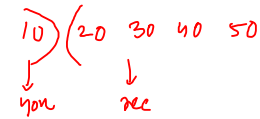
★ print Array Recursively :

	0	1	2	3	4
ip:	10	20	30	40	50
op:	10	20	30	40	50

1. faith
   - ⇒ printArr(arr, idx)
   - ⇒ It prints the array from idx → n

2. logic
   - ⇒ printArr(arr, 0) ⇒ 10 20 30 40 50
   - ⇒ printArr(arr, 1) ⇒ 20 30 40 50



3.
```
function printArr(arr, idx) {
    if(idx == n) {
        return;
    }
    process.stdout.write(arr[idx] + " "); // 10
    printArr(arr, idx + 1); // 20 30 40 50
}
```

```
852  function printArrayRecursive(arr, idx, n) {
853    if (idx == n) {
854      return;
855    }
856
857    process.stdout.write(arr[idx] + " ");
858    printArrayRecursive(arr, idx + 1, n);
859  }
```

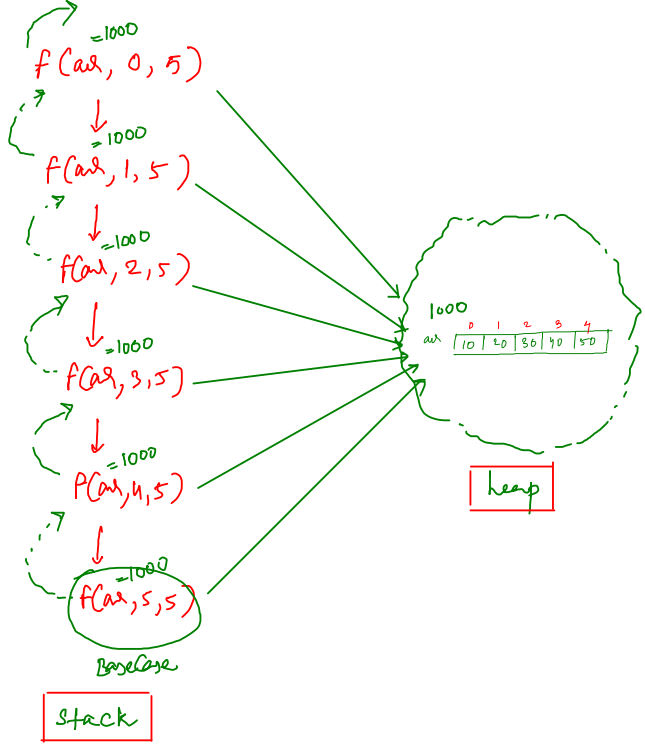
arr.length

	0	1	2	3	4
arr	10	20	30	40	50

Op: 10 20 30 40 50

★ Is "arr" same in every recursive call or is it different for every function ?

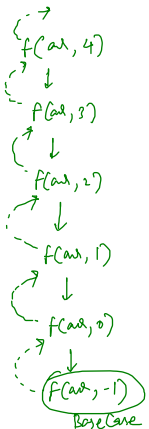
(Same, always are objects hence passed by reference as objects lie in heap where as recursion is in stack)

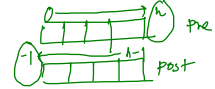


post order way:

```
function printArr(arr, idx) {
    if(idx == -1){
        return;
    }
    printArr(arr, idx-1); // 10 20 30 40
    process.stdout.write(arr[idx]+" ");
}
```

op: 10 20 30 40 50





Print call

10 (20 30 40 50 → preorder

you rec

call print

10 20 30 40 (50 → postorder

rec you

1. faith;

print(arr, idx)

⇒ will print all ele from 0 → idx

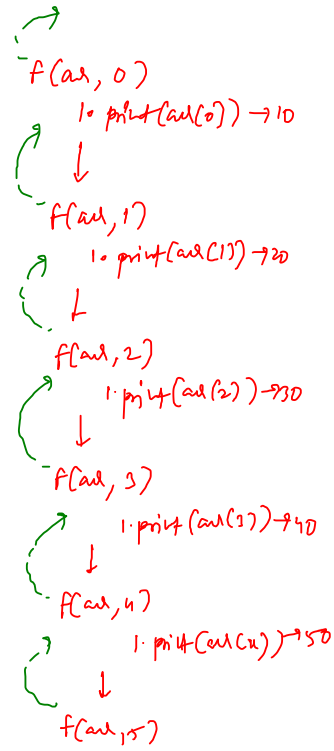
2. print(arr, 4)

