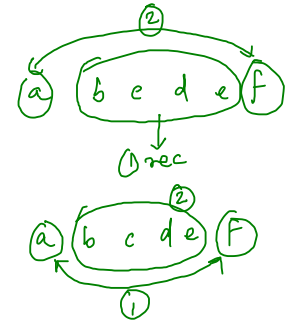
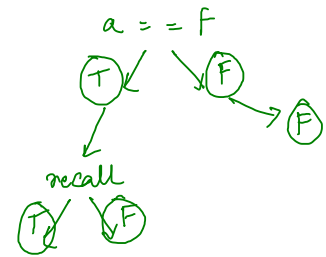
preorder palindrome :

```
function isPalindrome(arr, start, end) {
    if (start > end) {
        return 1;
    }
    if (arr[start] != arr[end]) {
        return 0; (false)
    }
    return isPalindrome(arr, start+1, end-1);
}
```





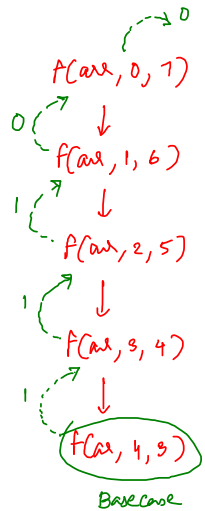
pre vs post in isPalindrome :

```
function isPalindromic(arr, start_idx, end_idx) {
  if (start_idx > end_idx) {
    return 1;
  }

  const smallAns = isPalindromic(arr, start_idx + 1, end_idx - 1);
  if (smallAns == 0) return 0;
  return arr[start_idx] == arr[end_idx] ? 1 : 0;
}

// preorder
function isPalindromic(arr, start_idx, end_idx) {
  if (start_idx > end_idx) {
    return 1;
  }

  if (arr[start_idx] != arr[end_idx]) {
    return 0;
  }

  return isPalindromic(arr, start_idx + 1, end_idx - 1);
}
```

arr: 5 4 3 2 2 3 5 5

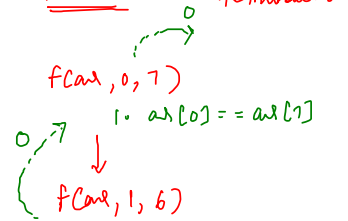
0 1 2 3 4 5 6 7

postorder



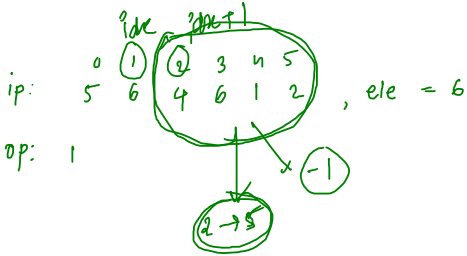
* go to depth s > e and while coming back check
* hence always entire array is traversed.

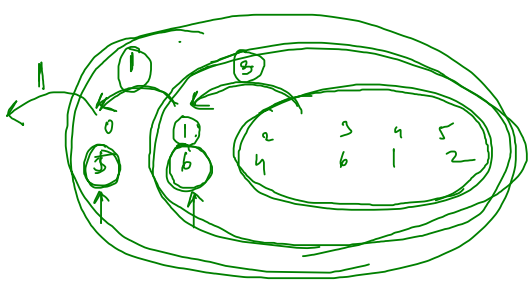
preorder (better due to early termination)



* 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 :





1. faith,

⇒ firstOccur(arr, ele, idx)

⇒ 1st occurrence of ele in arr from idx → n

2. logic

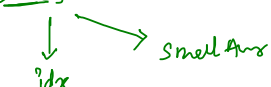
⇒ firstOccur(arr, 6, 0) ⇒ 1

⇒ firstOccur(arr, 6, 2) ⇒ 3

⇒ firstOccur(arr, 6, 4) ⇒ -1

smallAns = firstOccur(arr, ele, idx + 1)

