CS206

Recursion, Binary Search

The Factorial

- Recursive definition: $f(n) = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot f(n-1) & else \end{cases}$

- Java method

```
public static int factorial(int n) {
  if (n<=0)
    return 1;
  else
    return n*factorial(n-1)
}
```

Recursive Method

- Base case(s):
  - no recursive calls are performed
  - every chain of recursive calls must reach a base case eventually
- Recursive calls:
  - Calls to the same method in a way that progress is made towards a base case

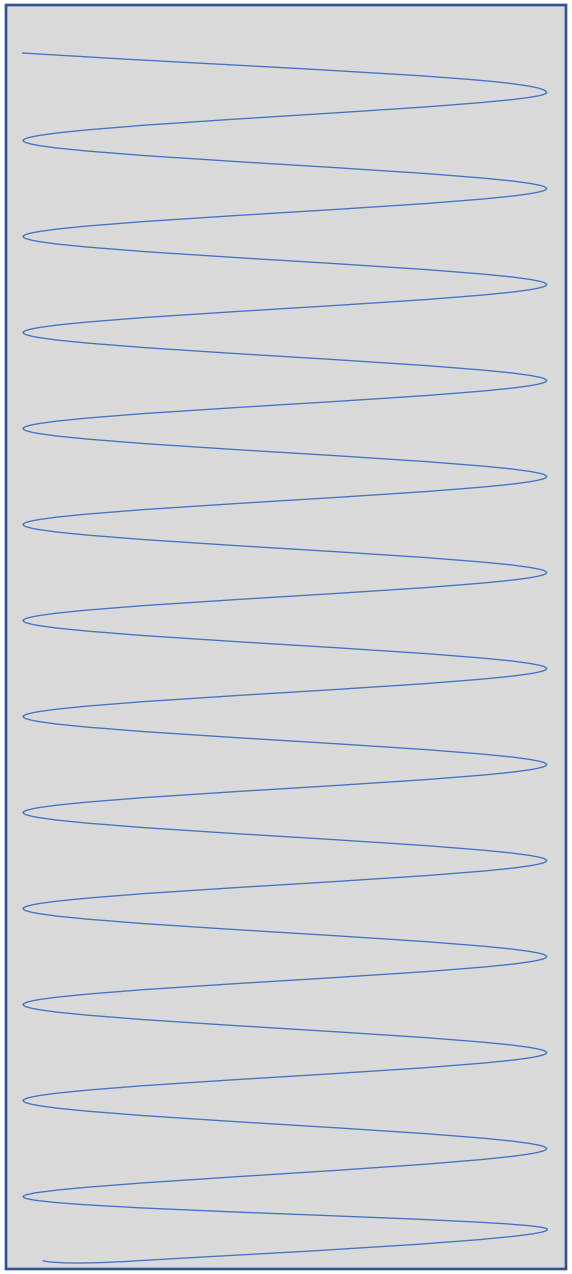
Compiled Code

```
1.   void main() {
2.     int A = 10;
3.     int B = factorial(5);
4.     System.out.println(B);
5.   }
```

```
1.   int factorial(int n) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int F = n *
     factorial(n-1);
6.       return F;
7.     }
8.   }
```

Executing Function

Call Stack



Compiled Code

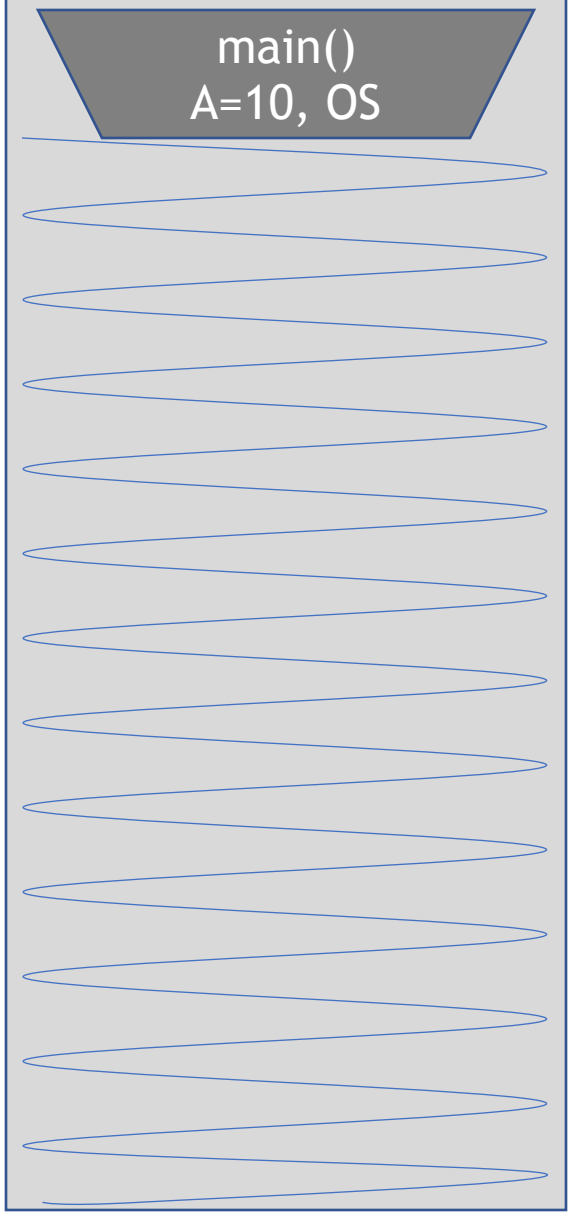
```
1.   void main() {
2.     int A = 10;
3.     int B = factorial(5);
4.     System.out.println(B);
5.   }
```

```
1.   int factorial(int n) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int F = n *
     factorial(n-1);
6.       return F;
7.     }
8.   }
```

Executing Function

```
2.     int A = 10;
3.     int B = factorial(5);
4.     System.out.println(B);
5.   }
```

Call Stack



Compiled Code

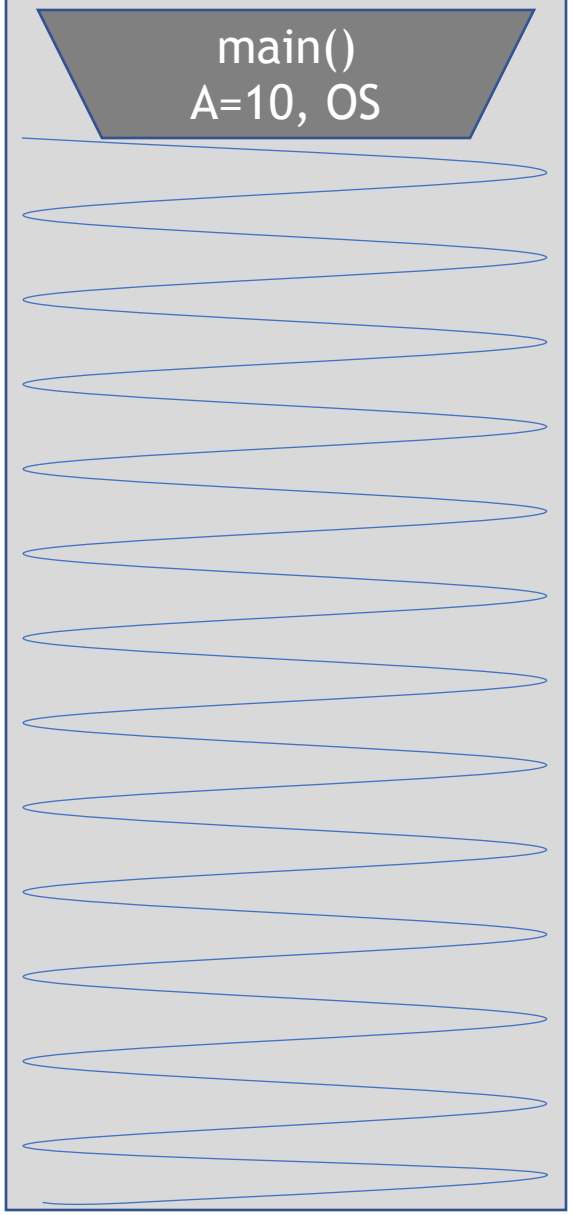
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

