Data Structures and Algorithms

Episode 8

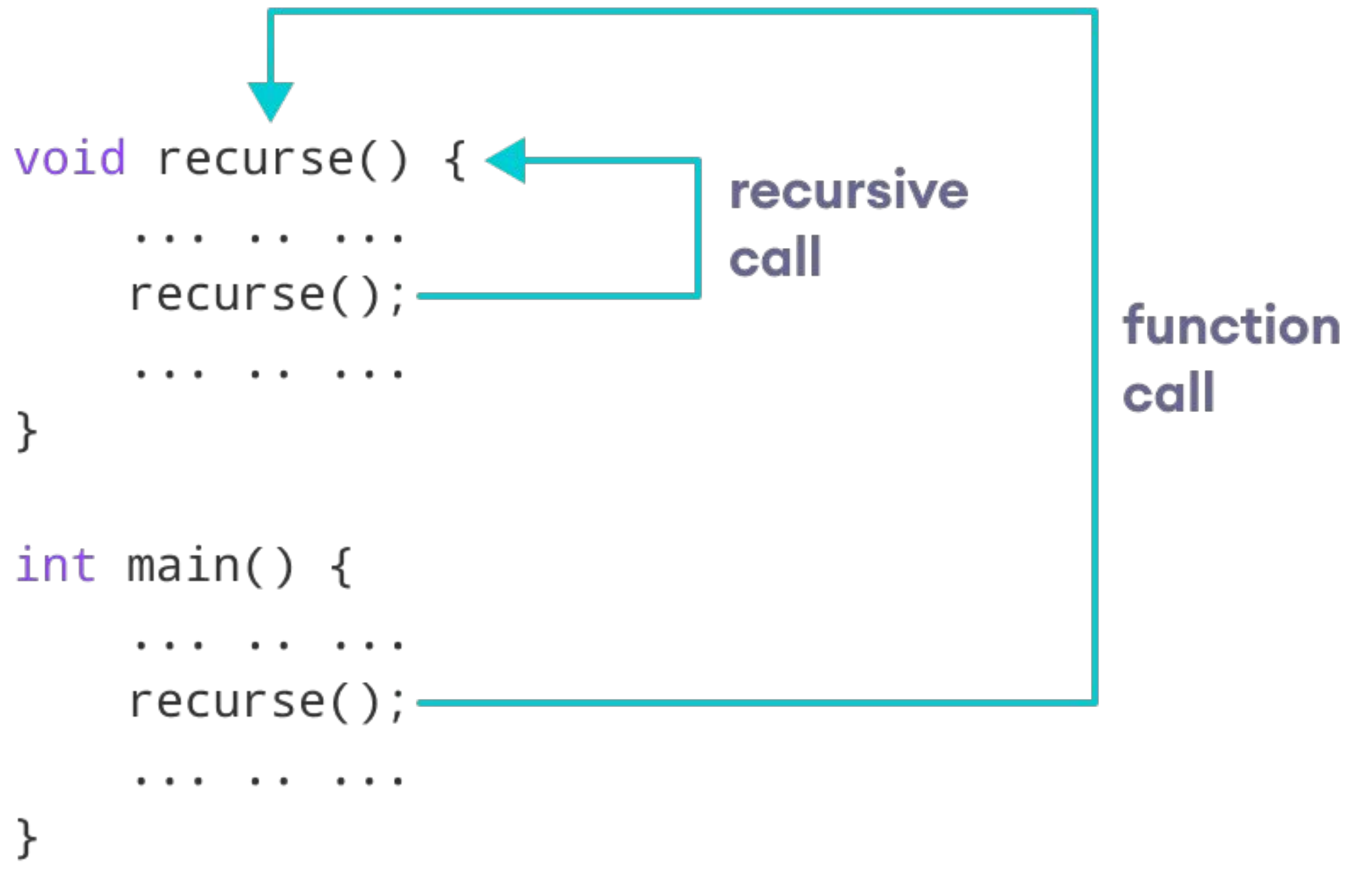
RECURSION AND SEARCH ALGORITHMS

Author: Ahmed Khaled

Recursive Functions

- A recursive function is a **function that calls itself**
  - With each recursive call, the problem size is reduced
  - The function terminates when **a terminal condition is reached**
- Recursion must terminate; otherwise, we have infinite recursion
- A recursive definition consists of two parts.
  - The first part is called the anchor or the base case.
  - In the second part, rules are given that allow for the construction of new elements out of basic elements.

Recursive Functions



```
void recurse() {
    ... .. ...
    recurse();
    ... .. ...
}

int main() {
    ... .. ...
    recurse();
    ... .. ...
}
```

- recursiveFunction:

```
    if (base case){ // there could be more than one base case
        compute the solution without recursion
    }
    else {
        break the problem into sub-problem(s) of the same form and call the recursive function on each sub-problem

        Assemble the results of the sub-problems
    }
```

Three <u>Musts</u> for Recursion

- 1. Your code must have a valid case for each input
- 2. Your code must have at least one base case
- 3. When making a recursive call, it must be a simpler instance that progresses towards the base case

