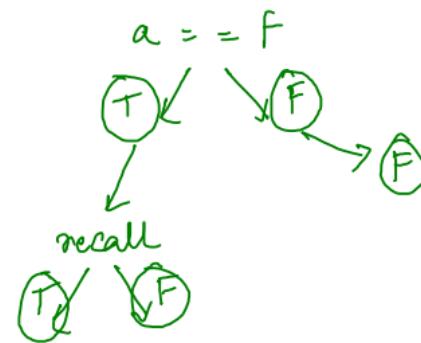
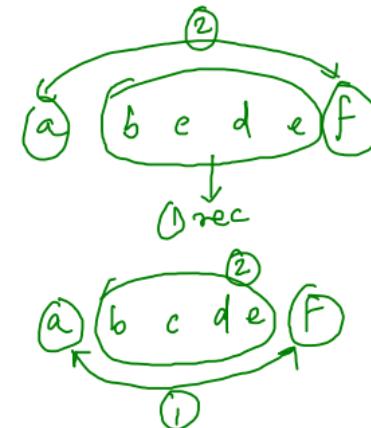


preorder palindrome :

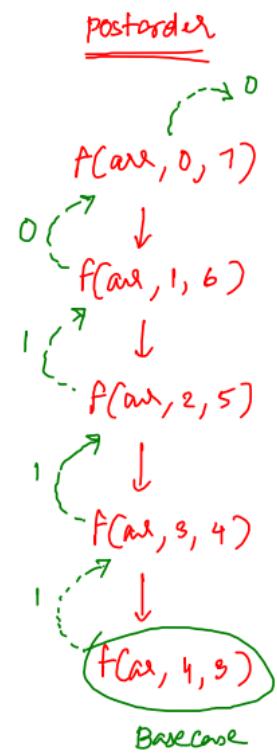
```
function isPalindrome (arr, start, end) {  
    if (start > end) {  
        return true;  
    }  
    if (arr [start] != arr [end]) {  
        return false;  
    }  
    return isPalindrome (arr, start + 1, end - 1);  
}
```



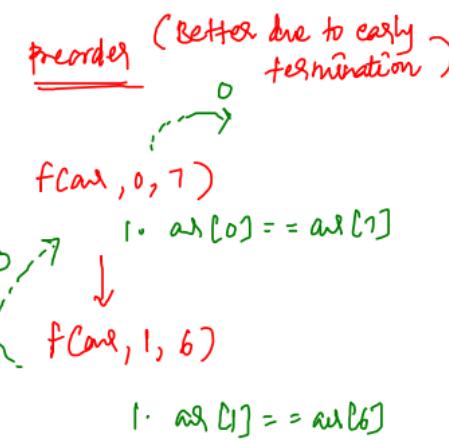
pre vs post in is Palindrome :

```
935 function isPalindromic(arr, start_idx, end_idx) {  
936   if (start_idx > end_idx) {  
937     return 1;  
938   }  
939  
940   const smallAns = isPalindromic(arr, start_idx + 1, end_idx - 1);  
941   if (smallAns == 0) return 0;  
942   return arr[start_idx] == arr[end_idx] ? 1 : 0;  
943 }  
944  
945 // preorder  
946 function isPalindromic(arr, start_idx, end_idx) {  
947   if (start_idx > end_idx) {  
948     return 1;  
949   }  
950  
951   if (arr[start_idx] != arr[end_idx]) {  
952     return 0;  
953   }  
954  
955   return isPalindromic(arr, start_idx + 1, end_idx - 1);  
956 }
```

arr: 5 (4) 3 2 2 3 (5) 5
0 (1) 2 3 4 5 6 7



- * go to depth $s > e$ and while coming back check
- * hence always entire array is traversed.