Call Stack



Compiled Code

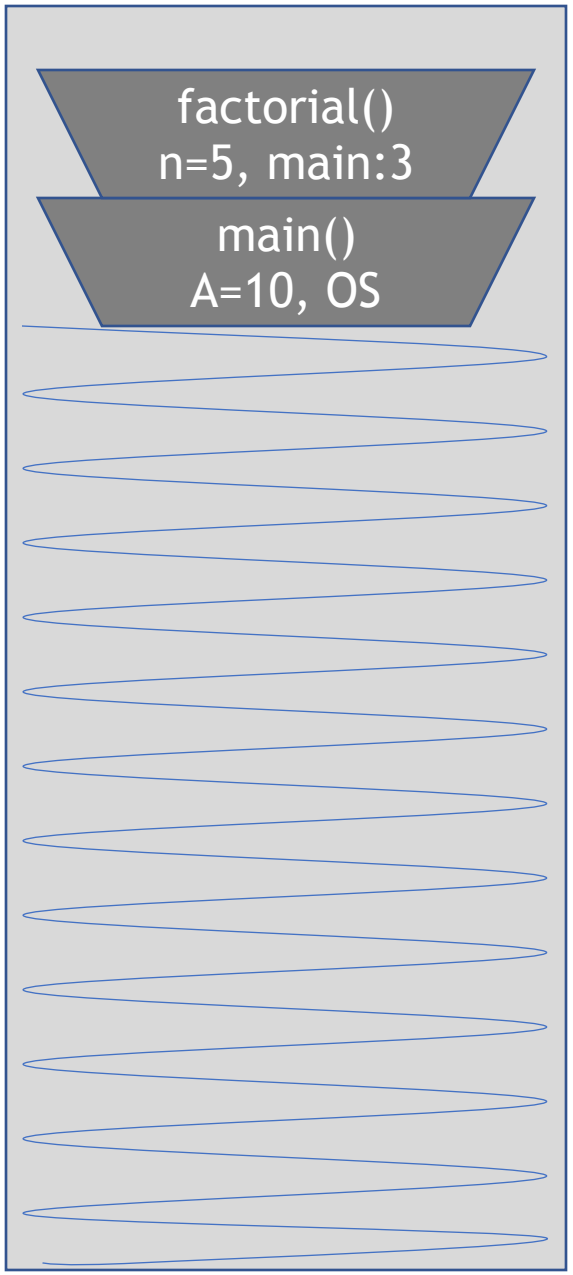
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

Call Stack



Compiled Code

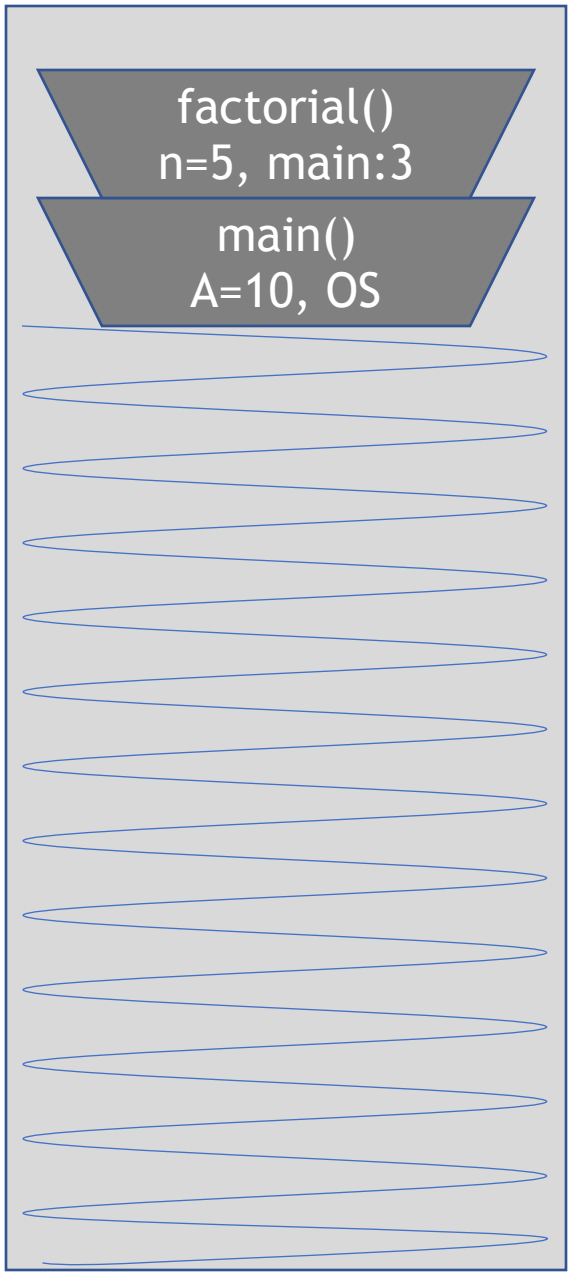
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=5) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

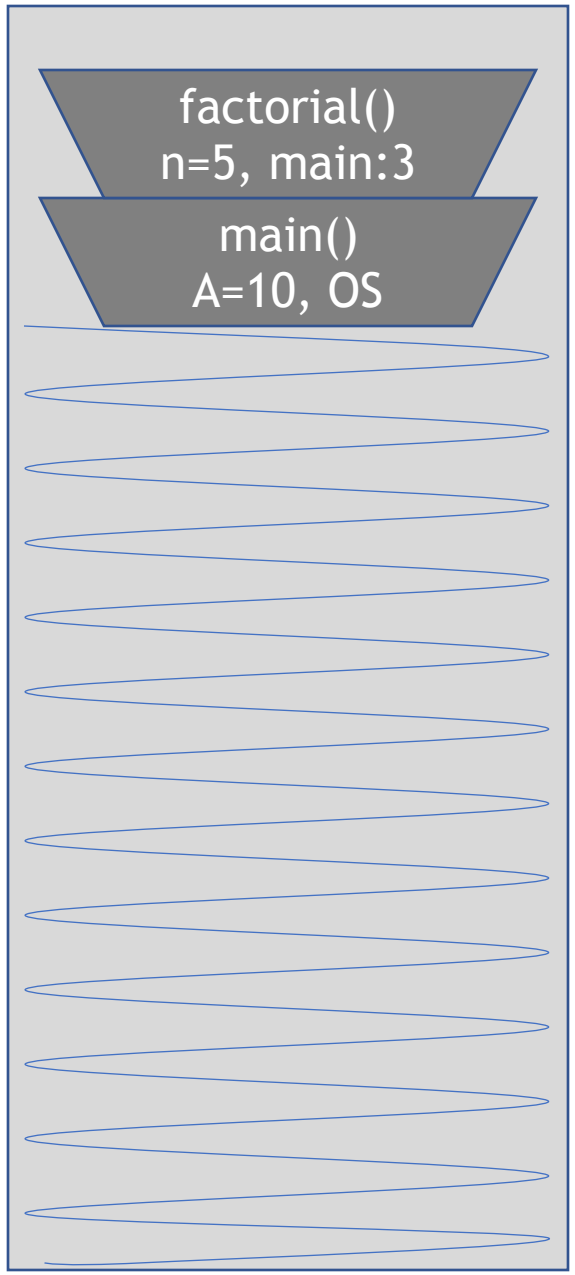
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=5) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

