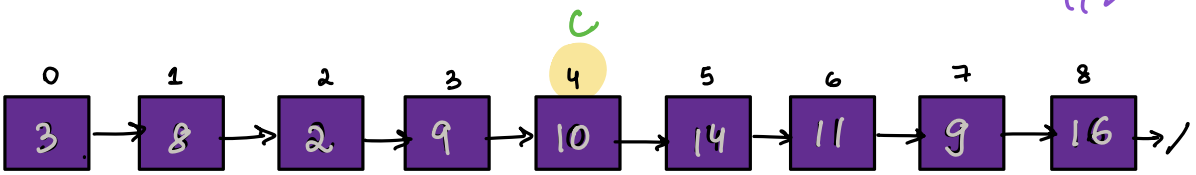
Find Middle

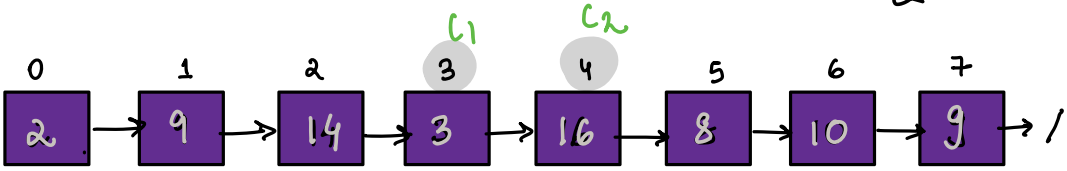
$\frac{N-1}{2} = \frac{9-1}{2} = 8/2 = 4$

$9/2 = 4$



$\frac{8-1}{2} = 7/2 = (3)$

$8/2 = (4)$



C & C2
(odd) (even)

1. Find size of the LL
2. K = size/2

getKthEle(head, size/2)

C & C1
(odd) (even)

1. Find size of LL
2. K = (size-1)/2

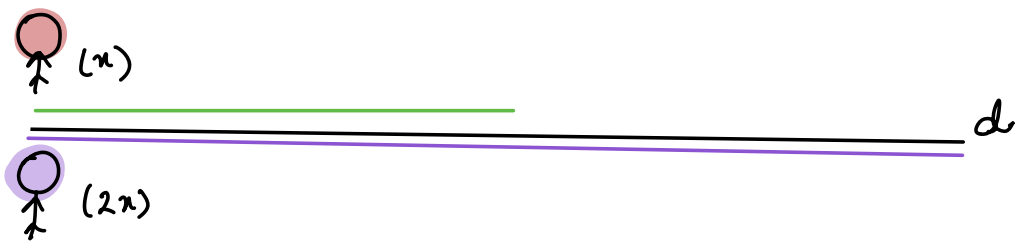
getKthEle(head, (size-1)/2)

TC: O(N)

Q.