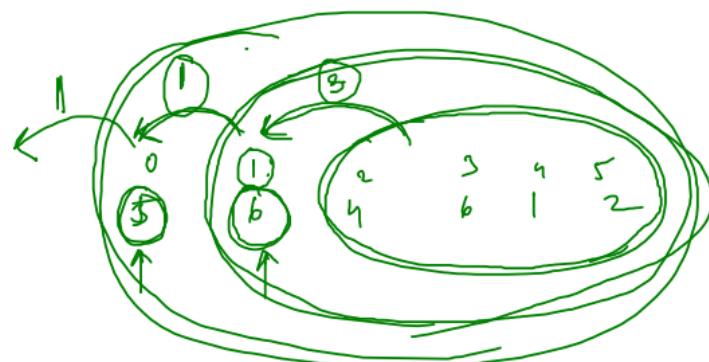


- * this will reach depth $s > e$ only when it is a palindrome. Check and proceed.
- * all elements are traversed/touched only when it is a palindrome

* First occurrence:

ip: 5 6 1 2 3 4 6 1 2 5 , ele = 6

dp: 1



1. faith,

$\Rightarrow \text{firstOccur}(\text{arr}, \text{ele}, ?\text{idx})$

\Rightarrow 1st occurrence of ele in arr
from $?idx \rightarrow n$

2. logic

$\Rightarrow \text{firstOccur}(\text{arr}, 6, 0) \Rightarrow 1$

$\Rightarrow \text{firstOccur}(\text{arr}, 6, 2) \Rightarrow 3$

$\Rightarrow \text{firstOccur}(\text{arr}, 6, 4) \Rightarrow -1$

$\text{smallAns} = \text{firstOccur}(\text{arr}, \text{ele}, ?\text{idx} + 1)$



$\text{arr}[?idx] == \text{ele}$

\downarrow
 $?idx \rightarrow \text{smallAns}$

```

961 function firstIndex(arr, ele, idx) {
962   if (idx == arr.length) {
963     return -1;
964   }
965
966   const smallAns = firstIndex(arr, ele, idx + 1);
967   if (arr[idx] == ele) return idx;
968   return smallAns;
969 }

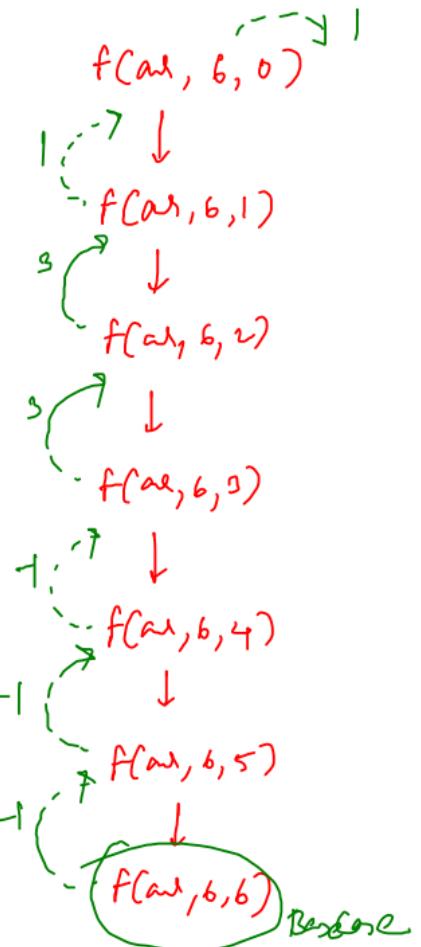
```

0 1 2 3 4 5
 5 6 4 6 1 2

```

972 function firstIndex(arr, ele, idx) {
973   if (idx == arr.length) {
974     return -1;
975   }
976
977   if (arr[idx] == ele) return idx;
978
979   return firstIndex(arr, ele, idx + 1);
980 }

```



preorder (early termination)
 f(arr, 6, 0)
 |
 f(arr, 6, 1)
 |
 f(arr, 6, 2)
 |
 f(arr, 6, 3)
 |
 f(arr, 6, 4)
 |
 f(arr, 6, 5)

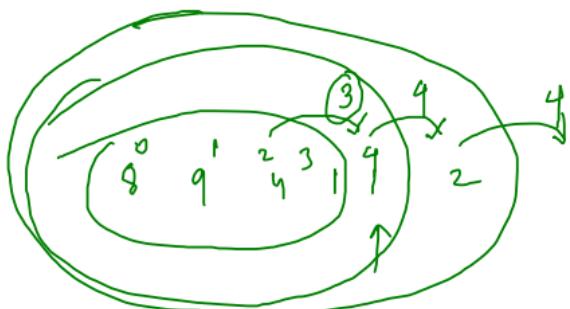
1. arr[0] == 6
 1. arr[1] == 6

* If ele is not present
 it will go until
 base case.

