

ON A MISSION TO MAKE YOU
LOVE DSA

CRASH CAMP

RECURSION

Introduction



LET'S
THINK
RECURSIVELY

Ruyank: Bhaiyaa can you help me to overcome 'fear' of 'Recursion' pls ?

Bhaiyaa: Okay, tell me what you know about 'Recursion' !

Ruyank: When a function calls itself, it is Recursion

Bhaiyaa: So you just know the definition part, let me help you in visualizing it once.

Ruyank, you are given following array nums[], calculate sum of all elements

0	1	2
2	1	3

Ruyank: Simple, define a variable sum = 0 & iterate from 0th index to last in a loop and do sum += nums[i]

Bhaiyaa: Okay that was pretty simple and also the intuitive solution but can you think of some other approach ?

Ruyank: No bhaiyaa, I can only come up with the iterative solution.

Bhaiyaa: Ruyank, now onwards whatever I say just believe it's true(hypothetical situation) & we will prove it someday but not now

Ruyank: Sure, bhaiyaa !

Bhaiyaa: I'm giving you below array & make a function which returns me sum of this array

0	1	2
2	1	3

Ruyank: I'll make a function `sum1(nums, l, h)` which takes array, `lowest` index & `highest` index & returns sum.
Bhaiyaa, let me write the definition for `sum1()`

Bhaiyaa: Wait wait, don't define it yet there's going to be something magical, just do what I say

Let me take away 1st value of `nums[]` so you are remaining with below array now write a function to calculate sum of array from `idx = 1` to `idx = 2`

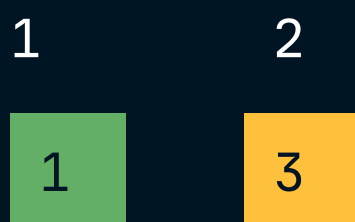
0	1	2
2	1	3

Ruyank: Bhaiyaa, I'll make another function `sum2(nums, l, h)` where I'll pass `nums`, `idx=1`, `idx=2` & returns me the sum

Bhaiyaa: Can you see similarity b/w `sum1()` & `sum2()` that you wrote ?

Ruyank: Yes bhaiyaa, they are taking same `arguments(arr, l, h)` & their working is also same i.e. `calculating sum`.

Bhaiyaa: Great, now let me take out `idx=1` from previous array, so you will make `sum3(nums, 2, 2)` to calculate rest of sum



Ruyank: Now bhaiyaa, if you take away `idx=2` out as well then we aren't remaining with any array, what to do then?



Bhaiyaa: Don't worry about that now, we will see that later
BTW you will make `sum4(arr, 3, 2)` finally and since `l(3) > h(2)`, you can say- you traversed entire array & you can return 0

To give you some idea, your `sum4()` was handling `base case` in terms of `Recursion`, but we will see later.

Ruyank: Bhaiyaa, before it becomes too confusing let's write down all the functions in 1 place

`sum1(nums, 0, 2)`

2 1 3

2

`sum2(nums, 1, 2)`

1 3

1

`sum3(nums, 2, 2)`

3

3

`sum4(nums, 3, 2)`

Bhaiyaa: Ruyank, now let's write definitions for all functions

`sum1(nums, l, h)`

`return nums[0] + sum2(nums, 1, 2)`

`sum2(nums, l, h)`

`return nums[1] + sum3(nums, 2, 2)`

`sum3(nums, l, h)`

`return nums[2] + sum4(nums, 3, 2)`

`sum4(nums, l, h)`

`return 0`

Bhaiyaa: Ruyank, why do we make functions ?