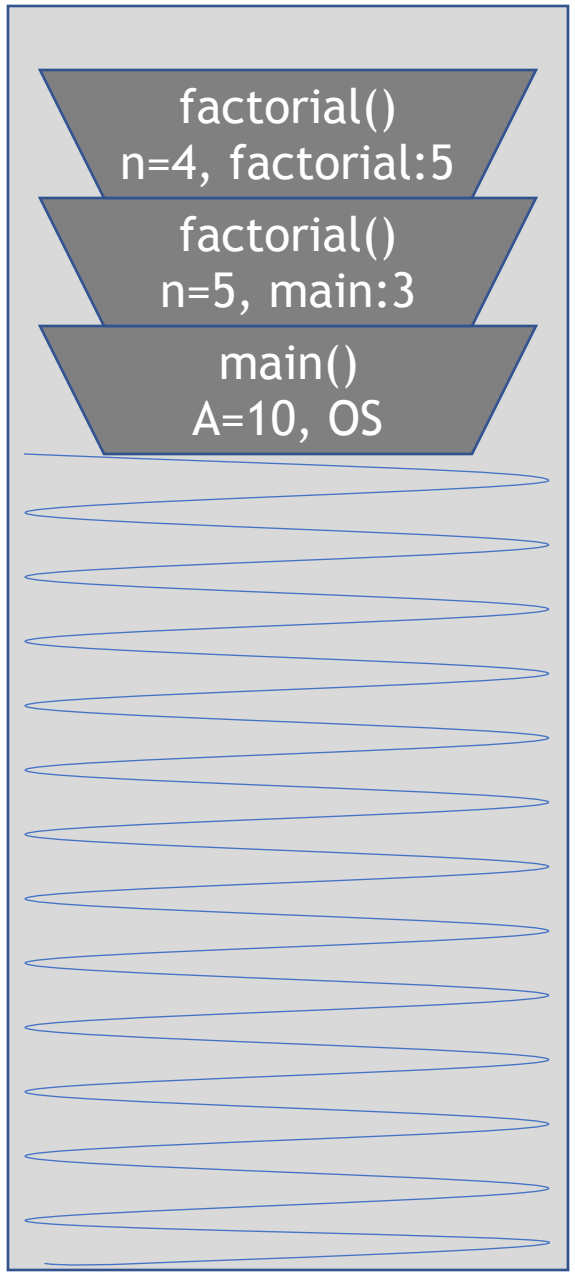
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=5) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

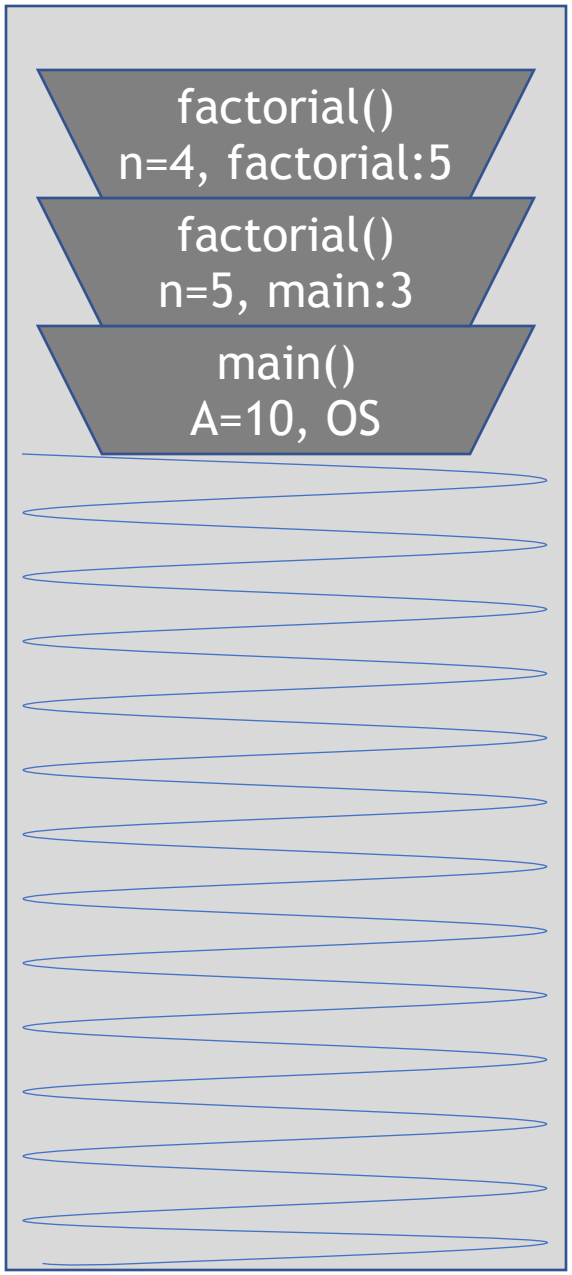
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
2.  int factorial(int n=4) {
3.    if (n == 1) {
4.      return 1;
5.    } else {
      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

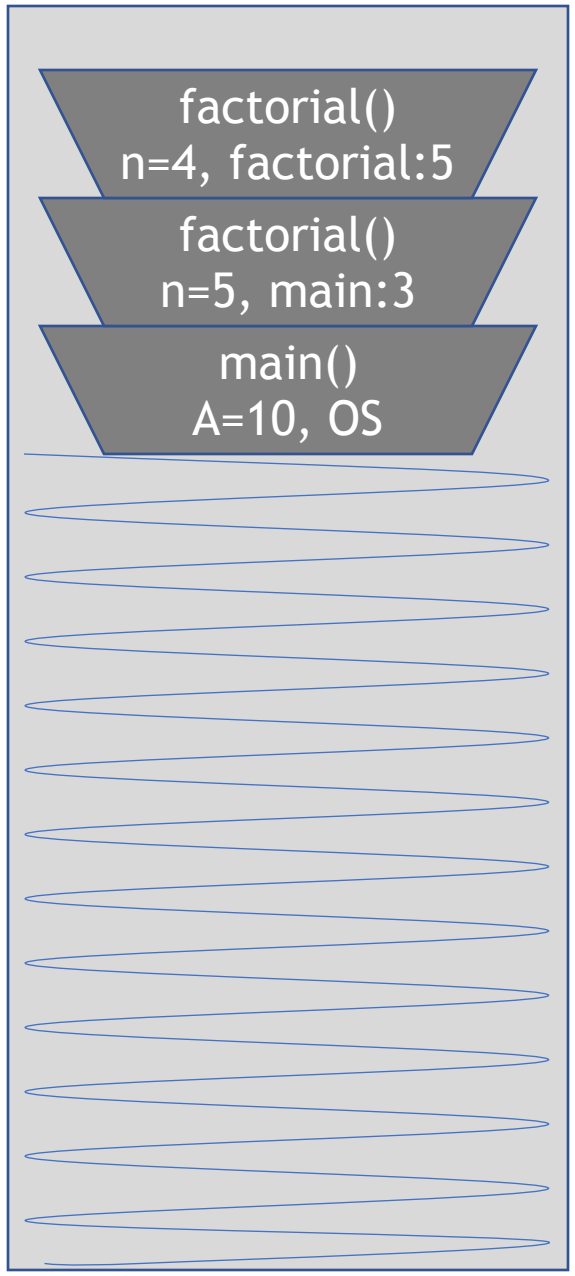
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=4) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