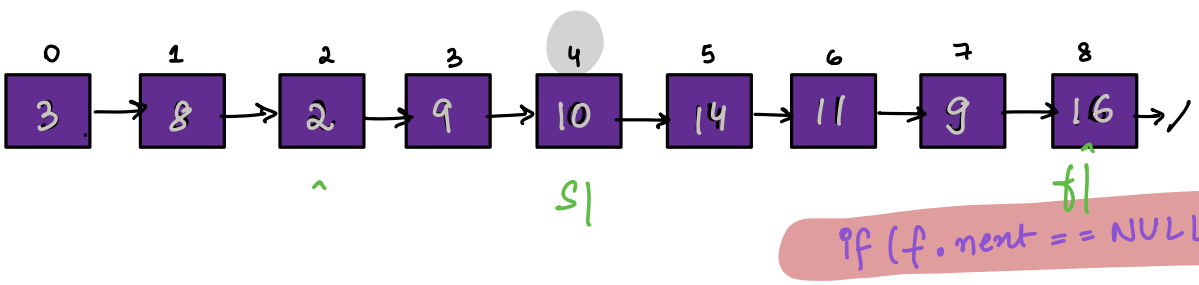
If odd → center
If even → C1

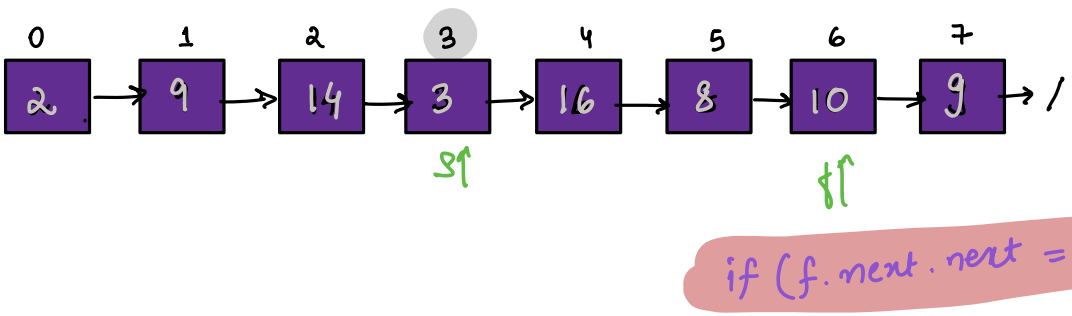
In single iteration



slow
↓
by 1

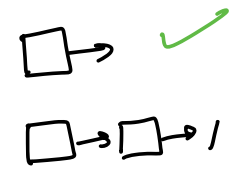
fast
↓
by 2 steps

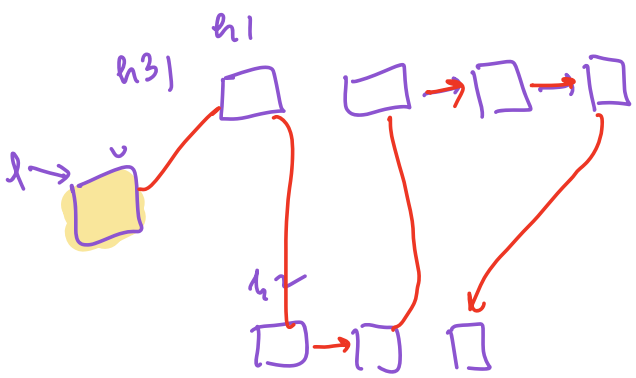




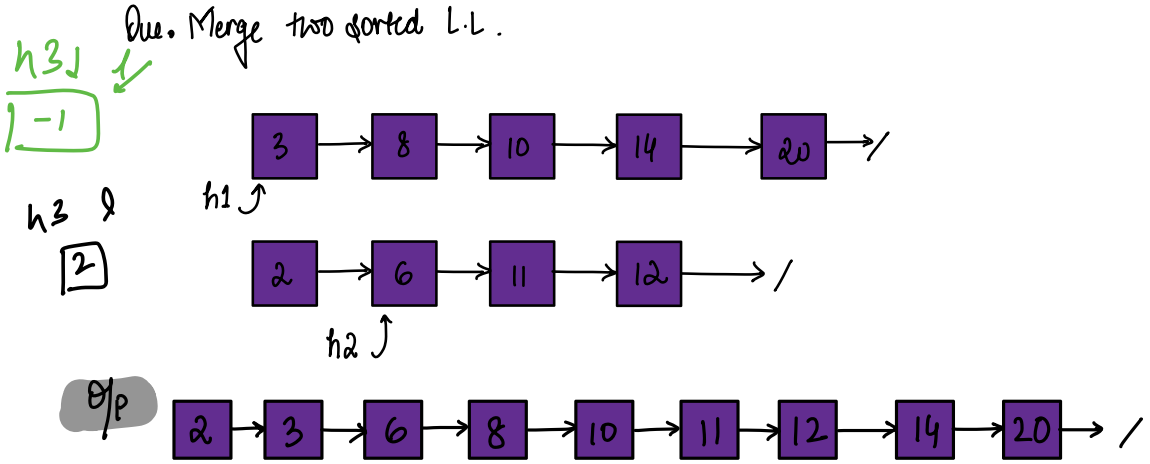
```
Node getMiddleEle (head){
    if(head == NULL) return head
    S=f=head
    while(f.next != NULL && f.next.next != NULL){
        S=S.next
        f=f.next.next
    }
} return S
```

