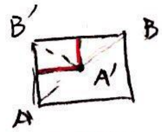
(a)

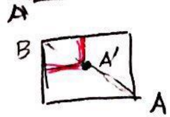
```
procedure FindShip(n, A, B)
  ** we have an nxn grid.
  ** A, B are two points
  A.x ← 0
  A.y ← 0
  B.x ← n
  B.y ← n
  bomb_num ← 0   //the number of bombs.
  while( dist(B.x, A.x)>1 or dist(B.y, A.y)>1 ) do
      x ← (A.x + B.x)/2
      y ← (A.y + B.y)/2
      bomb_num ← bomb_num+1
      c ← direction(x, y)
      if(c = NE) then
          if( A.x > B.x) then
              B.x ← A.x
          A.x ← x
          A.y ← y
```



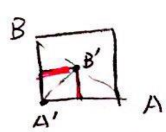


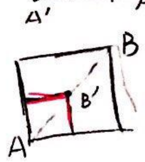

```
      if(c=NW) then
          if( A.x < B.x) then
              B.x ← A.x
          A.x ← x
          A.y ← y.
```



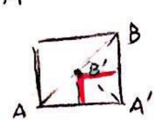


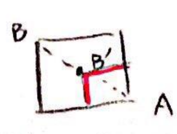

```
      if(c=SW) then
          if(A.x > B.x ) then
              A.x ← B.x
          B.x ← x
          B.y ← y.
```






```
      if (c= SE) then
          if( A.x < B.x) then
              A.x ← B.x
          B.x ← x
          B.y ← y
  return bomb_num
```






```
procedure dist(x, y)
  ** get the distance between x,y.
  ** return the absolute value
  if( (x-y)>0) then
      a ← x-y.
  else
      a ← y-x
  return a
```

```
procedure Direction(x, y)
  if( y > ship.y) then
      if( x > ship.x) then
          return NE
      else if( x < ship.x) then
          return NW.
  else if( y < ship.y) then
      if( x > ship.x) then
          return SE
      else if( x < ship.x) then
          return SW.
```

Each time we will find the center of the grid ~~create~~ that composed by A B, which would be the fastest way to determine the location of the battleship. This is because we divide the gride into 4 parts (each is a square grid), and it will help Alice to determine the specific location as soon as possible by using least number of sonar bombs.

(b). In the worst-case, we will take $O(\log n)$ time to find the ship, since each time we divide the grid into 4 parts. In this case we will use 5 ~~sno~~ sonar-bombs, and the ship will always be near the corner of each search grid composed by A, B. (but not equal to A or B), so we need to keep traversing until find the location, which will take $O(\log n)$ time.

Any other ~~so~~ cases, A, B will find the ship less than $O(\log n)$ time, ~~case~~ cause the coordinates between A and B is getting closer at $O(\log n)$ time, In each iteration, A or B's coordinates will be half of their original distances, so it will take $O(\log n)$ time to make their distance be ~~less~~ less ~~the lower-bounds and upper-bounds~~. or equal to 1. A, B will reach the lower-bounds and upper-bounds which will determine the ship's location.

Dong Boyuan

1547489

Problem 2

Dong Boyuan
1547489

(a).

we have n nodes, so we have $\binom{n}{2}$ possible edges in the graph, and each possible edge we have $Pr = p$ to draw the edge:

So $E[\text{edges}] = \binom{n}{2} p$ edges.

(b).

$E[\#\text{ triangles}] = \binom{n}{3} p^3$

we have n nodes, so we have $\binom{n}{3}$ possible different triangles in the graph, but it is a triangle only when all three edges are exist ~~whip~~ which is $Pr = p^3$. So, expected number of triangles will be $\binom{n}{3} p^3$.

Problem 3.

Dong Boyuan
1547489

procedure match (A, p, r, B, i)

** match the lightbulb's B[1..n] to the correct batteries A[p...r]

if(p< r) then

q ← partition (A, p, r, B[i])

** partition returns q such that i) A[q] is the correct battery for B[i]

** ii) All batteies have a voltage that is too small for the lightbulb B[i]

(which means that have a voltage smaller than battery A[q]) appear in A[p,...q-1]

** (iii) All batteries have a voltage that is too great for the light bulb B[i]

(which means that have a voltage greater than battery A[q]) appear in A[q+1,...,r]

match (A, p, q-1, B, i+1) ** math lightbulb B[i+1] in A[p...q-1]

match (A, q+1, r, B i+1) ** match lightbulb B[i+1] in A[q+1.., r].

procedure Partition (A, p, r, b)

```
pivot ← b
i ← p-1
for( j from p to r) do
    if( pivot has no light with A[j]) then
        i ← i+1
        exchange A[i] ⇔ A[j]
    if( pivot has light with A[j]) then
        battery ← A[j]

exchange A[i+1] ⇔ battery
find the battery for light b
return i+1
```

Label all the batteries with numbers from 1 to n, and label all lightbulbs with numbers from 1 to n. Then we find the correct battery for each lightbulbs one by one until we match them up.