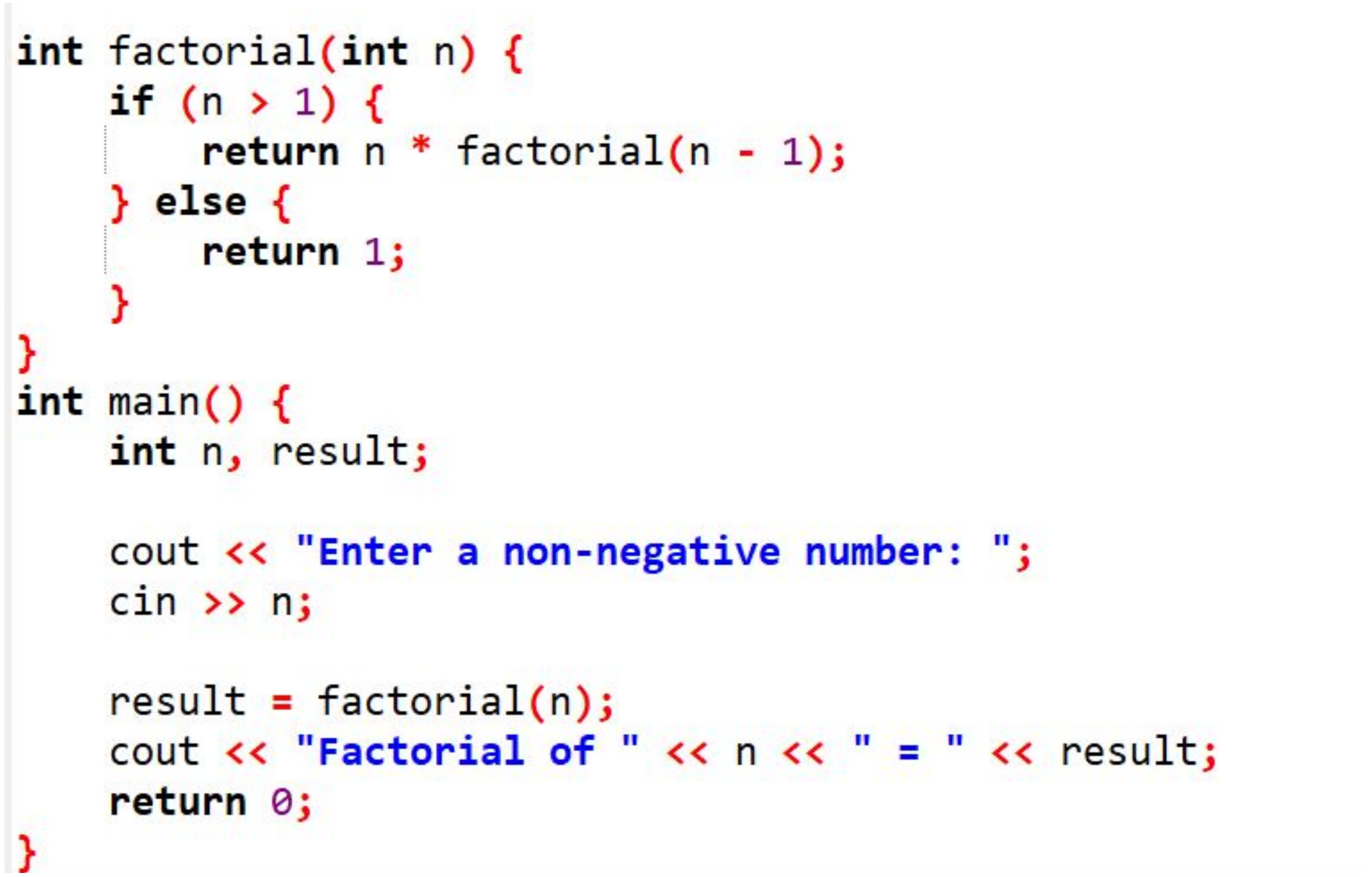
The factorial function !

- The factorial function ! , can be defined in the following manner:

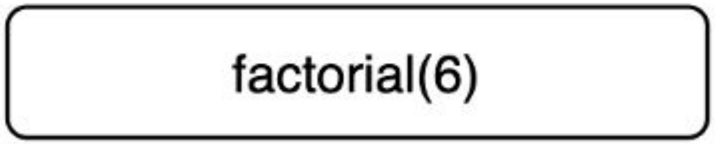
$$n! = \begin{cases} 1 & \text{if } n = 1 \text{ (anchor)} \\ n \cdot (n-1)! & \text{if } n > 1 \text{ (inductive step)} \end{cases}$$

- 6! = 6*5*4*3*2*1=720

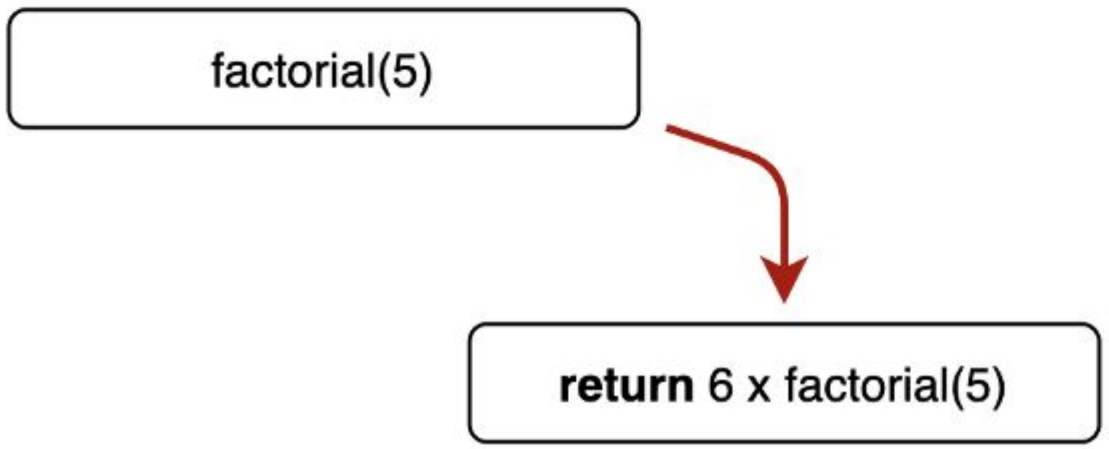
The factorial function !



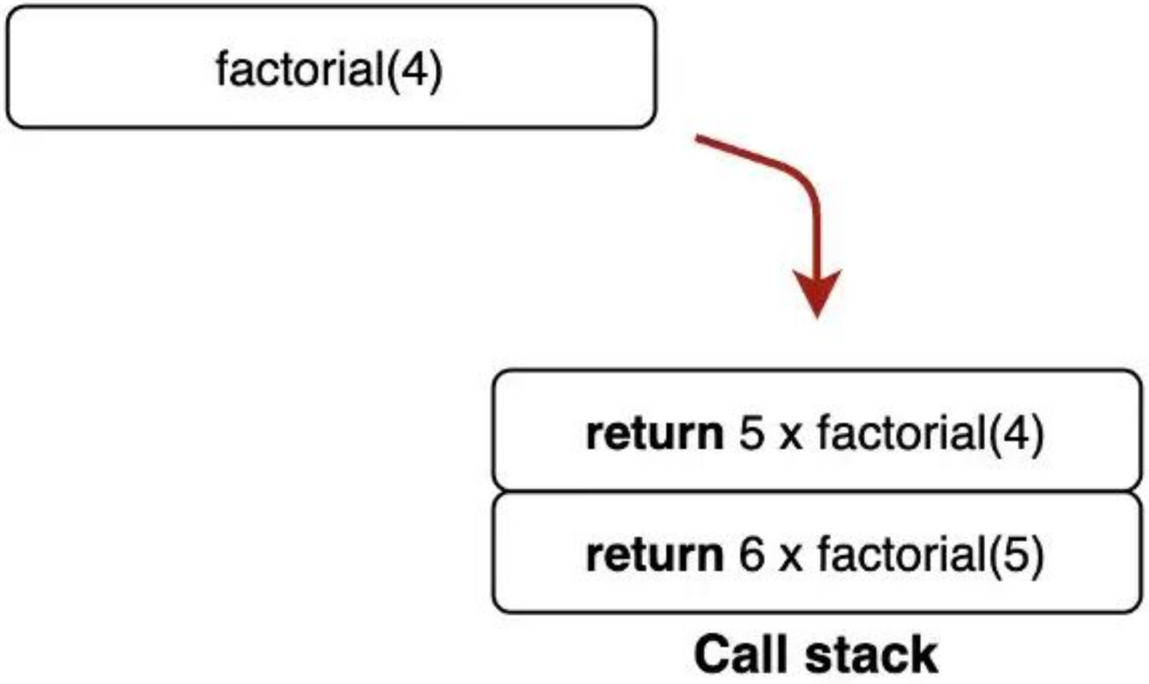
```
int factorial(int n) {
    if (n > 1) {
        return n * factorial(n - 1);
    } else {
        return 1;
    }
}
int main() {
    int n, result;

    cout << "Enter a non-negative number: ";
    cin >> n;

    result = factorial(n);
    cout << "Factorial of " << n << " = " << result;
    return 0;
}
```