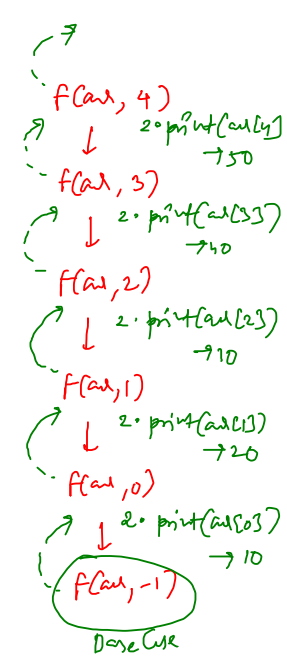
⇒ 10 20 30 40 (50)

Print(arr, 3) ⇒ 10 20 30 40

pre vs post in printArray Normal :

```
852  // preorder
853  function printArrayRecursive(arr, idx, n) {
854    if (idx == n) {
855      return;
856    }
857
858    process.stdout.write(arr[idx] + " ");
859    printArrayRecursive(arr, idx + 1, n);
860  }
861
862  // postorder
863  function printArr(arr, idx) {
864    if (idx == -1) {
865      return;
866    }
867
868    printArr(arr, idx - 1);
869    process.stdout.write(arr[idx] + " ");
870  }
871
872  function printArrayRecursive(arr, i, n) {
873    printArr(arr, n - 1);
874  }
```





★ print Array Reverse :

	0	1	2	3	4
ip :	10	20	30	40	50
op :	50	40	30	20	10

3. function printArrRev(arr, idx) {

```
if(idx == n){
    return;
}

printArrRev(arr, idx+1) // 50 40 30 20
process.stdout.write(arr[idx] + " "); // 10
}
```

1. faith

⇒ printArr(arr, idx)

⇒ It will print, n-1 → idx

2. logic

⇒ printArr(arr, 0)

⇒ 50 40 30 20 10

⇒ printArr(arr, 1)

⇒ 50 40 30 20

50 40 30 20) (10

↓ rec ↓ you

```
878  function printReverseArrayRecursive(arr, i, n) {
879    if (i == n) {
880      return;
881    }
882
883    printReverseArrayRecursive(arr, i + 1, n);
884    process.stdout.write(arr[i] + " ");
885  }
```

op: 50 40 30 20 10

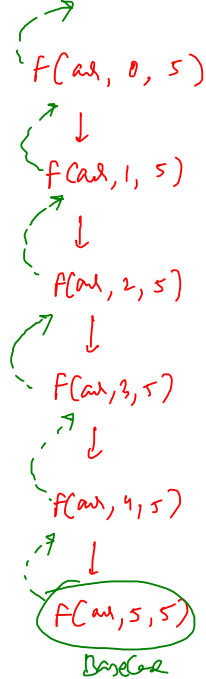
for(i = 0 → n) {
  arr(i)
}

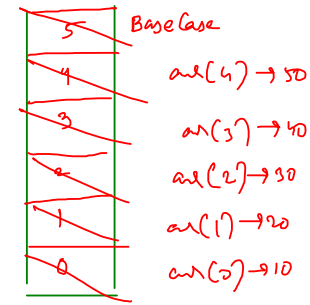
* If you do something before call
  ⇒ forward iteration in arr.