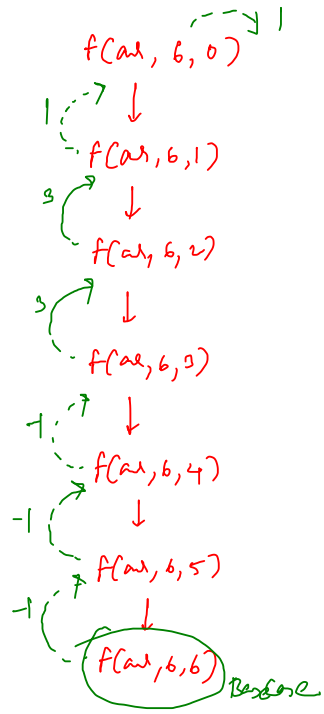
arr[idx] == ele



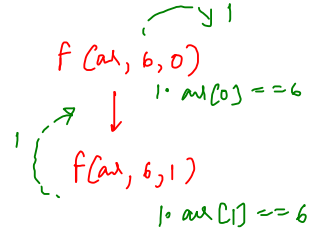
```
961 function firstIndex(arr, ele, idx) {
962   if (idx == arr.length) {
963     return -1;
964   }
965
966   const smallAns = firstIndex(arr, ele, idx + 1);
967   if (arr[idx] == ele) return idx;
968   return smallAns;
969 }
```

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)



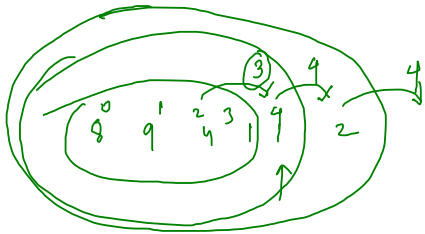
* If ele is not present it will go until base case.

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

	0	1	2	3	4	5	
ip :	8	9	4	1	1	2	, ele = 1

op : 4

★ idx = n-1



1. faith

⇒ lastOccur(arr, ele, idx)

⇒ this will give last occurrence of ele in arr from 0 → idx

e. lastOccur(arr, 1, 5) → 4

lastOccur(arr, 1, 3) → 3

lastOccur(arr, 1, 0) → -1

smallAns = lastOccur(arr, ele, idx-1)

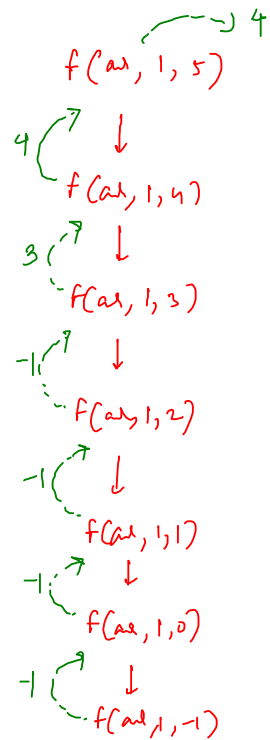
↙ r = idx-1 ↘ -1

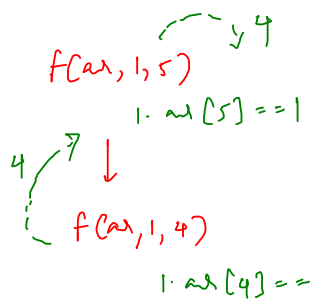
arr[idx] == ele

↙ idx ↘ smallAns

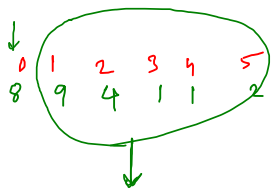
```
// post order
function lastIndex(arr, ele, idx) {
  if (idx == -1) {
    return -1;
  }

  const smallAns = lastIndex(arr, ele, idx - 1);
  if (arr[idx] == ele) return idx;
  return smallAns;
}

// preorder
function lastIndex(arr, ele, idx) {
  if (idx == -1) {
    return -1;
  }

  if (arr[idx] == ele) return idx;

  return lastIndex(arr, ele, idx - 1);
}
```

0	1	2	3	4	5
8	9	4	1	1	2





How to do with idx = 0 ?



smallAns = lastOccur(arr, ele, idx + 1)

-1 0+1 = 1

>= idx+1 : {1,2,3,4,5}

-1

arr[idx] == ele — False → SA

SA != -1 → SA

SA == -1 → idx

1. faith (idx = 0)