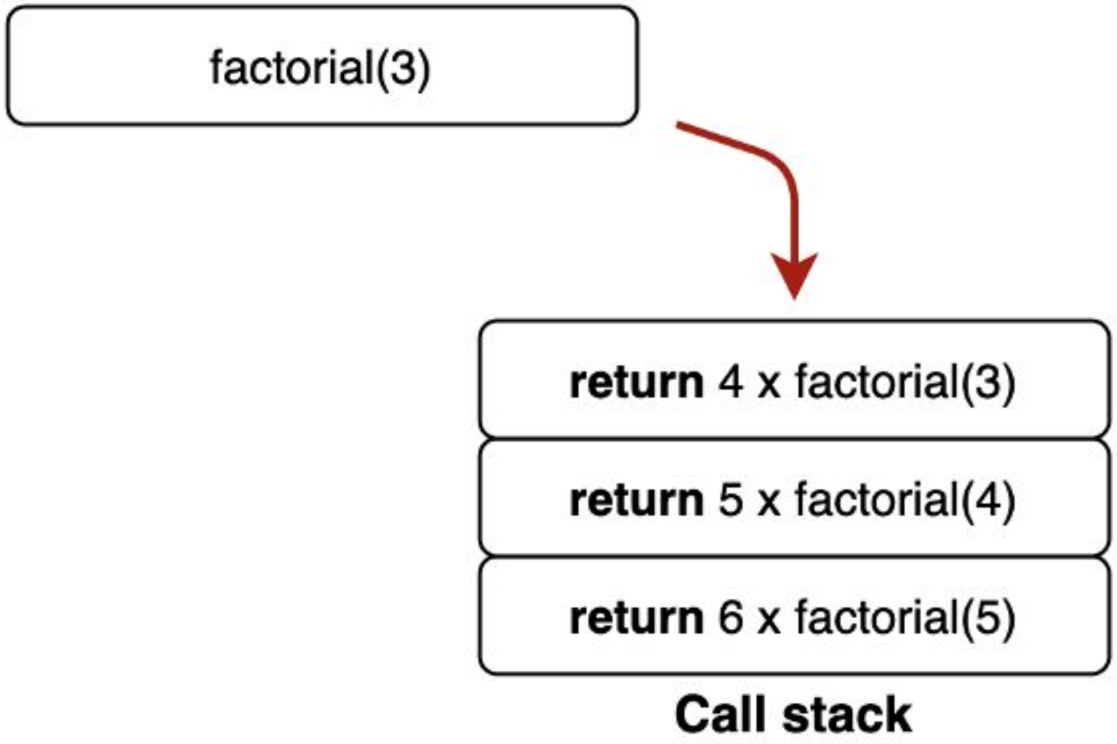


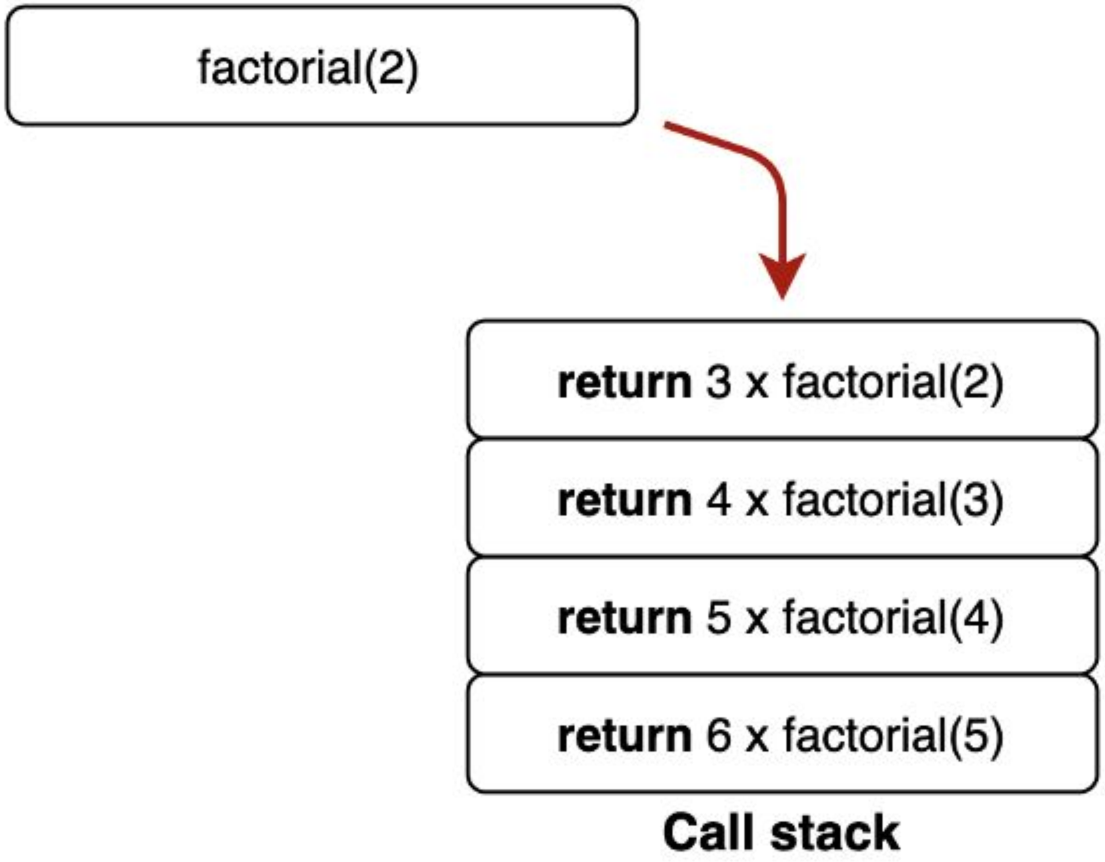
Call stack

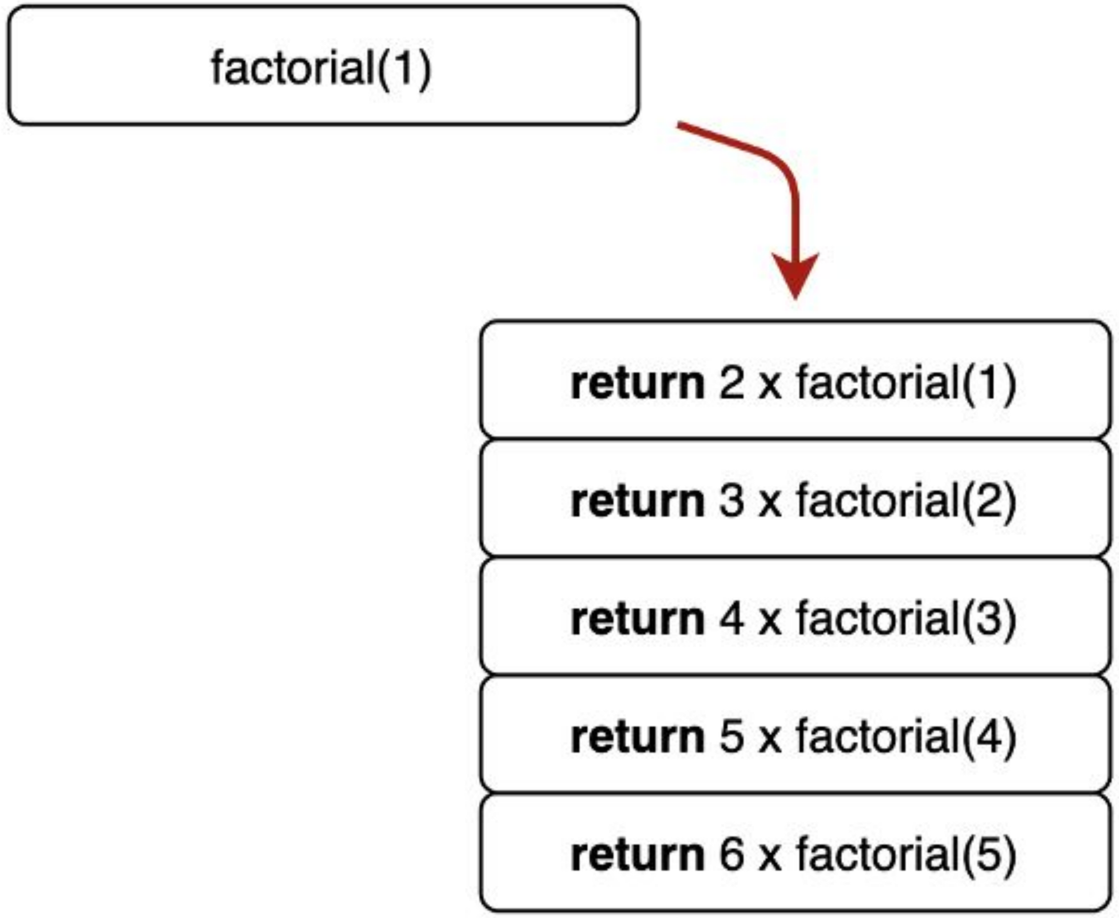


Call stack

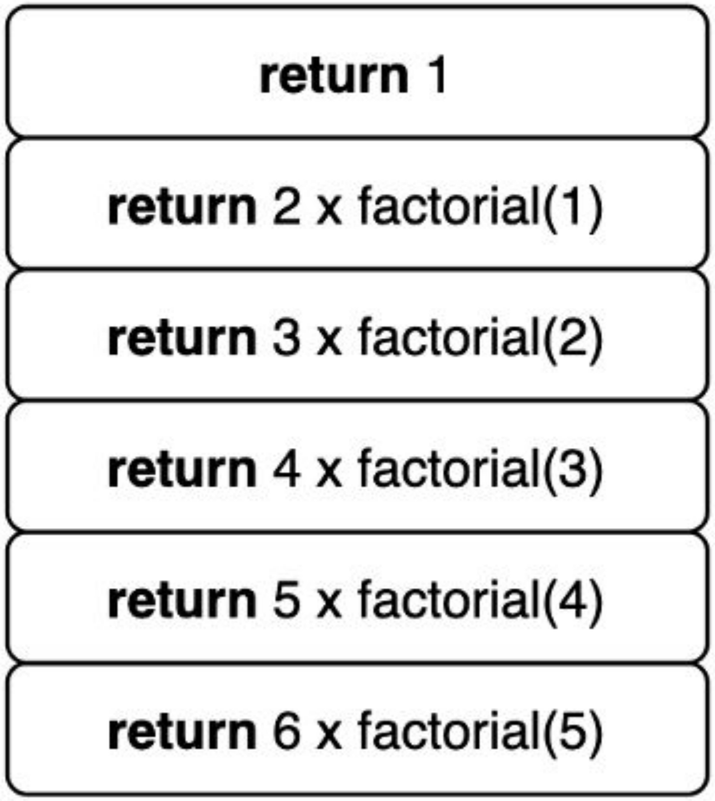

factorial(4)
return 5 x factorial(4)
return 6 x factorial(5)
Call stack


factorial(3)
return 4 x factorial(3)
return 5 x factorial(4)
return 6 x factorial(5)
Call stack