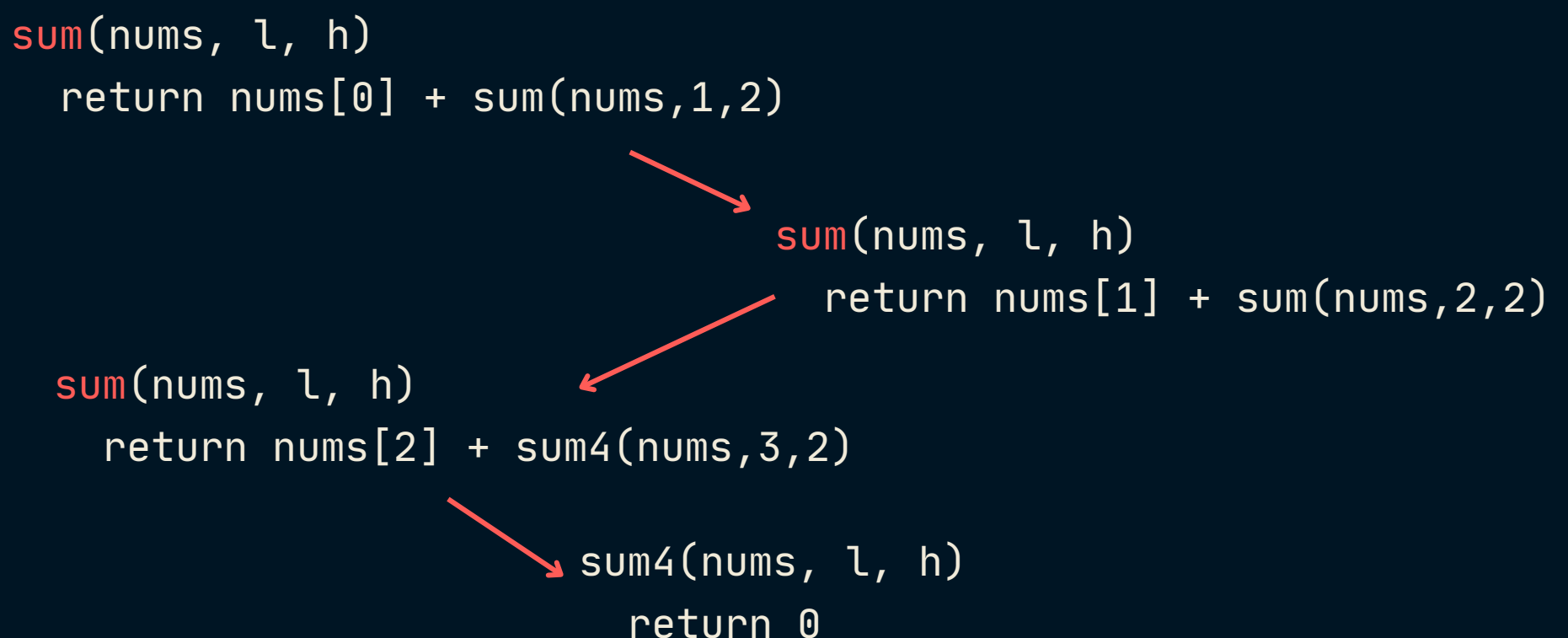
Ruyank: To reuse **same piece of code** & **avoid redundancy**

Bhaiyaa: Can you see in prev. slide `sum1()`, `sum2()`, `sum3()` are doing exactly same work i.e.

-> **taking input of array, low index, high index**

-> **computing sum b/w low & high index**

So, let's make a function `sum(nums, l, h)` which does the same thing that `sum1,2,3` are doing & we will reuse it everytime needed & replace all instances of `sum1,2,3` with `sum()`