* Last occurrence : (1st occurrence travel in reverse)

ip: 8 9 4 1 1 2 , ele = 1

op: 4



* $\text{idx} = n - 1$

1. faith

$\Rightarrow \text{lastOccur}(\text{arr}, \text{ele}, \text{idx})$

\Rightarrow this will give last occurrence of ele in arr from $0 \rightarrow \text{idx}$

2. $\text{lastOccur}(\text{arr}, 1, 5) \rightarrow 4$

$\text{lastOccur}(\text{arr}, 1, 3) \rightarrow 3$

$\text{lastOccur}(\text{arr}, 1, 0) \rightarrow -1$

smallAns = $\text{lastOccur}(\text{arr}, \text{ele}, \text{idx} - 1)$

$$\begin{array}{c} \swarrow \quad \searrow \\ T = \text{idx} - 1 \end{array} \quad \textcircled{-1}$$

$\text{arr}[\text{idx}] = \text{ele}$

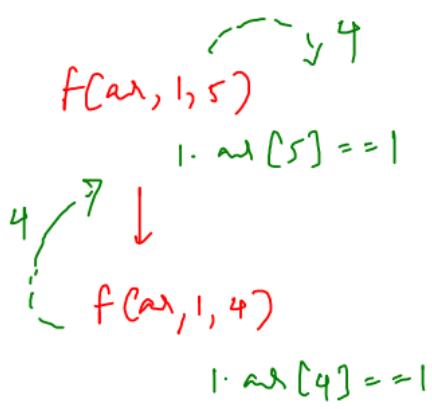
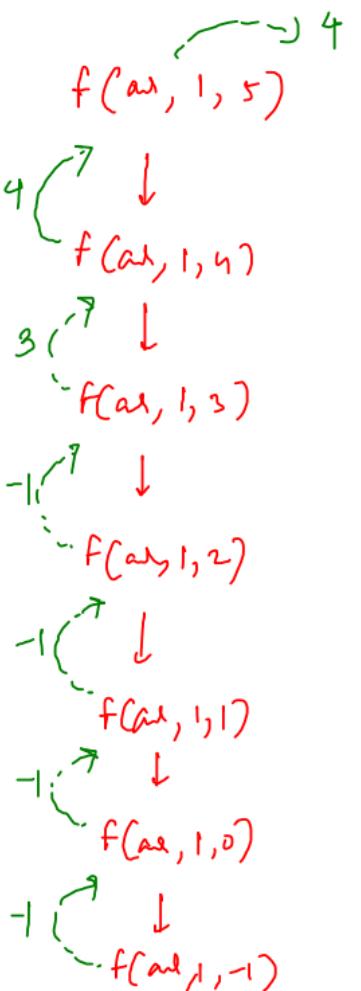
$$\begin{array}{c} \swarrow \quad \searrow \\ \text{idx} \quad \text{smallAns} \end{array}$$

```

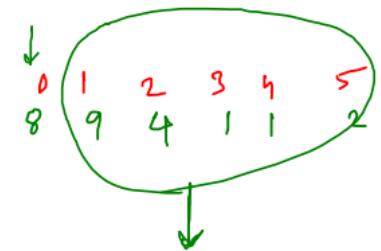
984 // post order
985 function lastIndex(arr, ele, idx) {
986   if (idx == -1) {
987     return -1;
988   }
989
990   const smallAns = lastIndex(arr, ele, idx - 1);
991   if (arr[idx] == ele) return idx;
992   return smallAns;
993 }
994
995 // preorder
996 function lastIndex(arr, ele, idx) {
997   if (idx == -1) {
998     return -1;
999   }
1000
1001   if (arr[idx] == ele) return idx;
1002
1003   return lastIndex(arr, ele, idx - 1);
1004 }

```

$\begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 \\ 8 & 9 & 4 & 1 & 1 & 2 \end{matrix}$



How to do with $\text{idx} = 0$?



$$\text{smallAns} = \text{lastOccur}(\text{arr}, \text{ele}, \text{idx} + 1)$$

$$\begin{aligned} &>= \text{idx} + 1 \\ &-1 \end{aligned}$$

$\{1, 2, 3, 4, 5\}$

$$\begin{aligned} \text{arr}[\text{idx}] &== \text{ele} && \xrightarrow{\text{False}} \text{SA} \\ \text{SA} &= 1 && \\ \text{SA} &= -1 && \\ \text{idx} & \end{aligned}$$