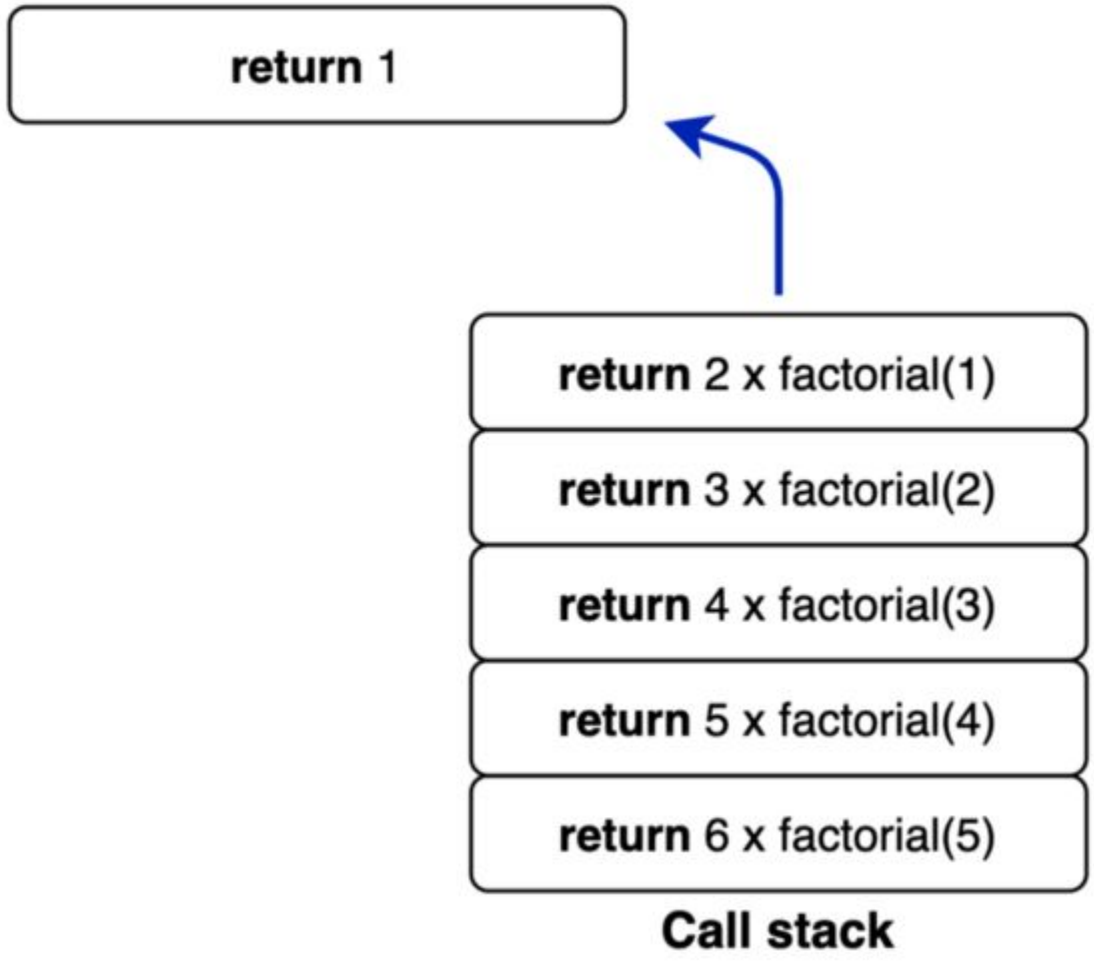

factorial(2)
return 3 x factorial(2)
return 4 x factorial(3)
return 5 x factorial(4)
return 6 x factorial(5)
Call stack

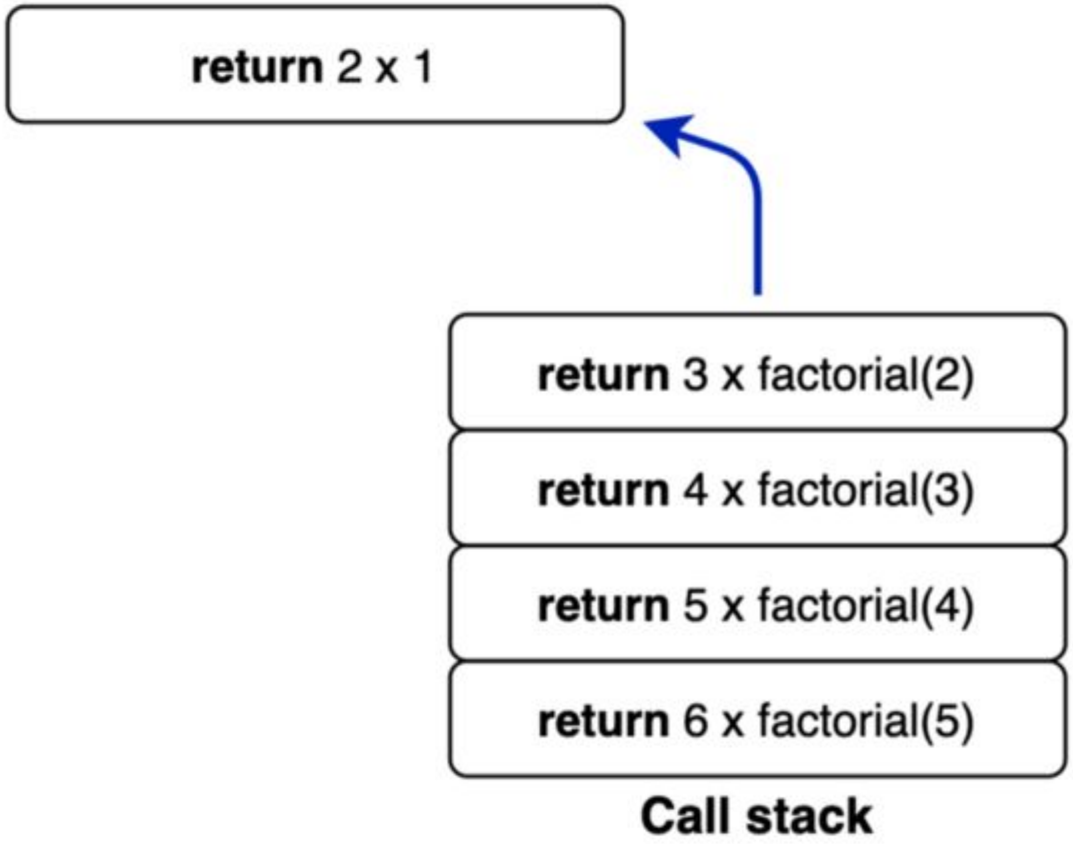

factorial(1)
return 2 x factorial(1)
return 3 x factorial(2)
return 4 x factorial(3)
return 5 x factorial(4)
return 6 x factorial(5)