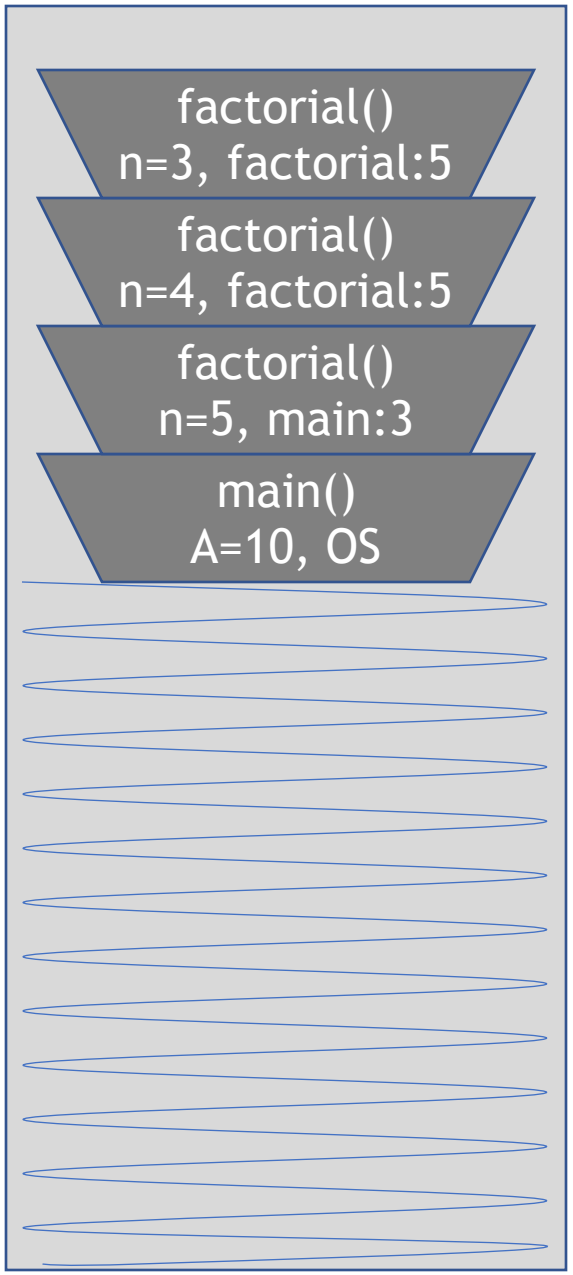
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=4) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

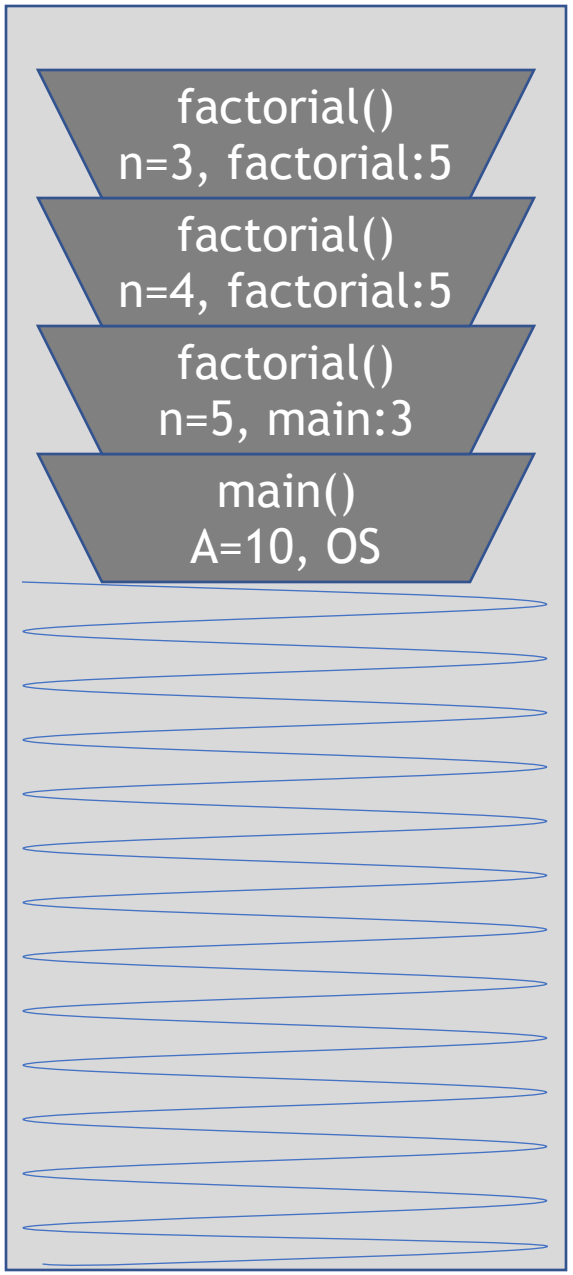
```
1.   void main() {
2.     int A = 10;
3.     int B = factorial(5);
4.     System.out.println(B);
5.   }
```

```
1.   int factorial(int n) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int F = n *
     factorial(n-1);
6.       return F;
7.     }
8.   }
```

Executing Function

```
1.   int factorial(int n=3) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int F = n *
     factorial(n-1);
6.       return F;
7.     }
8.   }
```

Call Stack



Compiled Code

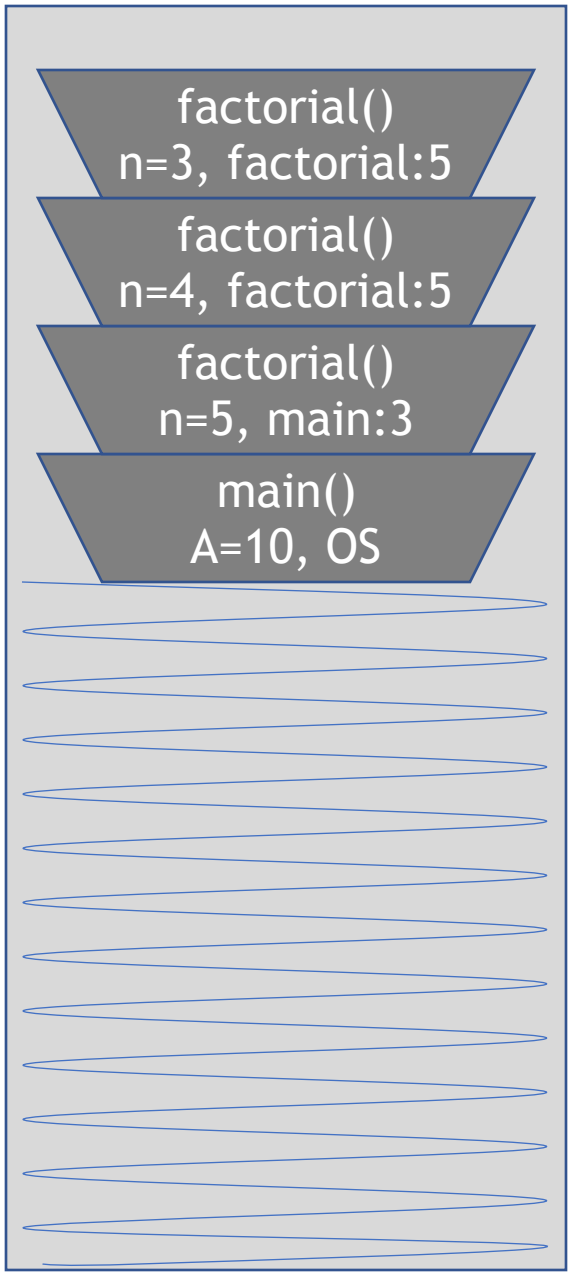
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=3) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