1. $\text{faidx}^*(\text{idx} = 0)$

$\text{lastOccur}(\text{arr}, \text{ele}, \text{idx})$

\rightarrow will give last occurrence of ele in arr from idx $\rightarrow n$

2. $\text{lastOccur}(\text{arr}, 1, 0) \Rightarrow 4$

$\text{lastOccur}(\text{arr}, 1, 4) \Rightarrow 4$

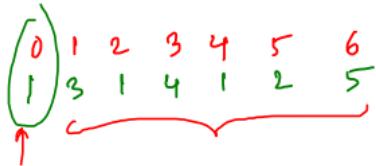
$\text{lastOccur}(\text{arr}, 1, 5) \Rightarrow -1$



```

1006 // postorder with idx = 0
1007 function lastIndex(arr, ele, idx) {
1008     if(idx == arr.length) {
1009         return -1;
1010     }
1011
1012     const smallAns = lastIndex(arr, ele, idx + 1);
1013     if(smallAns != -1) return smallAns;
1014     if(arr[idx] == ele) return idx;
1015     return -1;
1016 }

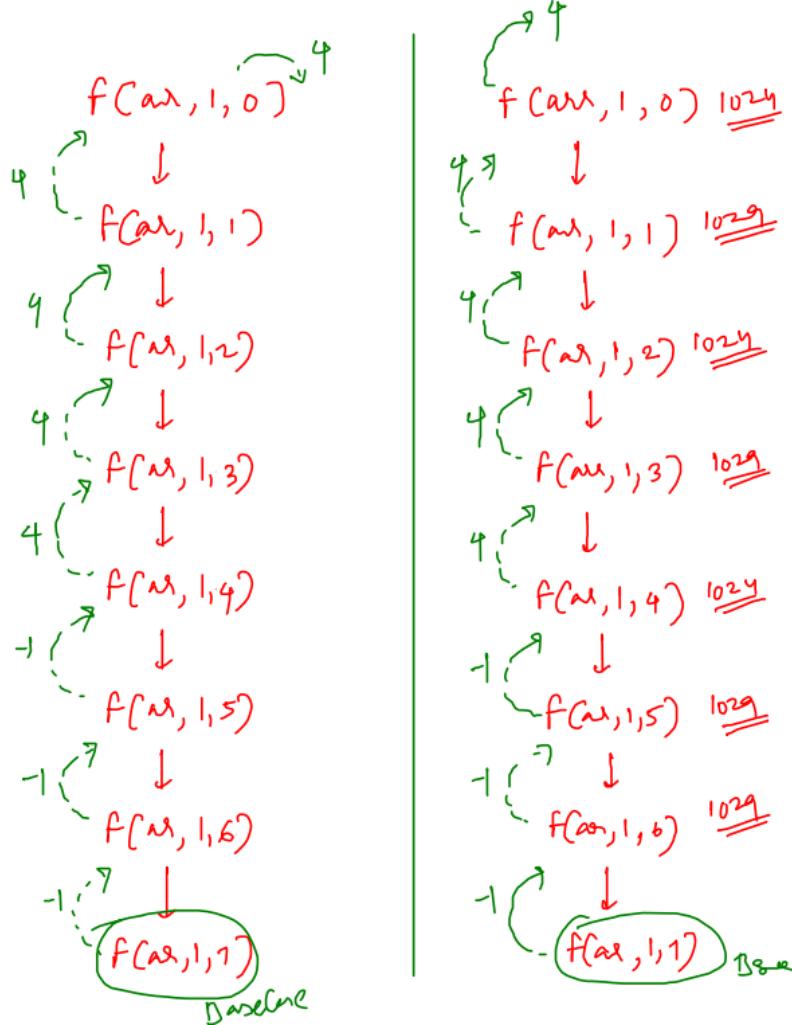
```



```

1018 // preorder with idx = 0
1019 function lastIndex(arr, ele, idx) {
1020     if (idx == arr.length) {
1021         return -1;
1022     }
1023
1024     if (arr[idx] == ele) {
1025         const smallAns = lastIndex(arr, ele, idx + 1);
1026         if (smallAns != -1) return smallAns;
1027         return idx;
1028     }
1029     return lastIndex(arr, ele, idx + 1);
1030 }

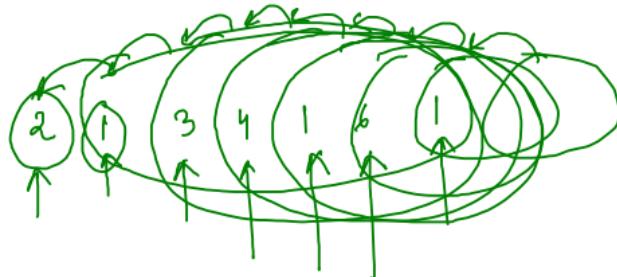
```