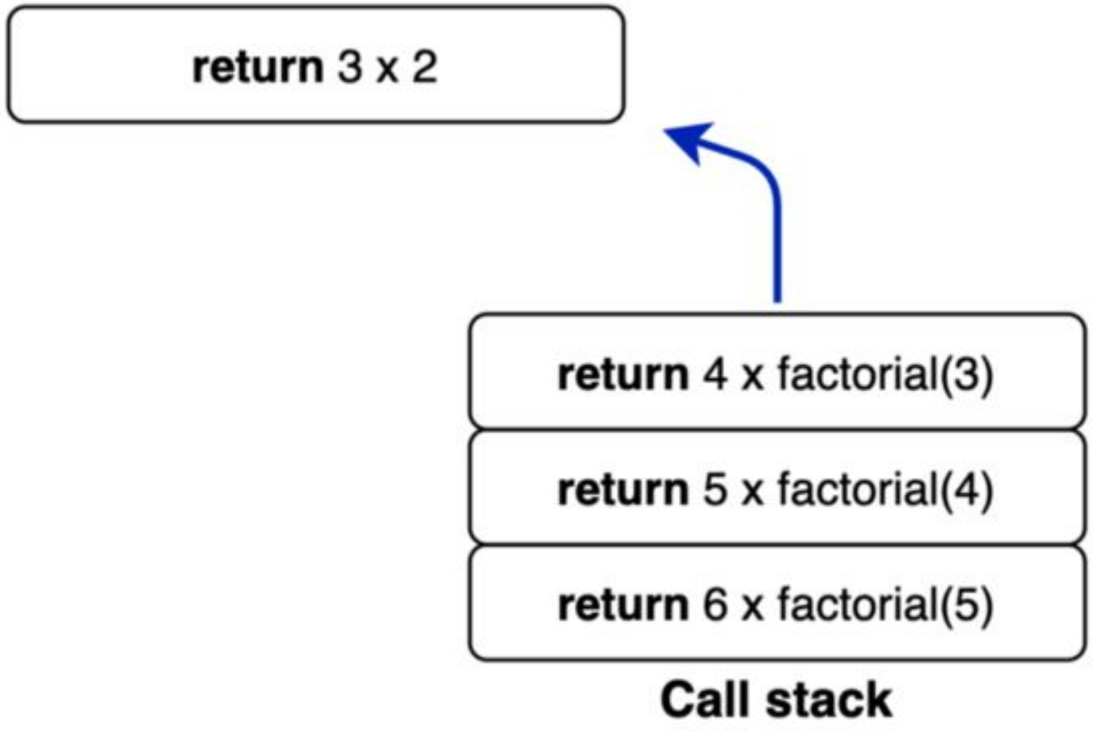
Call stack

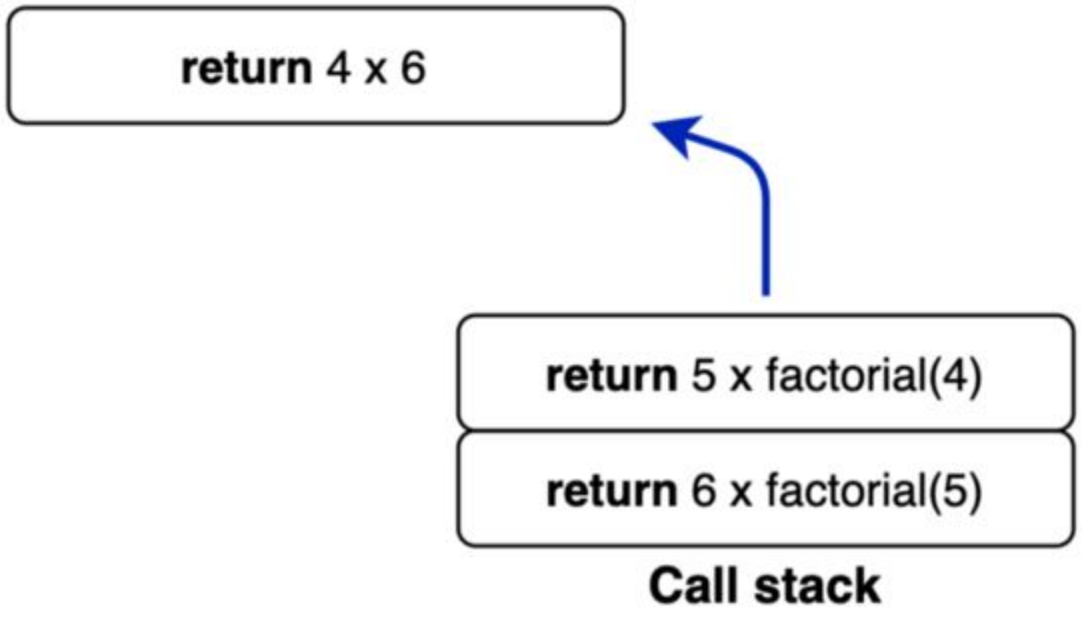

return 1
return 2 x factorial(1)
return 3 x factorial(2)
return 4 x factorial(3)
return 5 x factorial(4)
return 6 x factorial(5)

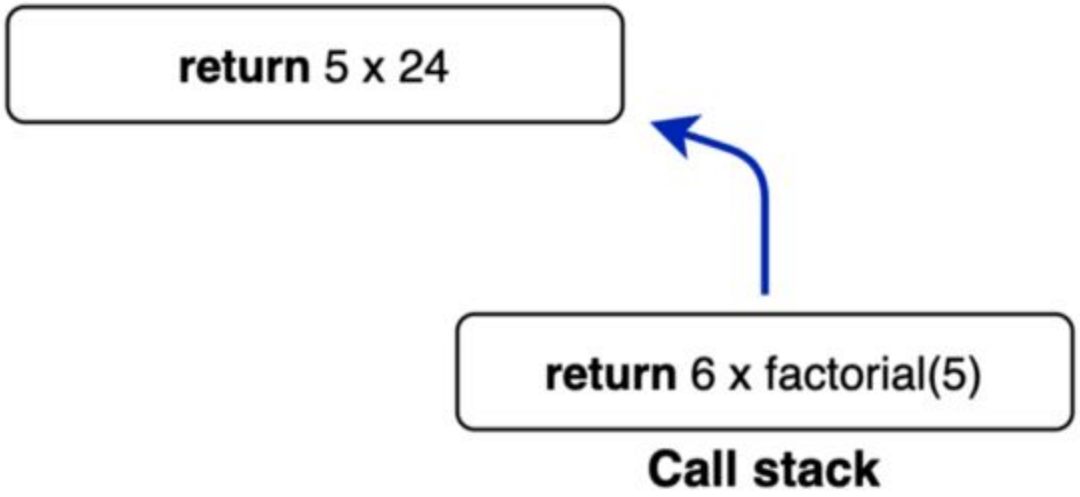
Call stack

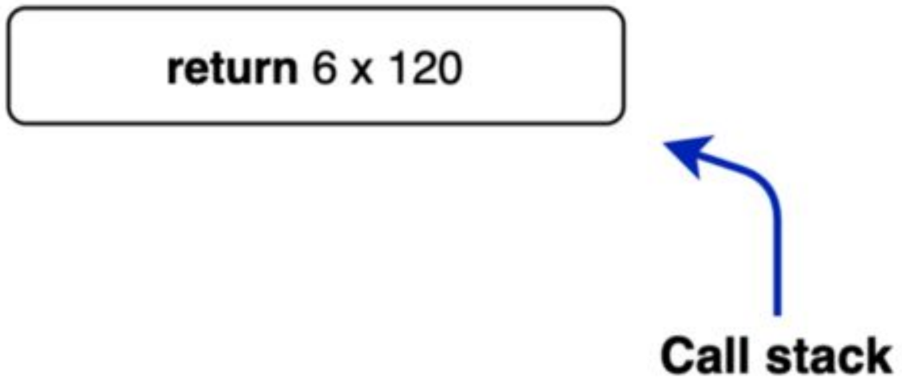

return 1
return 2 x factorial(1)
return 3 x factorial(2)
return 4 x factorial(3)
return 5 x factorial(4)
return 6 x factorial(5)
Call stack


return 2 x 1
return 3 x factorial(2)
return 4 x factorial(3)
return 5 x factorial(4)
return 6 x factorial(5)
Call stack