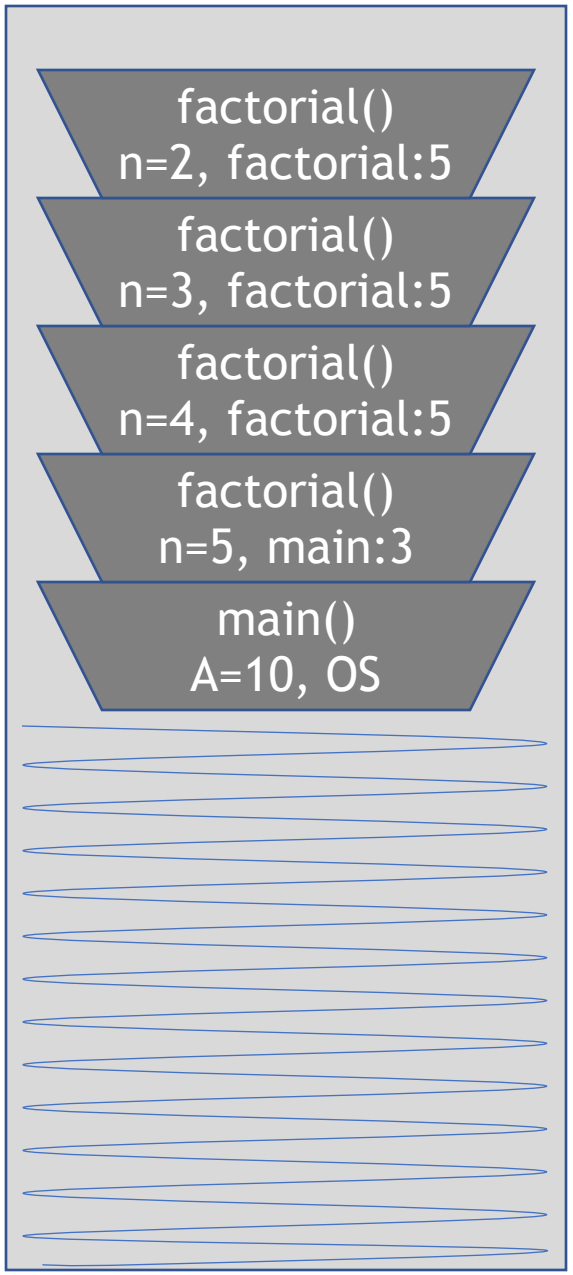
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=3) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

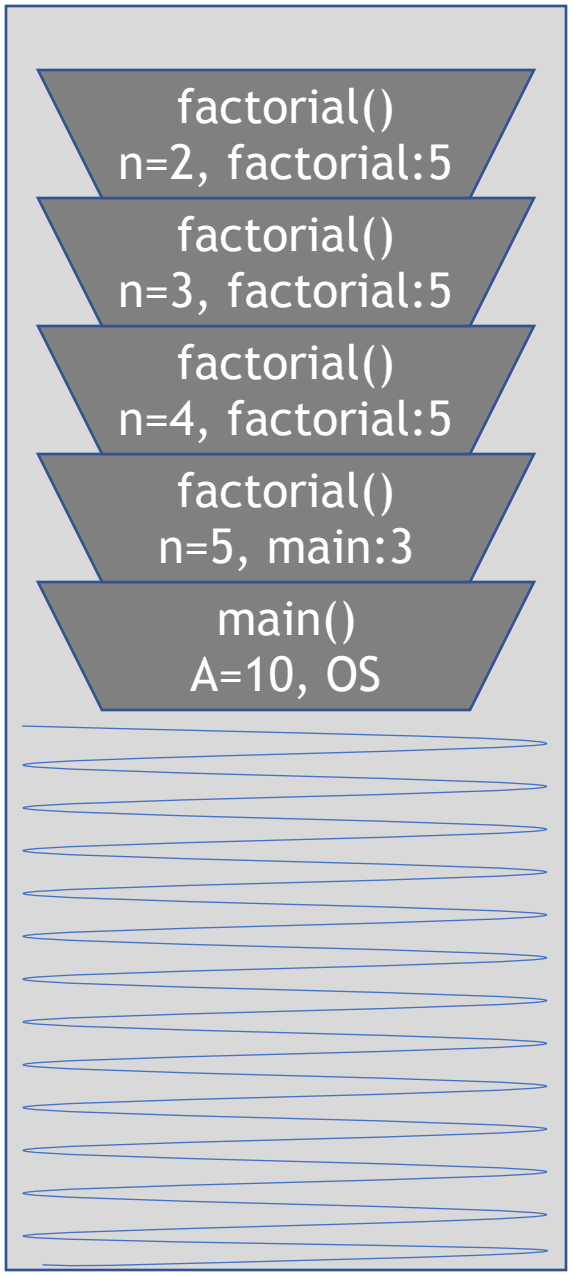
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=2) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

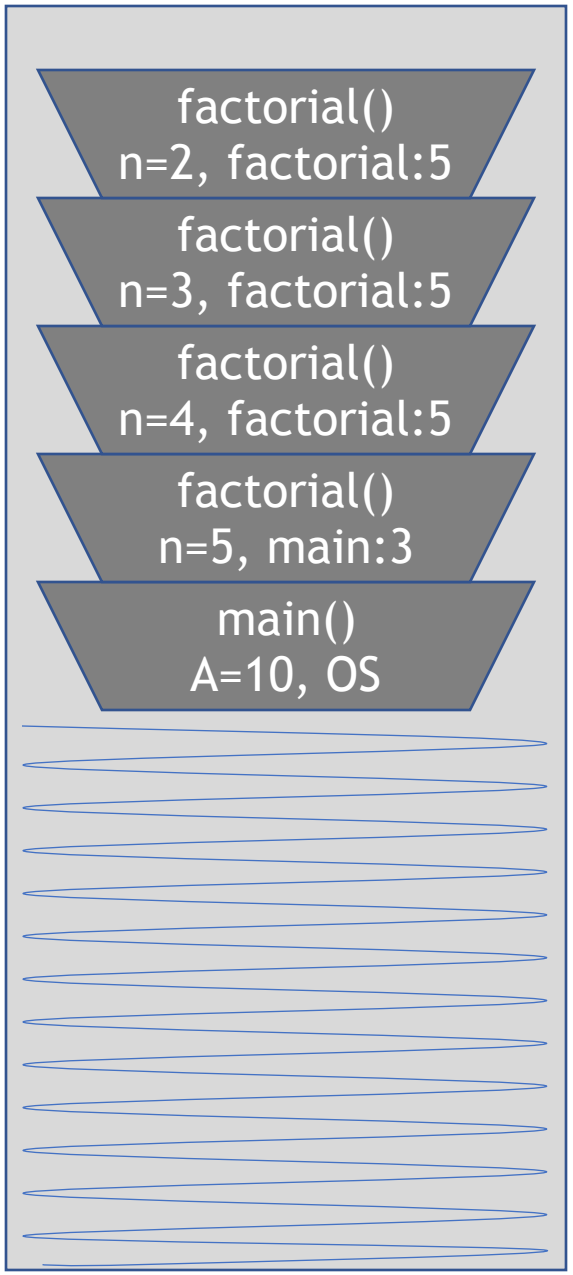
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=2) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Call Stack