* Find All Indices : # preorder

ip: 0 1 2 3 4 5 6
 2 1 3 4 1 6 1 , ele = 1

op: 1 4 6



1. faith

→ printAllIndices(arr, ele, idx)

⇒ all indices of ele in arr
from ?idx → n

2. printAllIndices (arr, 1, 0)

⇒ 1 4 6

printAllIndices (arr, 1, 3)

⇒ 4 6

postorder:

1. faith

\Rightarrow getAllIndices(ar, ele, ^0idx)

\Rightarrow return all indices of ele in ar
from $^0\text{idx} \rightarrow n$

2. getAllIndices(ar, 1, 0)

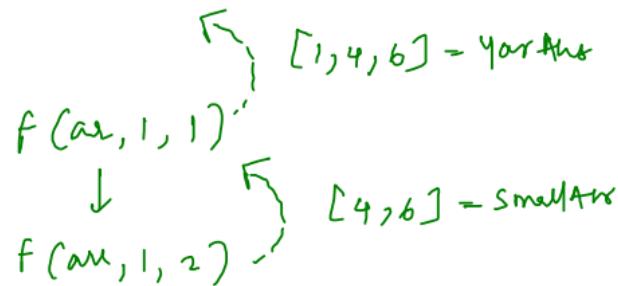
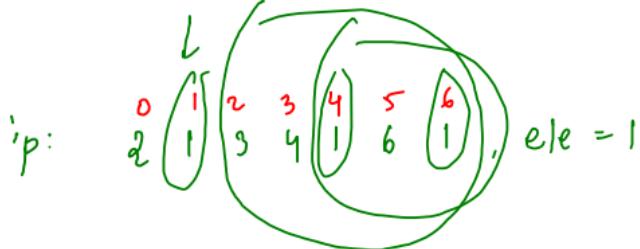
$\Rightarrow [1, 4, 6]$

getAllIndices(ar, 1, 4)

$\Rightarrow [4, 6]$

getAllIndices(ar, 1, 1)

$\Rightarrow []$



```

1035 function printAllIndices(arr, ele, idx) {
1036   if (idx == arr.length) { } op
1037   | return; > col0<del>.log(`>`); op
1038   |
1039
1040   if (arr[idx] == ele) {
1041     process.stdout.write(idx + " "); > op = p > h(m)
1042   }
1043   printAllIndices(arr, ele, idx + 1); op
1044 }