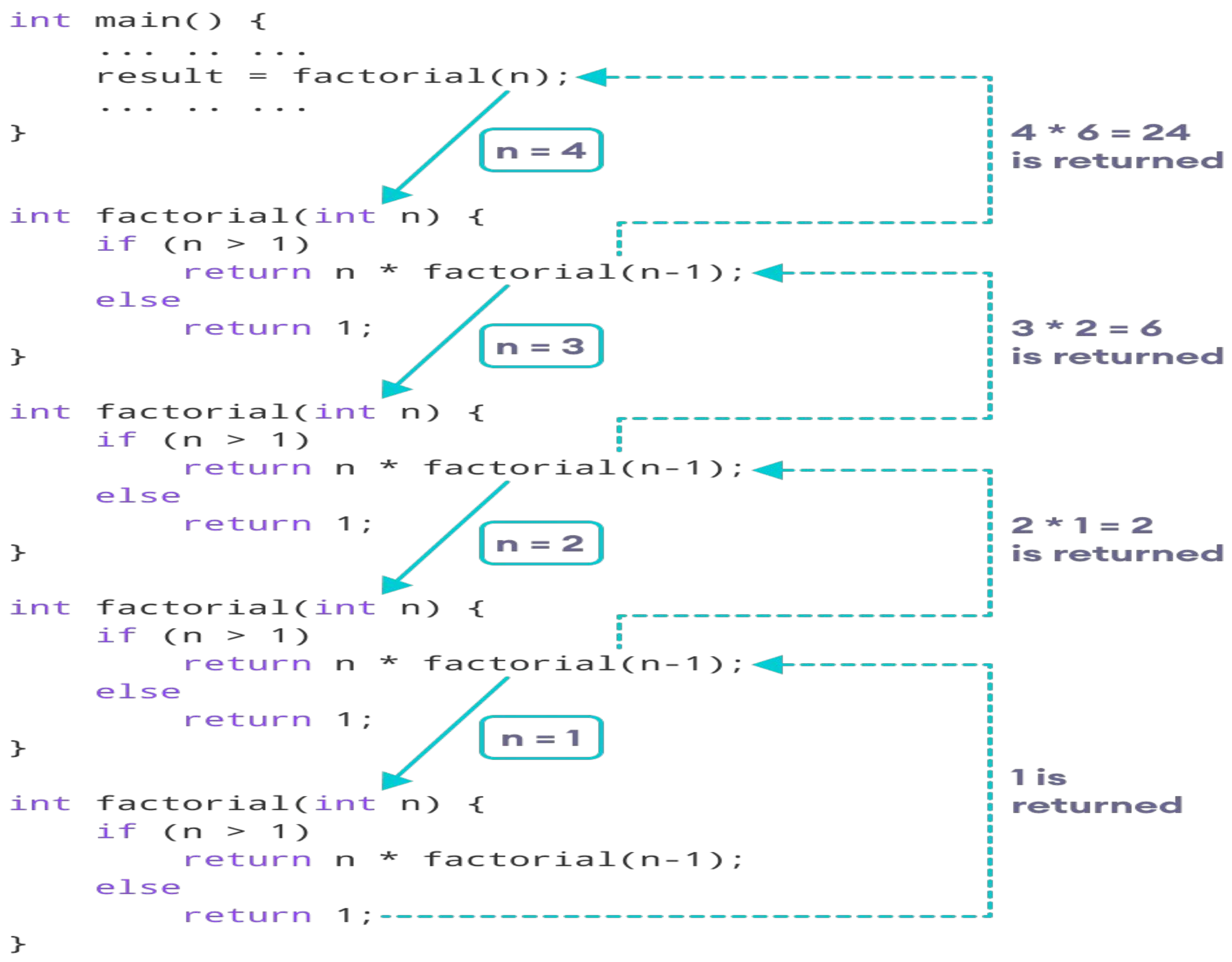

return 3 x 2
return 4 x factorial(3)
return 5 x factorial(4)
return 6 x factorial(5)
Call stack


return 4 x 6
return 5 x factorial(4)
return 6 x factorial(5)
Call stack


return 5 x 24
return 6 x factorial(5)
Call stack


return 6 x 120
Call stack

The factorial function !



```
int main() {
    ... .. ...
    result = factorial(n);
    ... .. ...
}

int factorial(int n) {
    if (n > 1)
        return n * factorial(n-1);
    else
        return 1;
}

int factorial(int n) {
    if (n > 1)
        return n * factorial(n-1);
    else
        return 1;
}

int factorial(int n) {
    if (n > 1)
        return n * factorial(n-1);
    else
        return 1;
}

int factorial(int n) {
    if (n > 1)
        return n * factorial(n-1);
    else
        return 1;
}
```

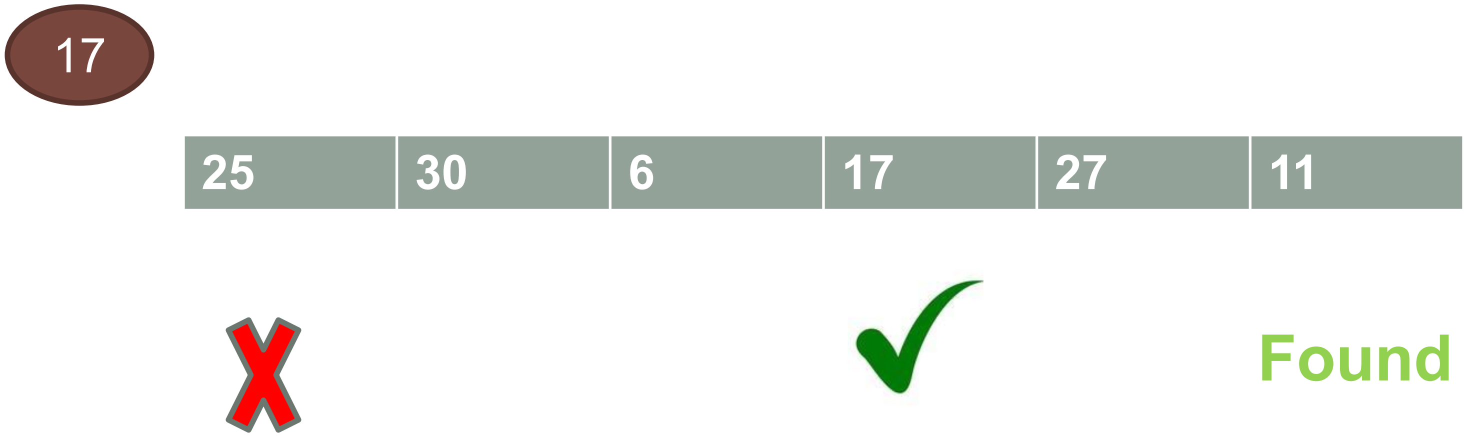
Search Algorithms

- **Linear** (*Sequential*) Searching Algorithm
  - Can deal with unsorted arrays

- **Binary** Searching Algorithm
  - Must deal with sorted arrays

Linear Search Algorithm

- Compare target value with all elements of the array until you find a match or reach the last element in the array
- EX: arr = [25,30,6,17,27,11], n = 6, target = 17



Linear Search Algorithm

```
int linSearch(int list [ ], int listLength, int
key)
{
        int loc;

        for(loc = 0; loc < listLength; loc++)
        {
        if(list[loc] == key)
              return loc;
       }
       return -1;
}
```

Linear Search Algorithm

- The best case time complexity of linear search takes place when the element to be searched for is on the first index □ Time complexity is **1**
- The worst case will take place if:
  - The element to be search is in the last index
  - The element to be search is not present in the list
  - □ Time Complexity is **N**

Recursive Linear Search

$$f(a, n, target) = \begin{cases} -1 & \text{if } n<=0 \\ n-1 & \text{if } target=a[n-1] \\ f(a, n-1, target) & \text{Otherwise} \end{cases}$$

Recursive Linear Search

```
int linearSearch(int a[], int n, int target) {
   // Recursive version of linear search
 if (n <= 0) // an empty list is specified
       return -1;
   else {
      if (a[n-1] == target) //test final
 position
         return n-1;
      else // search the rest of the list
 recursively
         return linearSearch(a, n-1, target);
   }
}
```