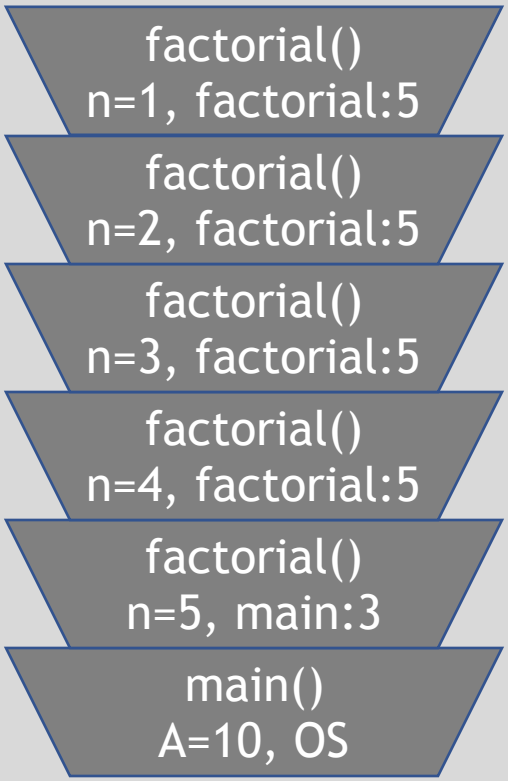
Compiled Code

```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=2) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```



Compiled Code

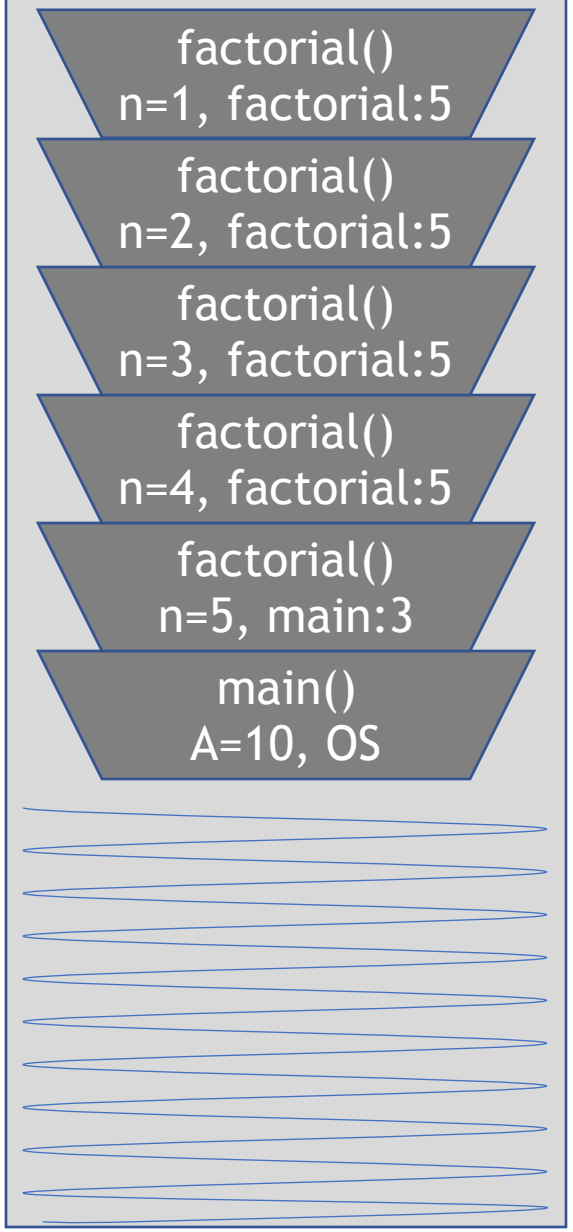
```
1.   void main() {
2.     int A = 10;
3.     int B = factorial(5);
4.     System.out.println(B);
5.   }
```

```
1.   int factorial(int n) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int F = n *
     factorial(n-1);
6.       return F;
7.     }
8.   }
```

Executing Function