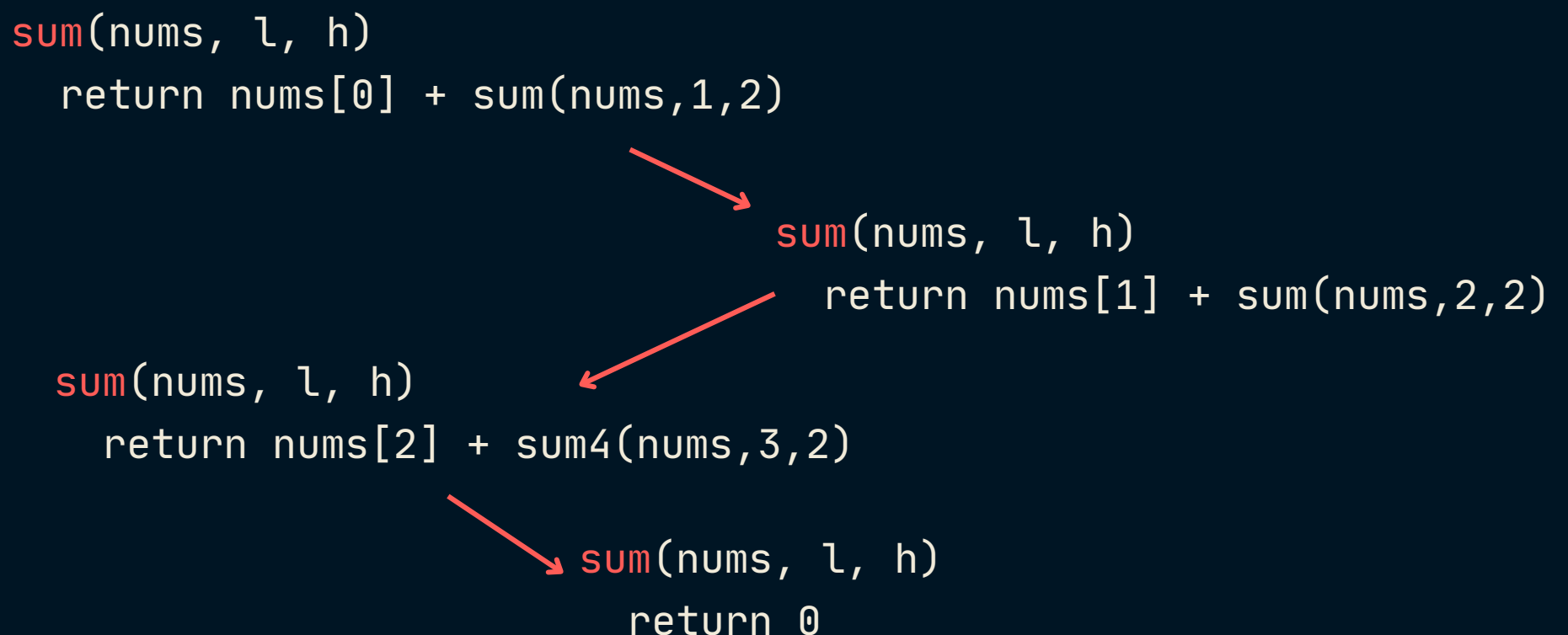


Ruyank: Bhaiyaa, I can see that now instead of writing `sum1()`, `sum2()`, `sum3()` separately their work will be done by `sum()` but **what about `sum4()`** ?

Bhaiyaa: sum4() or final function is used for something special, which we call **base case** & can you tell when will we encounter base case ?

Ruyank: When our test case **crosses it's valid boundary**, here it was valid till $l \leq h$ & as soon as l becomes $> h$, we know we are done with array so we just did
return 0; // when nothing left in array sum = 0

great bhaiyaa, I can observe that we can replace sum4() from sum() just with below condition
if($l > h$)
return 0;



Bhaiyaa: Ruyank, can you see above diagram now and get some idea that what exactly we did ?

Ruyank: Earlier we used `sum1()`, `sum2()`, `sum3()`, `sum4()` to calculate the sum of array but now we just have `sum()` & we are reusing it all the time

Bhaiyaa: We are `resuing` it or I can say we our `sum()` is `calling to itslef` all the time and that's what we call **RECURSION**

Ruyank: Great Bhaiyaa, so their is nothing to fear about recursion, either we can make multiple new functions or we can just reuse same function which is recursion.

Bhaiyaa: Ruyank, there is still a long long journey in mastering recursion & in coming days we will get to know about following stuff

- > Induction, Base, Hypothesis approach
- > Choice Diagram approach
- > Different ways of creating Induction
- > Getting recurrence relation
- > Handling base cases etc...

Ruyank: Bhaiyaa, atleast now I know what 'Recursion' exaclty is all about & I hope in coming days you will help me master that properly

2 1 3

```
sum1(nums, l, h)
  return nums[0] + sum2(nums,1,2)
```

```
sum2(nums, l, h)
  return nums[1] + sum3(nums,2,2)
```

```
sum3(nums, l, h)
  return nums[2] + sum4(nums,3,2)
```

```
sum4(nums, l, h)
  return 0
```

Either you can make many new functions()

```
sum(nums, l, h)
  return nums[0] + sum(nums,1,2)
```

```
sum(nums, l, h)
  return nums[1] + sum(nums,2,2)
```

```
sum(nums, l, h)
  return nums[2] + sum(nums,3,2)
```

```
sum(nums, l, h)
  return 0
```

OR

```
sum(nums, l, h)
  if(l > h) return 0
  return nums[l] + sum(nums,l+1,h)
```

Or you can reuse one function all the time

I know I took way more time to explain a simple concept but as a beginner it becomes tough to actually visualize what exactly is recursion & then many times you see people telling to believe in your function() it will do it's work just handle your induction step.

Now all these hypothetical things mess around in mind so I just tried to go from scratch & I hope you would have got something new in Recursion

There is much more exciting things in Recursion I'll be posting but those things will be possible only if you like, comment and share these posts