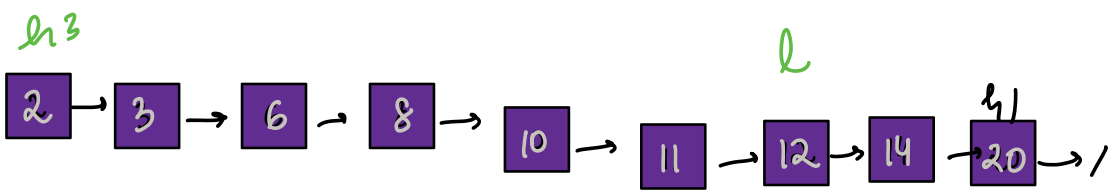
head == NULL ✓





Que. Merge two sorted L.L.





```
Node merge(h1, h2){
    if(h1==NULL) return h2
    if(h2==NULL) return h1
    if(h1.data < h2.data){
        h3 = h1 ; h1 = h1.next
        l = h3
    }
    else { h3 = h2 ; h2 = h2.next }
        l = h3
    while(h1 != NULL && h2 != NULL){
        if(h1.data < h2.data){
            l.next = h1
            h1 = h1.next
        }
        else{
            l.next = h2
            h2 = h2.next
        }
        l = l.next
    }
    if(h1==NULL){
        l.next = h2
    }
    if(h2==NULL){
        l.next = h1
    }
    return h3
}
```

Node h3 = new Node(-1)

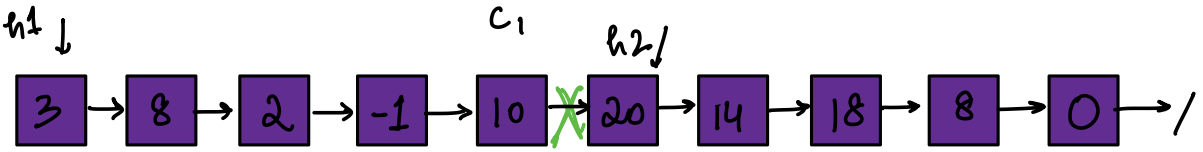
TC :
O(N+M)

SC :
O(1)

* If online contests, okay to use dummy node
+ Else avoid using it

MergeSort

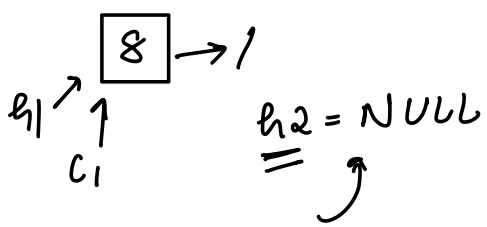
Assumption : your function will return the sorted LL



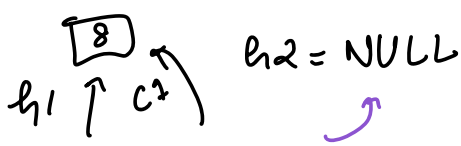

```
Node mergeSort ( head) {
    if (head == NULL || head.next == NULL) return head .

    h1 = head
    Node c1 = getMiddleEle( head)        O(N)
    h2 = c1.next
    c1.next = NULL
    h1 = mergeSort(h1)
    h2 = mergeSort ( h2)
    return merge ( h1, h2)               O(N)
}
```

head == NULL



mergerSort (h1)



$T(N) = 2T(N/2) + N$

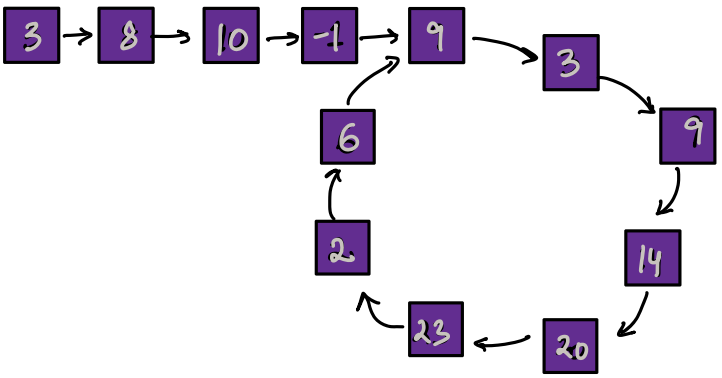
TC : O(N log N)