Problem 4

Dong Boyuan
1547489

(a) In the worst case all m bits are flipped, so we take $O(m)$ time ($O(\log(n))$ time)

In the n increament operations:

0 ··· 0	we can find that
0 ··· 1	the 1st bit flip every time
0 ··· 10	the 2nd bit flip every 2nd increaments
0 ··· 11	the 3rd bit flip every $2^2=4$th increaments
0 ·100	the 4th bit flip every $2^3=8$th increaments
0 ··101	⋮
0 ··110	the i-th bit flip every 2^{i-1}-th increaments
0 ··111	
0 ··1000	we have n increaments in total.
0 ··1001	so the total operations is:
0 ··1010	
0 ··1011	
0 ··1100	
0 ··1101	
0 ··1110	
0 ··1111	
0 ··10000	

$$n+\frac{n}{2}+\frac{n}{4}+\frac{n}{8}+\cdots+\frac{n}{2^{m-1}}$$

$$= n\left(1+\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\cdots+\frac{1}{2^{m-1}}\right) = n\cdot\frac{1(1-(\frac{1}{2})^m)}{1-\frac{1}{2}}$$

$$= 2n(1-(\tfrac{1}{2})^m) < 2n\cdot 1 \in O(n)$$

In total the operation will take $O(n)$ time.

Amortized cost per operation is $\frac{O(n)}{n} = O(1)$ time

(b)

At first half $\frac{n}{2}$, i will keep increasing which will take $O(n)$ in total.

Then the binary ~~repree~~ representation will become $10\ldots0$.

In the last half $\frac{n}{2}$, i will decrease or increase which will make each iteration operate worst-case $O(\log n)$-time. So it will take $\frac{n}{2}\cdot O(\log n)$-time in total in the 2nd half part.

In total: It will take $O(n)+\frac{n}{2}O(\log n)$ time.

Amortized cost per operation is:

$$\frac{O(n)+\frac{n}{2}O(\log n)}{n} = O(1)+\frac{1}{2}O(\log n) > \frac{1}{4}(\log n) \in \Omega(\log n) \qquad c=\frac{1}{4};\ c\in N\ \forall n>10$$

So, its amortized cost is $\Omega(\log n)$

Problem 5

Dong Boyuan 1547489

(a) Claim: In any n-node tournament one can order all vertices from first to last (Hamiltonian path).

Prove by induction:

Base case: There are two nodes, only one edge. No matter which direction the edge is, the two nodes get the Hamiltonian path in this graph. → true.

Inductive step: Asumming n-1 node tournament holds the claim.

We need to show that for n node tournament also holds the claim.

We add a new node U_n to a tournament of size n-1

Case I: There's an edge from U_n to V_1, so we will get the Hamiltanian path

$U_n \to V_1 \to V_2 \to \cdots \to V_{n-1}$ → true.

Case II: There's an edge from V_{n-1} to U_n. so we will get the Hamiltonian path

$V_1 \to V_2 \to \cdots \to V_{n-1} \to U_n$ → true.

Case III: There's an edge from V_i to U_n and an edge from U_n to V_{i+1}

then we will get the Hamiltonian path:

$V_1 \to V_2 \to \cdots \to V_i \to U_n \to V_{i+1} \to \cdots \to V_{n-1}$ $(1 < i < n-1)$

$\cdots \to V_2 \to U_n \to V_3$. or

$\cdots \to V_3 \to U_n \to V_3$ or

$\cdots \to V_4 \to U_n \to V_5$ or

$\cdots$ $\cdots$ $\cdots$ or

$\cdots \to V_{i-2} \to U_n \to V_{i-1}$

① We already know $U_n \to V_{n-1}$, so, even though all V_i direct U_n, there will also a Hamilitonian path:

$V_1 \to \cdots \to V_{n-2} \to U_n \to V_{n-1}$

② We already know $V_1 \to U_n$, so, ~~es~~ even though ~~all~~ U_n direct to all V_i, there will also a Hamilitonian path:

$V_1 \to U_n \to V_2 \to \cdots \to V_{n-1}$

→ true.

So, in any n-node tournament one can order all vertices from first to last.

Dong Boyuan
1547489

(b).

procedure visit (G, node, stack, visited, n)

** G is an adjacency matrix that stores edges among nodes.
** node is the node that current visit.
** stack is a stack stores the path.
** Visited is a list stores n nodes, initialized with 0s (Visited [n] = $\{\underbrace{0, 0, \cdots, 0}_{n}\}$)
if the node is already visited, Visited [node] will change to be 1.
** n is the number of nodes

visited [node] ← 1
stack. push (node)
if (stack. capacity = n) then
    return stack.
for (i from node to n)
    if ((visited [i] = 0) and (G [node] [i] = 1)) then
        visit (G, i, stack, visited, n)
visited [node] ← 0
stack. pop ( )

~~procedure full(visited, n)~~
~~** check if the list, visited is full.~~
~~** n is the number of nodes.~~
~~for (i from 1 to n)~~
~~if (visited [i] = 0)~~
~~return 0~~
~~return 1~~

procedure printS(stack)
** print the path.
if (stack. capacity = 0)
    return 0

c ← stack. top
stack. pop( )
printS(stack)
print (c)

Traversing all edges adjacent to a node u takes O(n) time, and there are n nodes in total, so, it will take $O(n^2)$ time to get n-1-edge path.
This algorithm is pretty like InsertionSort algorithm which also take $O(n^2)$ time.

7 6.