```
1.   int factorial(int n=1) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int F = n *
     factorial(n-1);
6.       return F;
7.     }
8.   }
```

Call Stack



Compiled Code

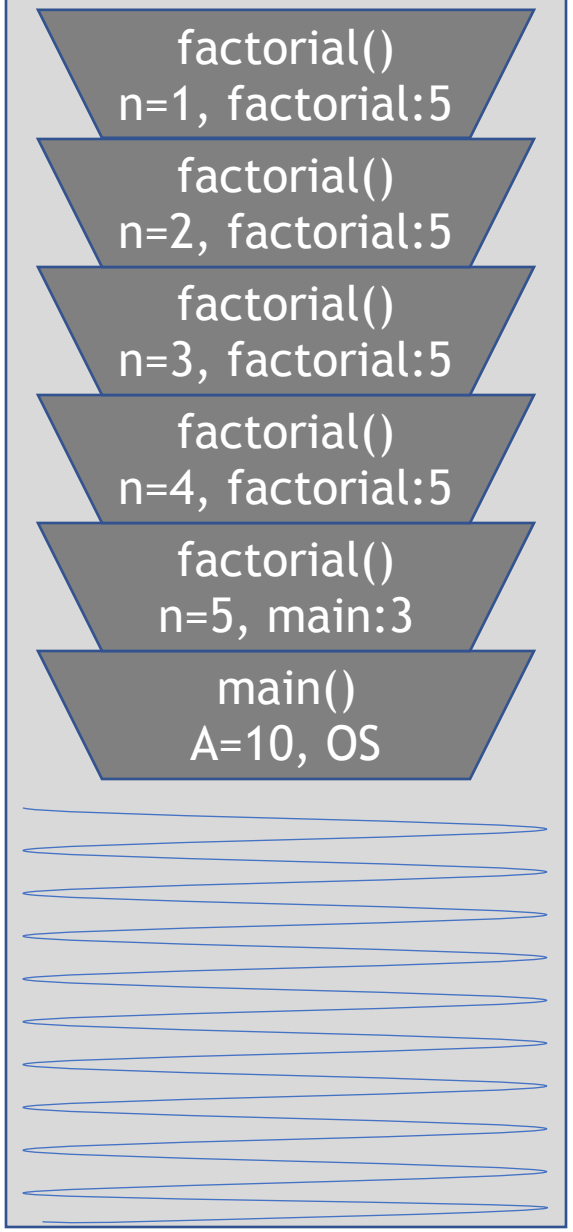
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=1) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

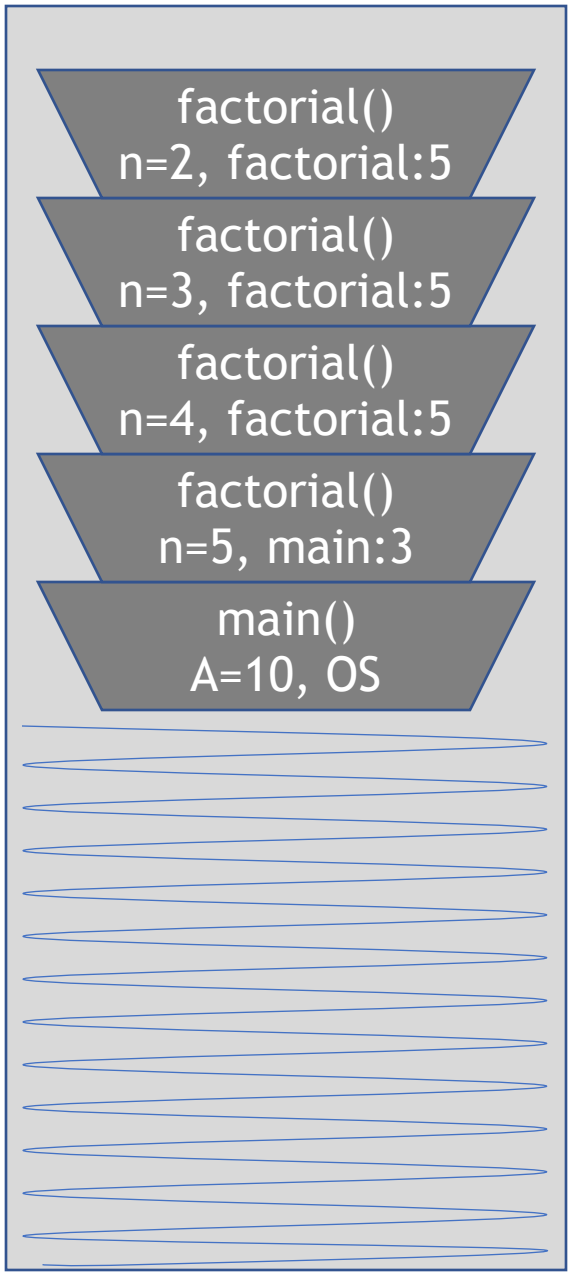
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=2) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n * 1;
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

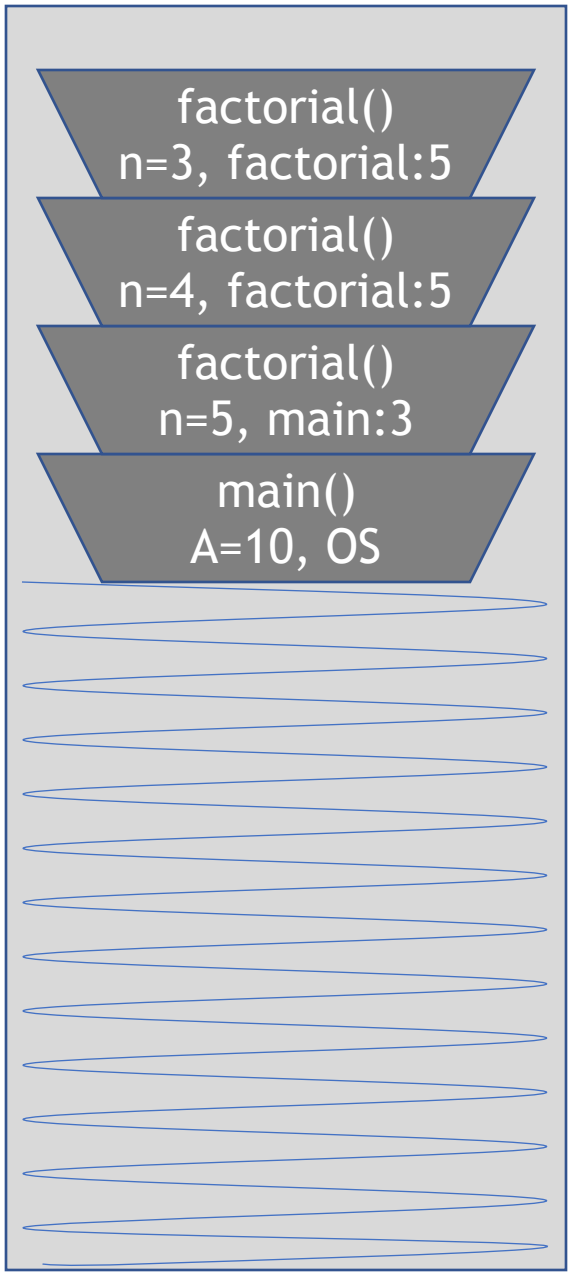
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=3) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n * 2;
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

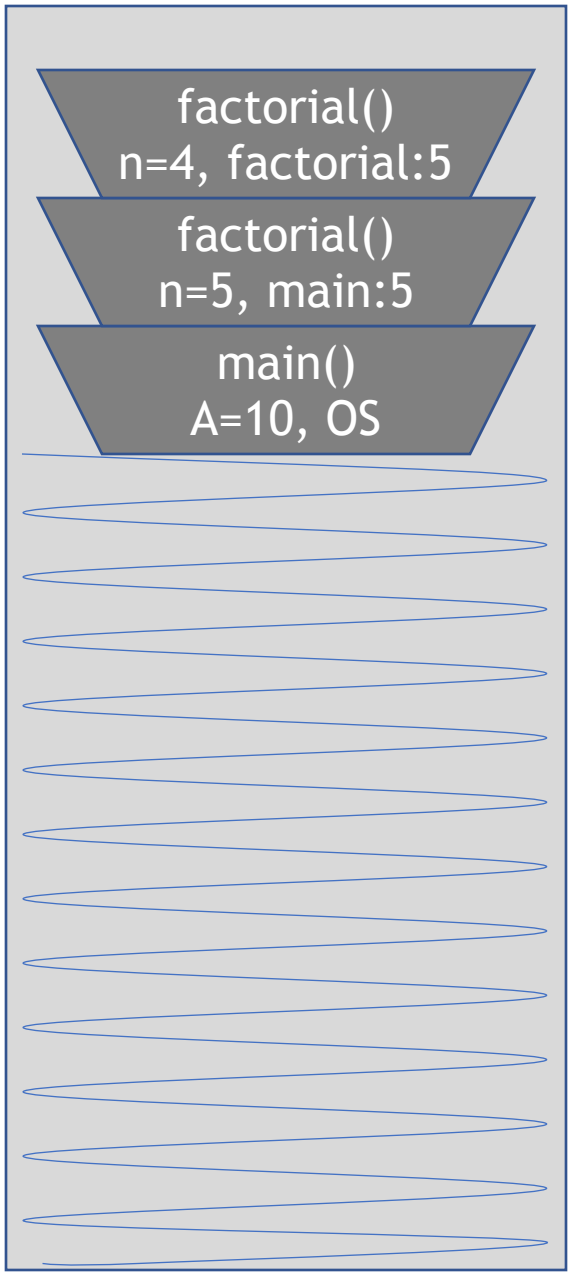
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=4) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n * 6;
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

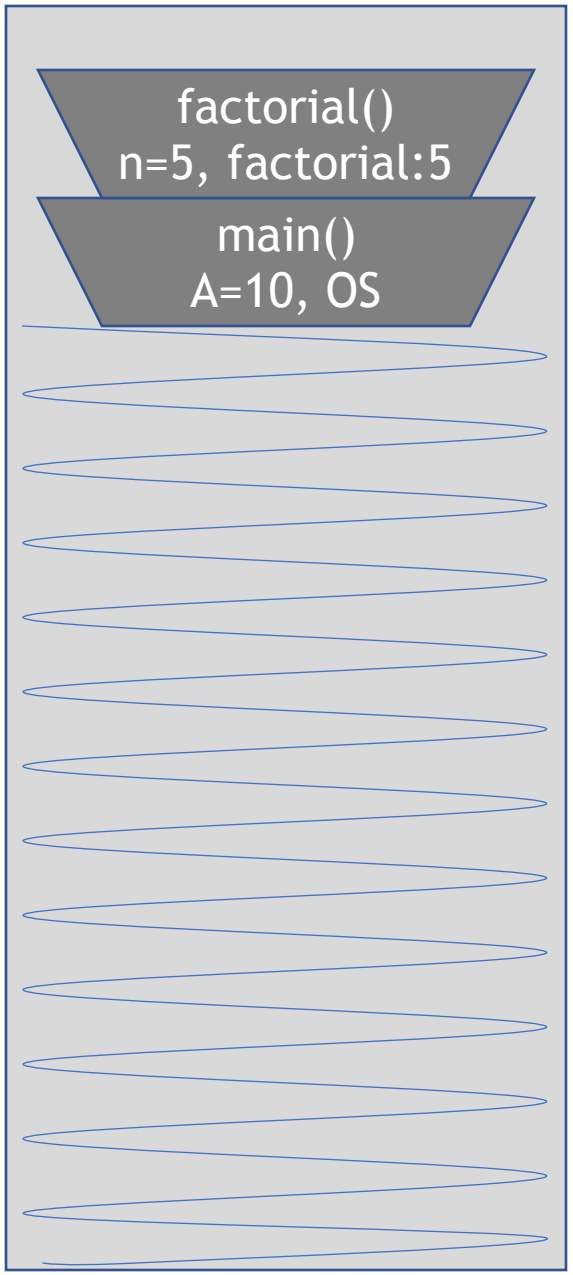
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=5) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n * 24;
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

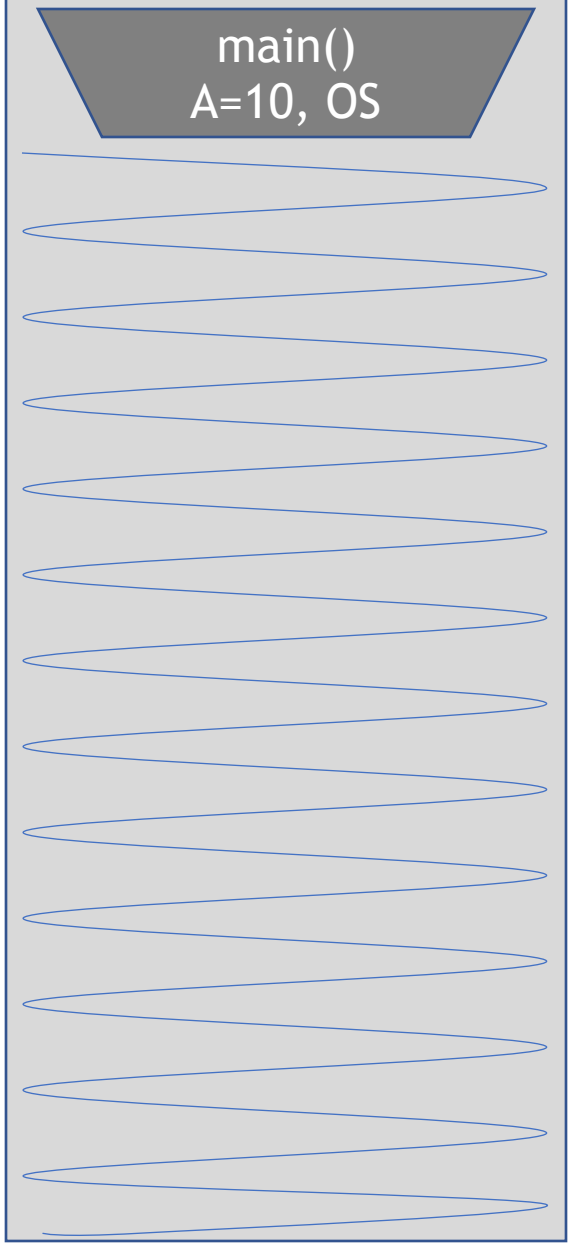
```
1.   void main() {
2.     int A = 10;
3.     int B = factorial(5);
4.     System.out.println(B);
5.   }
```