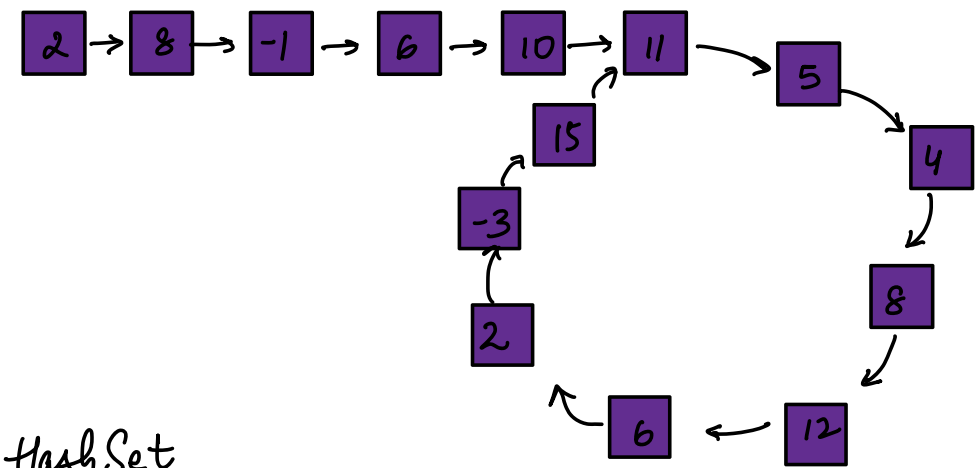
SC : O(log N) stack space

Break : 10 : 35

Detect Cycle

HashSet<Node> s;