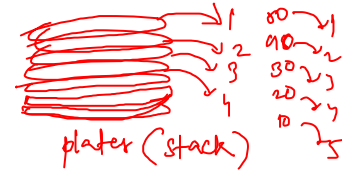
If you do something after call
  ⇒ backward iteration in arr.

for(i = n-1 → 0) {
  arr(i)
}

(Analogy rec vs loop)







pre order way :

```
function printArrRev(arr, idx) {
    if (idx == -1) {
        return;
    }
    process.stdout.write(arr[idx] + " ");
    printArrRev(arr, idx - 1) // 40 30 20 10
}
```

call print

50 40 30 20 10

rec you

print call

50 | 40 30 20 10

you rec

1. faith (arr, n-1)

⇒ printArr(arr, idx)

⇒ all elements from idx → 0

2. printArr(arr, 4)

⇒ 50 40 30 20 10

printArr(arr, 3)

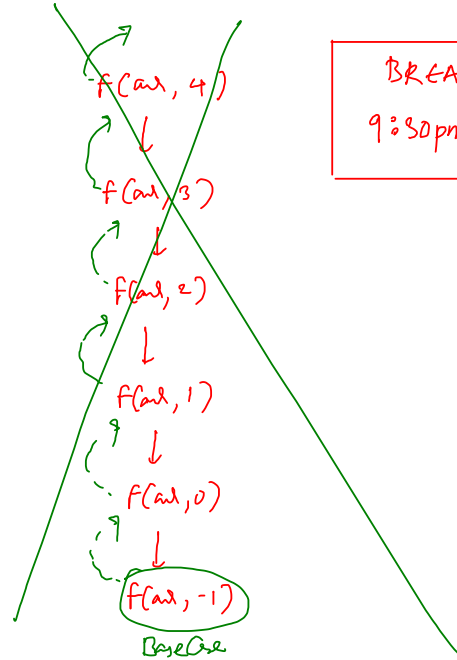
⇒ 40 30 20 10

```
function printArrRev(arr, idx) {
  if (idx == -1) {
    return;
  }

  process.stdout.write(arr[idx] + " ");
  printArrRev(arr, idx - 1);
}

function printReverseArrayRecursive(arr, i, n) {
  printArrRev(arr, n - 1);
}
```

op: 50 40 30 20 10



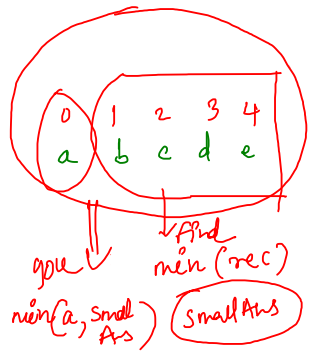
BREAK
9:30pm – 9:45pm

need not think to code → rec will do the magic

★ Find Minimum in array :

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

op : 1



1. faith

⇒ findMin (arr, idx)

⇒ will fetch/get you the min ele

idx → n

2. logic

⇒ findMin (arr, 0)

⇒ 0 → 5 → (1)

⇒ findMin (arr, 1)

⇒ 1 → 5 → (1)

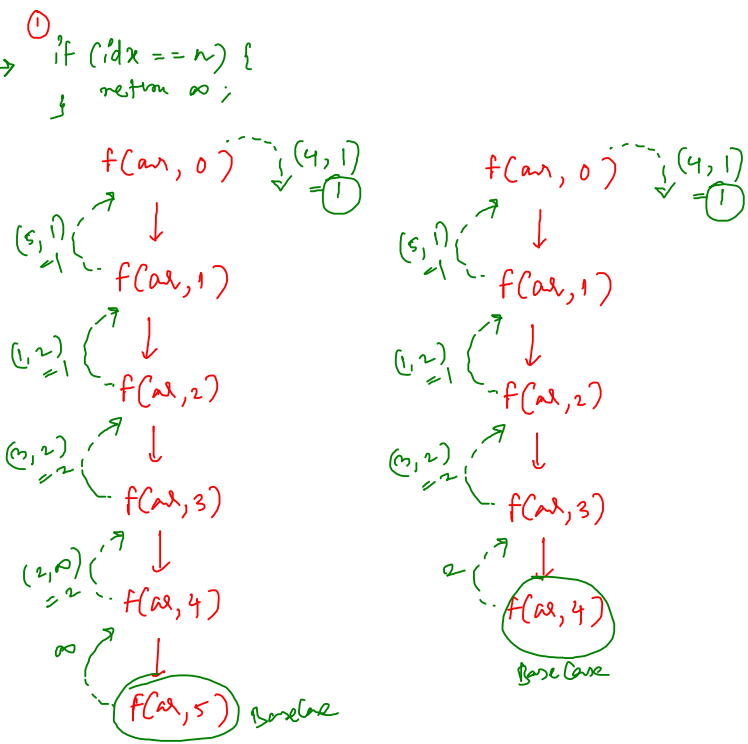
⇒ findMin (arr, 3)