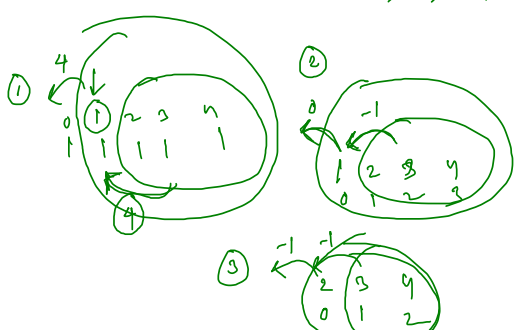
lastOccur(arr, ele, idx)

⇒ will give last occurrence of ele in arr from idx → n

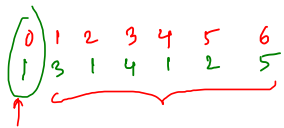
2. lastOccur(arr, 1, 0) ⇒ 4

lastOccur(arr, 1, 4) ⇒ 4

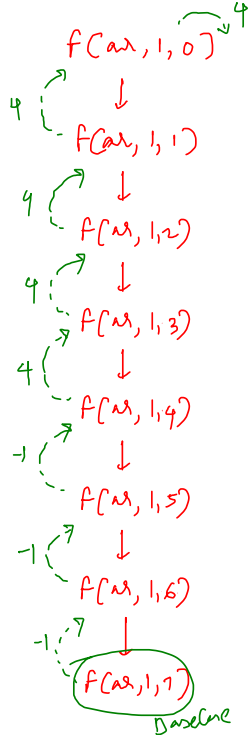
lastOccur(arr, 1, 5) ⇒ -1

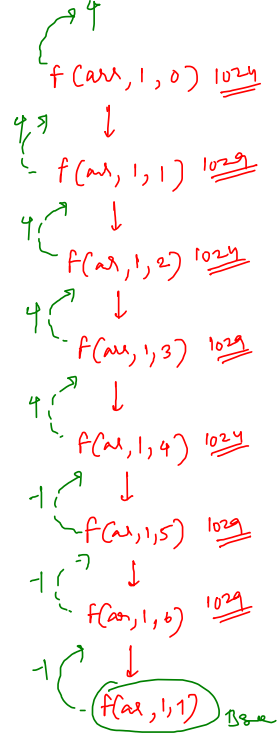


```
// postorder with idx = 0
function lastIndex(arr, ele, idx) {
  if(idx == arr.length) {
    return -1;
  }

  const smallAns = lastIndex(arr, ele, idx + 1);
  if(smallAns != -1) return smallAns;
  if(arr[idx] == ele) return idx;
  return -1;
}
```




```
// preorder with idx = 0
function lastIndex(arr, ele, idx) {
  if (idx == arr.length) {
    return -1;
  }

  if (arr[idx] == ele) {
    const smallAns = lastIndex(arr, ele, idx + 1);
    if (smallAns != -1) return smallAns;
    return idx;
  }
  return lastIndex(arr, ele, idx + 1);
}
```



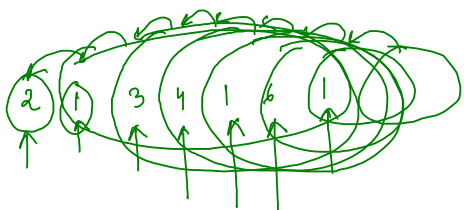


★ Find All Indices :