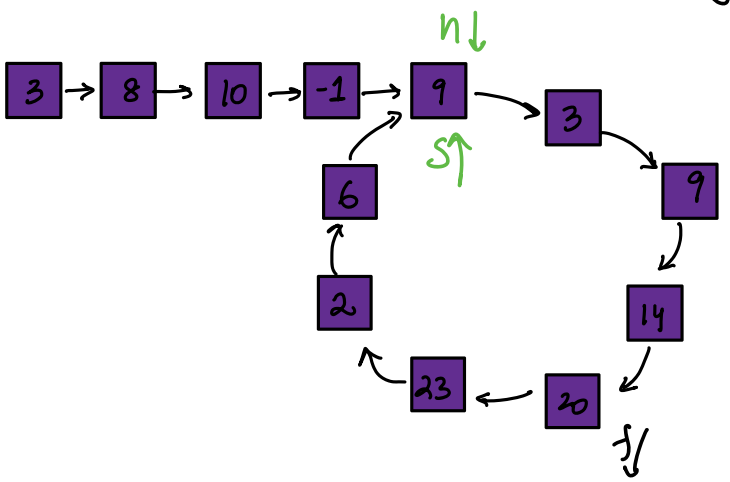


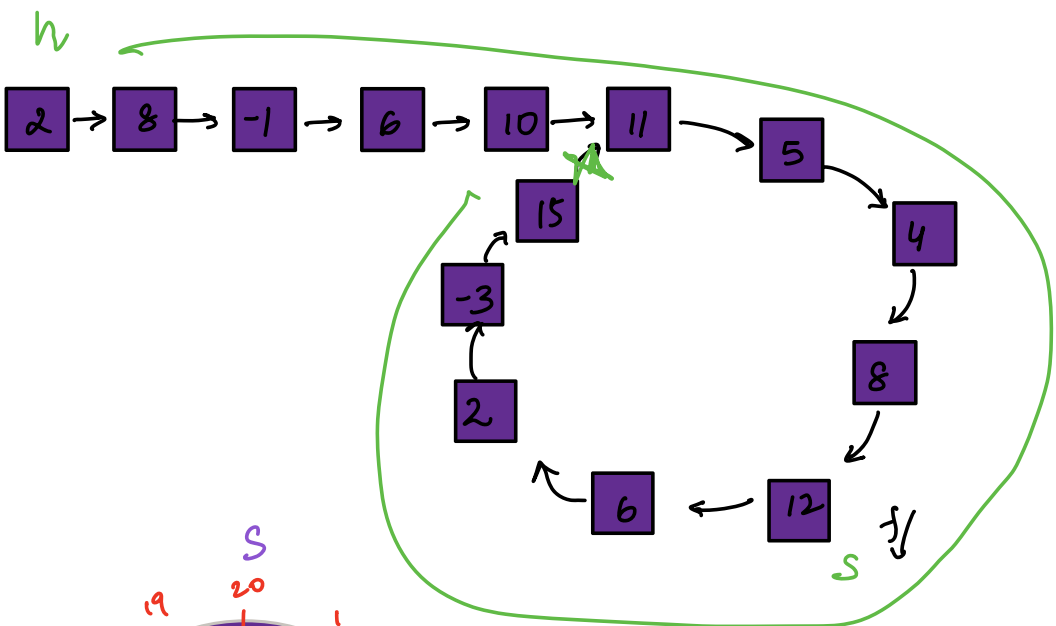
HashSet

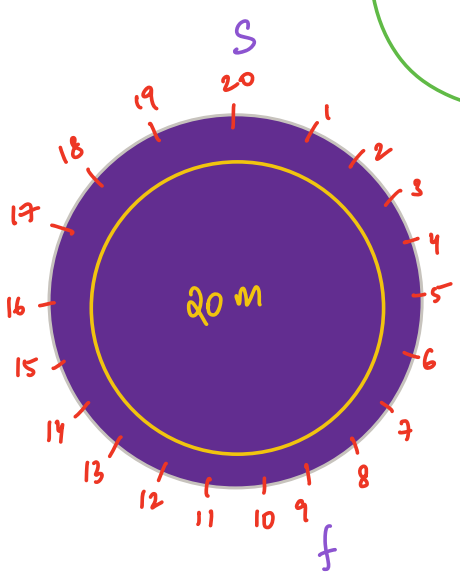
Keep iterating on list & putting nodes in set if not already present
if present return true
if you found a NULL in LL, return f

Detect Cycle

Floyd cycle







h1.next = h1

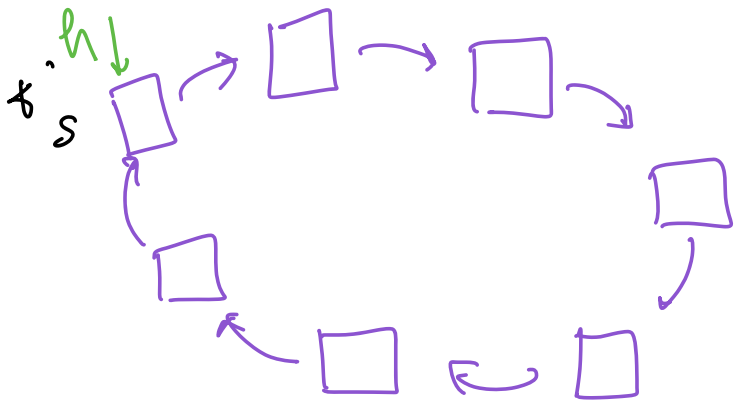
```
s = head
f = head
while(f.next != NULL && f.next.next != NULL){
    s = s.next
    f = f.next.next
    if (s == f) return true
}
return false
```

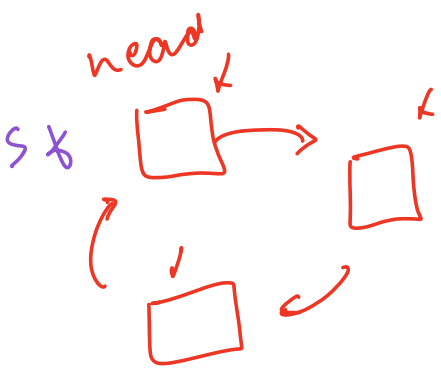
1) Follow up question : Find start of the cycle

put h on head
//s is present where s & f met in the
// cycle
move h & s by 1 step
where they meet is the start of the cycle.