⇒ 3 → 5 → (2)

```
function findMinRec(arr, idx) {
  if (idx == n - 1) {
    return arr[idx];
  }

  const smallAns = findMinRec(arr, idx + 1);
  return Math.min(arr[idx], smallAns);
}

function findMin(arr, n) {
  return findMinRec(arr, 0);
}
```



preorder way :

```
function findMin(arr, idx, curr_min) {
  if(idx == n) { return curr_min; }

  return findMin(arr, idx+1,
                 Math.min(arr[idx],
                          curr_min)
  );
}
```

1. faith

⇒ findMin(arr, idx, curr_min)

⇒ curr_min ⇒ min ele from 0 → idx-1

2.

findMin(arr, 1, 4)
findMin(arr, 2, 4)
findMin(arr, 3, 1)
findMin(arr, 4, 1)
findMin(arr, 5, 1)

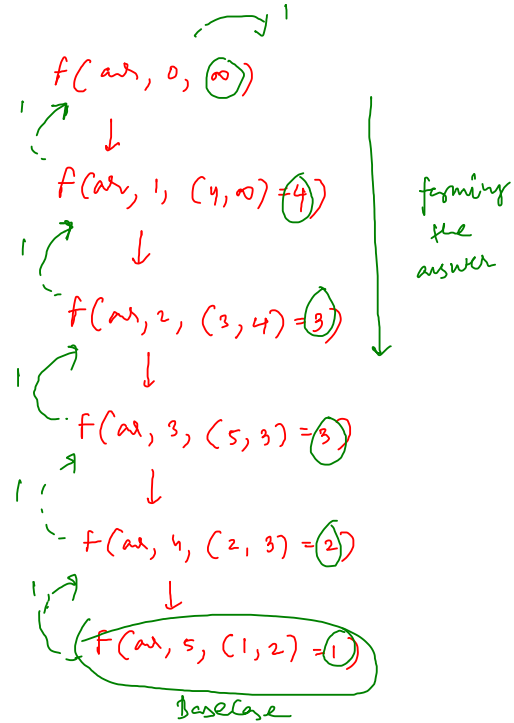
* keep updating curr_min arr rec build

```
// preorder
function findMinRec(arr, idx, curr_min) {
  if (idx == arr.length) {
    return curr_min;
  }

  return findMinRec(arr, idx + 1, Math.min(arr[idx], curr_min));
}

function findMin(arr, n) {
  return findMinRec(arr, 0, Infinity);
  // return findMinRec(arr, 1, arr[0]);
}
```

	0	1	2	3	4
arr:	4	3	5	2	1



★ check arr is a palindrome :

1. faith

⇒ checkPalindrome(arr, start_idx, end_idx)

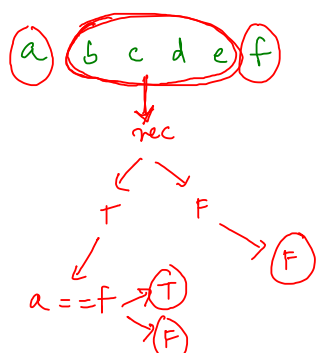
⇒ It will arr[start_idx : end_idx] is a palindrome or not.

ip:

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

2. cp(arr, 0, 5)

cp(arr, 1, 4)



```
934  function isPalindromic(arr, start_idx, end_idx) {
935    if (start_idx > end_idx) {
936      return 1;
937    }
938
939    const smallAns = isPalindromic(arr, start_idx + 1, end_idx - 1);
940    if (smallAns == 0) return 0;
941    return (arr[start_idx] == arr[end_idx]) ? 1 : 0;
942  }
```