precedes

```
      0  1  2  3  4  5  6
ip:   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

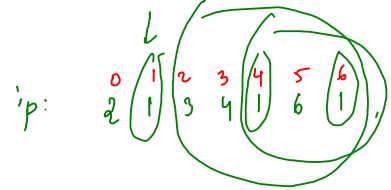
postorder :

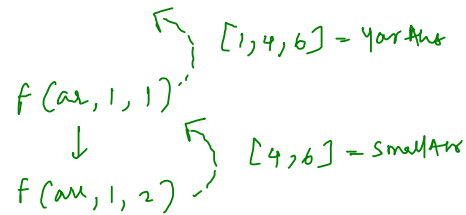
1. faith
   ⇒ getAllIndices(arr, ele, idx)
   ⇒ return all indices of ele in arr from idx → n

2. getAllIndices(arr, 1, 0)
   ⇒ [1, 4, 6]

   getAllIndices(arr, 1, 4)
   ⇒ [4, 6]

   getAllIndices(arr, 1, 7)
   ⇒ []





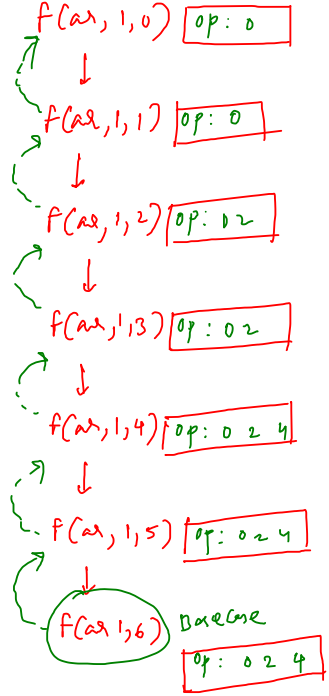
```
function printAllIndices(arr, ele, idx) {
  if (idx == arr.length) {
    return;
  }

  if (arr[idx] == ele) {
    process.stdout.write(idx + " ");
  }
  printAllIndices(arr, ele, idx + 1);
}
```

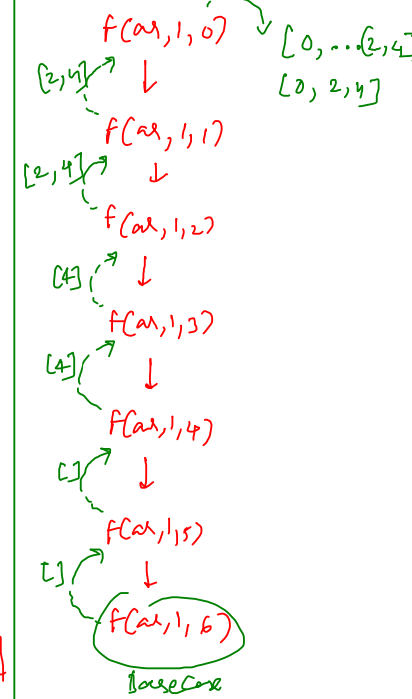
0 1 2 3 4 5

1 2 1 3 1 4

op: 0 2 4

```
// postorder
function getAllIndices(arr, ele, idx) {
  if (idx == arr.length) {
    return [];
  }

  const smallAns = getAllIndices(arr, ele, idx + 1); // [4, 6]
  if (arr[idx] == ele) return [idx, ...smallAns]; // [1, 4, 6]
  return smallAns;
}
```





★ N^{th} Even Fibonacci Number :

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

$F(n) = F(n-1) + F(n-2)$

0, 2, 8, 34, 144, 610

1st 2nd 3rd 4th 5th 6th

1st 2nd 3rd 4th 5th

$F_e(n) = 4F_e(n-1) + F_e(n-2)$

$\Rightarrow F(3) = 4F(2) + F(1)$

$= 4(2) + 0$

$= 8$

$\Rightarrow F(4) = 4F(3) + F(2)$

$= 4(8) + 2$

$= 34$

$\Rightarrow F(5) = 4F(4) + F(3)$

$= 4(34) + 8$

$= 144$

$\Rightarrow F(6) = 4F(5) + F(4)$

$= 4(144) + 34$

$= 610$

* Modulo Arithmetic :

C/C++/JAVA → int → 1, 2, 3

$2^{31}-1$, -2^{31} (MAX, MIN)

python/JS → Number → decimal, int, large

decimal	int
1.2	1
4.8	2

ans = 2^{32}

This will not fit in Integer

2^{32} % $(1e9+7)$

x % (10^9+7)

x % 4

{0, 1, 2, 3}

<= {0, 10^9+6}

ans % M

M = $1e9+7$

return a + b ;

ans = a + b

= (a + b) % M

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

* ans % $(1e9+7)$

* why only $1e9+7$? (larger prime < $2^{31}-1$)

→ wide range of values