Name: Hiroshi Shu; NetID: hshu5
Name: Sicheng Chu; NetID: schu37

1 (a)

```
N = $2^k$
divideAlgorythm($List[0...N]$)
if(n = 4)
Return findFaultyCoin(n)
End if
Left[] = $List[0 \cdots \frac{N}{4}]$
Right[] = List$[(\frac{N}{4}) + 1\frac{N}{2}]$
Compare (Left[] with Right[])
if (equal weight)
    divideAlgorythm($List[(\frac{N}{2} + 1) \cdots N]$)

    Return
End if
Else
    divideAlgorythm($List[0 \cdots \frac{N}{2}]$)
    Return
End else

findFaultyCoin($n = 2^2$)
Name 4 coins a, b, c, d arbitrarily
 Compare a with b
If (equal weight)
    Compare b with c
    if(equal weight)
        d is the target
        Return d
    End if
    else
        c is the target
        Return c
    End else
End if
Else
    Compare b with c
    if(equal weight)
        a is the target
        Return a
    End if
    else b is the target
        Return b
    End else
End else
```

1(b) findFaultyCoin use tester twice.
So when
$N = 4 = 2^k, k = 2, T(N = 4) = 2$
The recurrence for divideAlgorythm is

$$T(N) = T(\frac{N}{2}) + 1$$

$$= T(\frac{N}{4}) + 1 + 1$$

$$= T(\frac{N}{8}) + 1 + 1 + 1$$

With i iterations $T(N) = T(\frac{N}{2^{i+1}}) + i + 1$

let $\frac{N}{2^{i+1}} = 4$ then $i + 1 = log_2\frac{N}{4})$

$$T(N) = T(4) + i + 1 = 2 + log_2\frac{N}{4}$$

$$= 2 + log_2N - log_2 4 = log_2 N$$

Since $N = 2^k, k = log_2 N$, therefore we have $O(N) = k$.

1(c) Proof of divideAlgorythm(List[])
base case: when $k > 1$ i.e, $N = 4$, assume findFaultyCoin(n) correctly find the coin, hence we find the faulty coin, solved.
Induction case: divideAlgorythm($List[N = 2^k]$) works for any $k > 1$.
When $N = 2^{k+1}$:
Compare $List[0 \cdots \frac{N}{4}]$ with

$List[(\frac{N}{4}) + 1 \cdots \frac{N}{2}]$ means compare $List[(0 \cdots 2^{k-1}]$
with $List[2^{k-1} + 1 \cdots 2^k]$. If they weigh the same, obviously the faulty coin is in $List[2^k \cdots 2^{k+1}]$
where we call divideAlgorythm($List[N = 2^k]$ hence
solved because of the base case. If they weigh
differently than the faulty coin is in $List[0 \cdots 2^k]$. we also call divideAlgorythm($List[N = 2^k]$ and solves the problem.

Proof of findFaultyCoin($n = 2^2$)
comparison between a and b gives us the fact that
the faulty coin isn't in between them when they
weighed the same and is in between them when weighed differently. When the faulty one is in between a and b, compare a with a real coin(coin c or coin d) tells us which one among a and b is
a counterfeit. When the faulty coin is in c and d, vice versa, compare one of them with a real coin gives us the targeted faulty coin.

2(a)
Counter example: Let A be [1,4,2,3]. When removing line 6-8, the middle part of number will not be swapped in one Gather call. Following the given algorithm, since the n = 4 in this case:
(A)Line(9): Gather(A[1,2]);
(B)Line(10): Gather(A[3,4]);
(C)Line(11): Gather(A[2,3]);
Follow the recursive call:
For (A): A[1,2] = [1,4];
For (B): A[3,4] = [2,3];
For (C): A[2,3] = [2,4];
Since in each recursive call of A,B,C, n = 2, we have reach the base case. The final result if A = [1,2,4,3], which is not the expected result.

2(b)
Counter example: Let A be [3,4,1,2]. After swapping line 10 and 11, the middle part of array will be gathered before the end of the array. Following the given algorithm, since the n = 4 in this case:
(A)Line(6) to (8): A = [3,1,4,2];
(A)Line(9): Gather(A[1,2]);
(B)Line(10): Gather(A[2,3]);
(C)Line(11): Gather(A[3,4]);
Follow the recursive call:
For (A): A[1,2] = [1,3];
For (B): A[2,3] = [3,4];
For (C): A[3,4] = [2,4];
Since in each recursive call of A,B,C, n = 2, we have reach the base case. The final result if A = [1,3,2,4], which is not the expected result.