Binary Search Algorithm

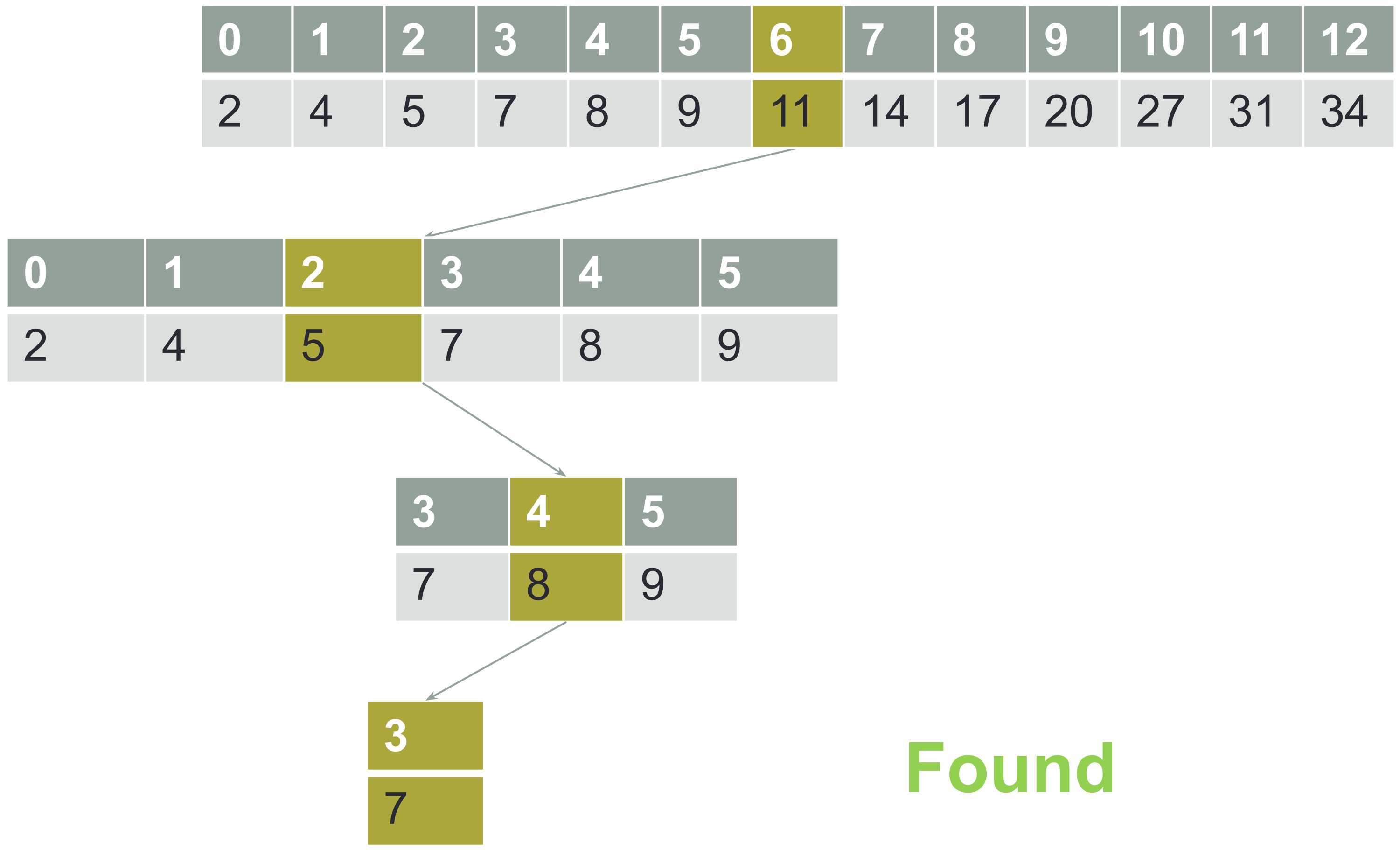
- Can only be performed on <u>**a sorted list**</u> !!!
- Uses *divide and conquer* technique to search list

Binary Search Algorithm

- Search item is compared with middle element of list
- If search item **<** middle element of list, search is restricted to **first half** of the list
- If search item **>** middle element of list, search **second half** of the list
- If search item = middle element, search is complete

Binary Search Algorithm

- Search for (7)



0	1	2	3	4	5	6	7	8	9	10	11	12
2	4	5	7	8	9	11	14	17	20	27	31	34

0	1	2	3	4	5
2	4	5	7	8	9

3	4	5
7	8	9

3
7

Found

Binary Search Algorithm

```
int binarySearch(int [ ] list, int listLength, int key)
{
    int first = 0, last = listLength - 1;
    int mid;
    boolean found = false;

    while (first <= last && !found) {
        mid = (first + last) / 2;
        if (list[mid] == key)
            found = true;
        else
            if(list[mid] > key)
                last = mid - 1;
            else
                first = mid + 1;
    }
    if (found)
        return mid;
    else
        return -1;
} //end binarySearch
```

Binary Search Algorithm

- **The Best Case Complexity -** occurs when the element to search is found in first comparison Time complexity is **1**
- **The Worst Case Complexity -** occurs, when we have to keep reducing the search space till it has only one element Time Complexity is Log**N**

Recursive Binary Search

$$f(a,\ f,\ l,\ target) \begin{cases} -1 & \text{if } f > l \\ (f+l)/2 & \text{if } target = (f+l)/2 \\ f(a,\ f,\ ((f+l)/2) - 1,\ target) & \text{if } target < (f+l)/2 \\ f(a,\ ((f+l)/2) + 1, l,\ target) & \text{if } target > (f+l)/2 \end{cases}$$

Recursive Binary Search

```
int binarySearch(int a[], int first, int last, int target)
{
   // Preconditions: a is an array sorted in ascending order, first is the index of
  //the first element to search, last is the index of the last element to //search,
  target is the item to search for.

   if (first > last)
       return -1; // -1 indicates failure of search

   int mid = (first+last)/2;

  if (a[mid] == target)
       return mid;

  else if (target < a[mid]) // left/lower sub-array
       return binarySearch(a, first, mid-1, target);

  else // target must be > a[mid] // right/upper sub-array
       return binarySearch(a, mid+1, last, target);
}
```

Task

- Implement function that check if an array has two sequential numbers that their sum is K using concept of recursion and binary search where the array is sorted

$$F(arr,st,end,K)\begin{cases} \text{false} & \text{if (mid = st and sum < K) OR} \\ & \text{if(mid = end and sum > K)} \\ \text{True} & \text{if sum = K} \\ F(arr,st,mid) & \text{if sum > K} \\ F(arr,mid,end) & \text{if sum < K} \end{cases}$$

- Ex: arr = [6,8,9,10,15] , K = 17 -> True
- Ex: arr = [5,12,20,25,30] , K = 38 -> False

Bonus

- The same of previous task but use linked list instead of array

Thank You