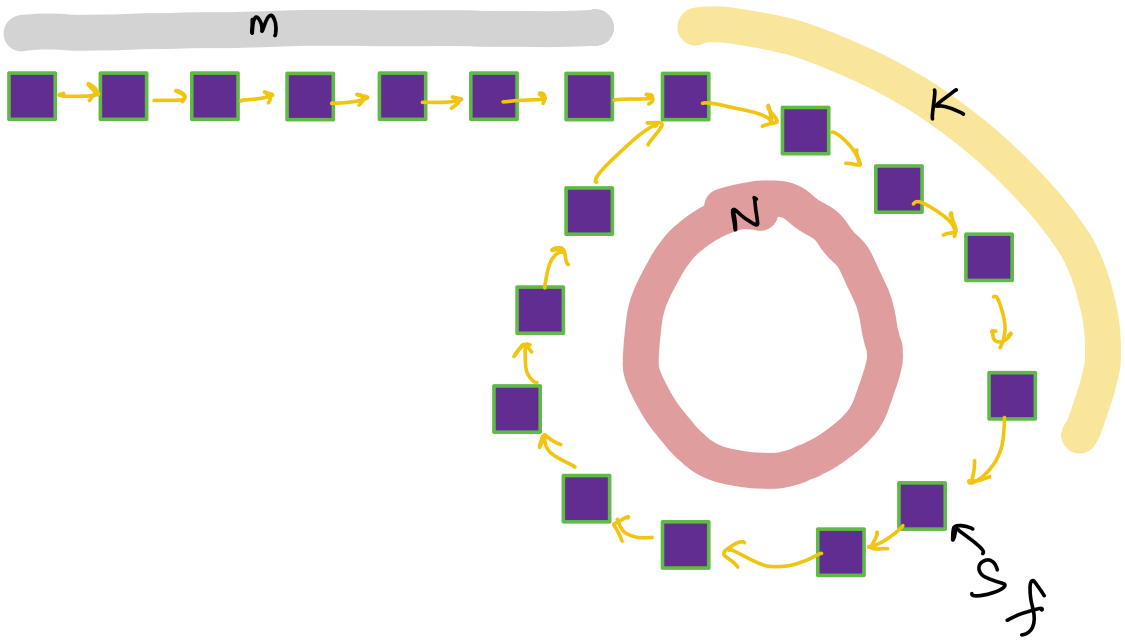
2) Break the cycle

start pointer from start of the cycle & keep moving while h1.next != start is true.


h
f
s


head
f
s f
f
f

Proof



$D_s = M + K + N \times C_1$

$D_f = M + K + N \times C_2$

} constants

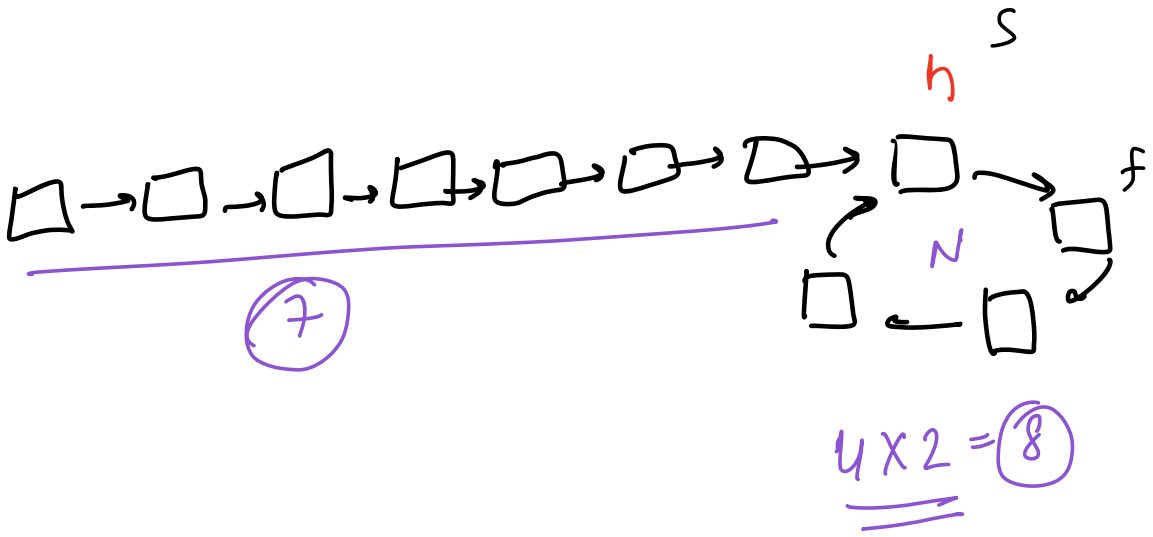
$$D_f = 2 \times D_s$$

$$M + K + N \times C_2 = 2(M + K + N \times C_1)$$

$$M + K + N \times C_2 = 2M + 2K + 2N \times C_1$$

$$K + NC_2 - 2K - 2NC_1 = 2M - M$$

$$N(C_2 - 2C_1) - K = M$$


s
h
f
N
7
4 X 2 = 8