```

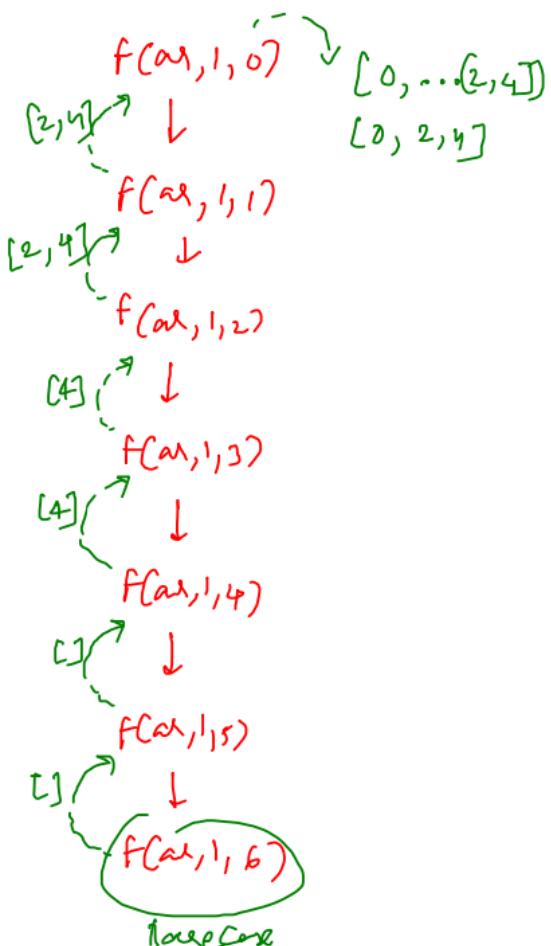
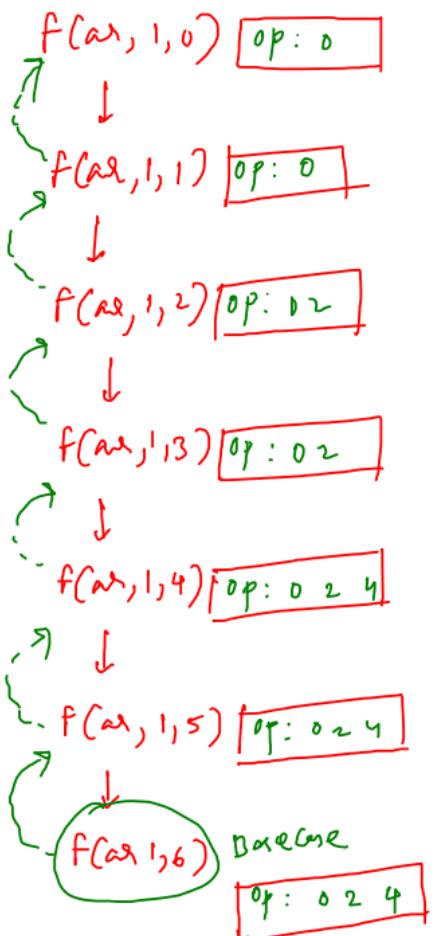
0 1 2 3 4 5
| 2 1 3 | 4

op: 0 2 4

```

1050 // postorder
1051 function getAllIndices(arr, ele, idx) {
1052   if (idx == arr.length) {
1053     return [];
1054   }
1055
1056   const smallAns = getAllIndices(arr, ele, idx + 1); // [4, 6]
1057   if (arr[idx] == ele) return [idx, ...smallAns]; // [1, 4, 6]
1058   return smallAns;
1059 }

```



* N^{th} even Fibonacci Number :

* for every 3 fibonacci numbers you will
get an even fibonacci

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, \dots \quad F(n) = F(n-1) + F(n-2)$$

$$\begin{matrix} 0, & 2, & 8, & 34, & 144, & 610 \\ 1^{\text{st}}, & 2^{\text{nd}}, & 3^{\text{rd}}, & 4^{\text{th}}, & 5^{\text{th}}, & 6^{\text{th}} \\ 1^{\text{st}}, & 2^{\text{nd}}, & 3^{\text{rd}}, & 4^{\text{th}}, & 5^{\text{th}}, & \end{matrix} \quad F_E(n) = 4F_E(n-1) + F_E(n-2)$$

$$\Rightarrow F(3) = 4F(2) + F(1) \quad \Rightarrow F(5) = 4F(4) + F(3)$$
$$\begin{aligned} &= 4(2) + 0 \\ &= 8 \end{aligned} \quad \begin{aligned} &= 4(34) + 8 \\ &= 144 \end{aligned}$$

$$\Rightarrow F(4) = 4F(3) + F(2) \quad \Rightarrow F(6) = 4F(5) + F(4)$$
$$\begin{aligned} &= 4(8) + 2 \\ &= 34 \end{aligned} \quad \begin{aligned} &= 4(144) + 34 \\ &= 610 \end{aligned}$$

* Modulo Arithmetic :

C/C++ / JAVA \rightarrow int \rightarrow 1, 2, 3
 \downarrow
 $2^{31} - 1$, -2^{31}
 (MAX, MIN)

python / JS \rightarrow Number
 \downarrow
 defined int
 $1 \cdot 2$ 1
 $4 \cdot 8$ 2
 large

ans = 2^{32}
 \downarrow
 This will not fit in Integer

$$2^{32} \% (1e9 + 1)$$

$$2 \% (10^9 + 1)$$

$$\begin{aligned} & \% 4 \\ & \{0, 1, 2, 3\} \end{aligned}$$

$$\leq \{0, 10^9 + 6\}$$

* why only $1e9 + 1$? (larger prime)
 \rightarrow wide range of values

return $a + b$;

$$\begin{aligned} ans &= a + b \\ &= (a + b) \% M \end{aligned}$$

$$= (a \% M + b \% M) \% M$$

* $ans \% (1e9 + 1)$