2(c)
slowSort
Base case: $n = 1$ the list has one element, be default is sorted.
Induction hypothesis:Assume the slowSort is correct for an input array with $n = 2^k$ elements, we want to prove the slowSort is true for $n = 2^{k+1}$ case.
Induction step:
Assume Gather will produce correct result for an input array A.
For $n = 2^{k+1}$, there are 2^{k+1} elements in the array. In slowSort, by line (1), the array will be divide into to two part. Let A1 = $A[1 \cdots \frac{n}{2}]$, A2 = $[\frac{n}{2} + 1 \cdots n]$. Then for recurrence call happened on A1, since $\frac{n}{2} = \frac{2^{k+1}}{2} = 2^k$, the slowSort will give the correct result for A1. Similarly, the slowSort will give correct result of A2. By line (4), the well-sorted result of A1,

denoted with A1’, and the correct result of A2, denoted with A2’, will compose new A. That is A’ = [A1’, A2’]. Since by assumption, Gathe will produce correct result, line (4) will make A’ sorted. ||

Claim: Gather algorithm will produce the correct result for an input array A.

Base case: for $n = 2$, the algorithm swap the two numbers when they are unsorted and leave them unchanged when they are in the sorted order. This complete the sort when there is only 2 elements.

Induction hypothesis: Assume the Gather algorithm will produce correct result for an input array with size $n = 2^k$.

Inductive step: For an input array A with size $n = 2^{k+1}$, let A = [a,b,c,d] where a,b,c,d are the four equal size part of the array. Notice that after the slowSort for $n = 2^k$, a and b are in $A[1 \cdots \frac{n}{2}]$ which is in ascending order, Similarly, c and d are in ascending order. So we have $a < b$ and $c < d$. Then part b and part c are swapped from line 6-9, so A = [a, c, b, d]; Notice for a,b,c,d their sizes are $\frac{2^{k+1}}{4} = 2^{k-1}$ equally. Thus Gather($A[1 \cdots \frac{n}{2}]$) becomes Gather(list[a,c]) Since a,b,c,d have the the same size of 2^{k-1}, then the Gather(list[a,c]) has size 2^k. According to the inductive hypothesis, this gives us a sorted list which we name List[a1,c1] where $a1 < c1$.Similarly Gather($A[(\frac{n}{2}) + 1 \cdots n]$) gives us a sorted list which we name List[b1, d1] where $b1 < d1$. For a1, since $a1 < c1$, then a1 contains the smallest 2^{k-1} elements in set {a,c}, so $a1 \leq a < b$. Similarly, $a1 \leq c < d$. Since elements from {b1,d1} are from {b,d}. Thus, $a1 < b1$ and $a1 < d1$. So a1 contains the smallest 2^{k-1} number of elements in A. Similarly, $max(c1) < min(d1)$. So $c1 < d1$. Since $a1 < b1 < d1, d1$ contains the largest 2^{k-1} elements in A. After line (11), Gather (list[b1,c1]) will produce a list (b2,c2) such that b2 contains smallest 2^{k-1} number of elements in {b1,c1} and c2 contains the largest 2^{k-1} number of elements in {b1,c1}. By ht program terminate, A = [a1,b2,c2,d1] in ascending order. So Gather algorithm correctly sorted the array A.

2(d)

Recurrence for Gather: denote the number of comparison as G

Base case: $n = 2$, then $G(n = 2) = 1 = O(1)$.

Recurrence relation: $G(n) = 3T(\frac{n}{2})$

Result: $G(n) = 3^{log_2 N - 1} = \frac{N^{log_2 3}}{3}$ with O($c^{log_2 N}$) or O($N^{log_2 3}$)

Recurrence for SlowSort: denote the number of comparison as T

Base case: $n = 1$. Then $T(n = 1) = 0$;

Recurrence relation: $T(n) = 2T(\frac{n}{2}) + G(n)$

Result = $O(c^{log_2 N} \cdot \frac{1-(c_1)^{log_2 N}}{1-c_1})$ or $O(c \cdot N^{log_2 3})$