```
1.   int factorial(int n) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int F = n *
     factorial(n-1);
6.       return F;
7.     }
8.   }
```

Executing Function

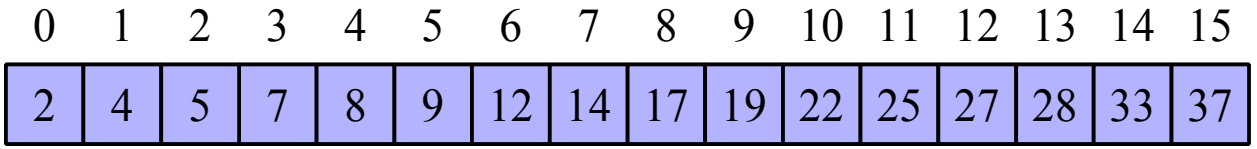
```
1.   void main() {
2.     int A = 10;
3.     int B = 120;
4.     System.out.println(B);
5.   }
```

Call Stack



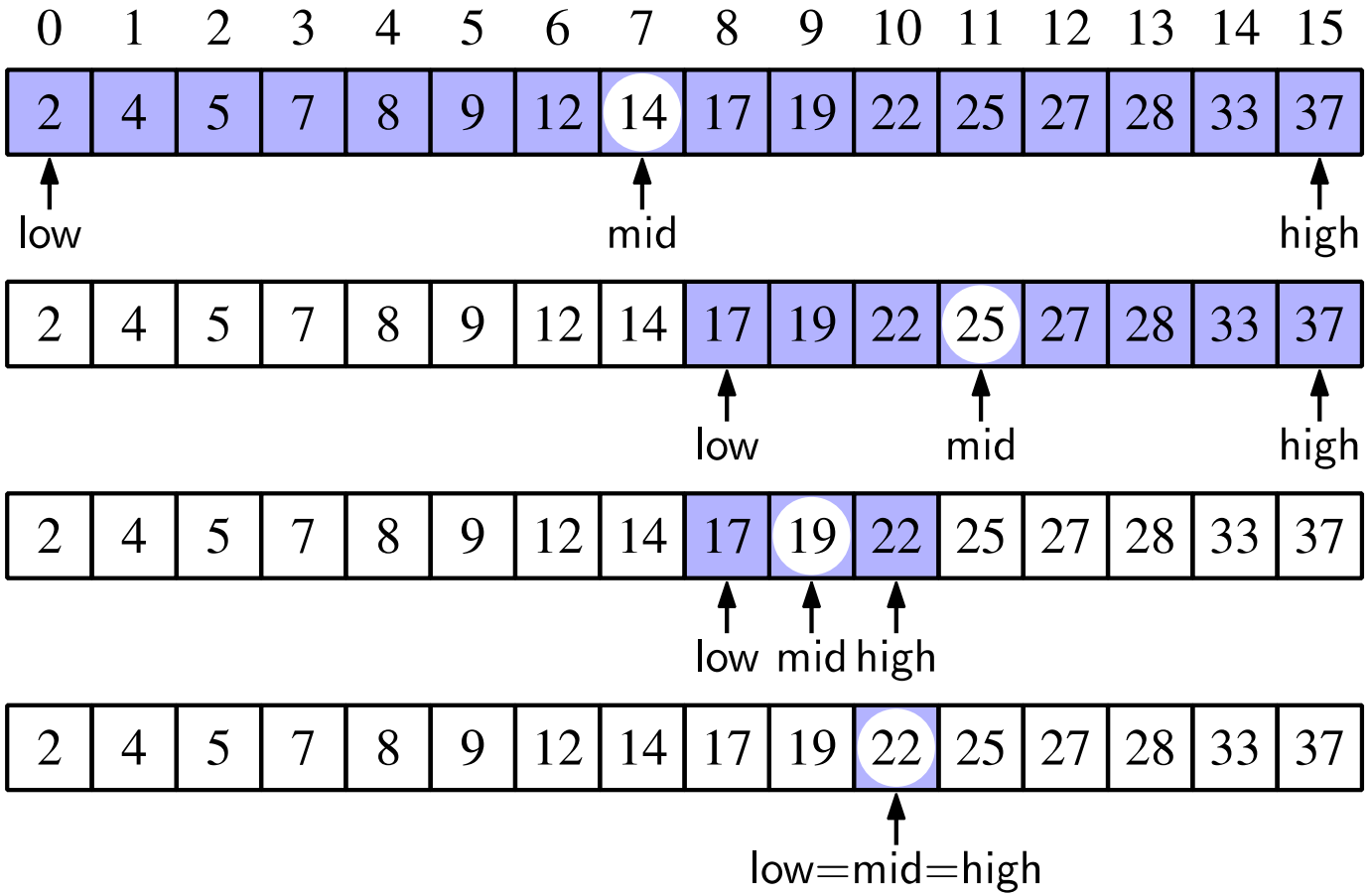
Binary Search

- Search for an integer (22) in an ordered list



- $mid = \left\lfloor \frac{low + high}{2} \right\rfloor = \left\lfloor \frac{0 + 15}{2} \right\rfloor = 7$
  - `target==data[mid]`, found
  - `target>data[mid]`, recur on second half
  - `target<data[mid]`, recur on first half

target = 22


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
2 4 5 7 8 9 12 14 17 19 22 25 27 28 33 37
low
mid
high
2 4 5 7 8 9 12 14 17 19 22 25 27 28 33 37
low
mid
high
2 4 5 7 8 9 12 14 17 19 22 25 27 28 33 37
low mid high
2 4 5 7 8 9 12 14 17 19 22 25 27 28 33 37
low=mid=high

Code

```
   /**
    * Implement Binary search, recursively on internal array of ints
    * @param target the item to be found
    * @param lo the bottom of the range being searched
    * @param hi the top of the range being searched
    * @param steps the number of steps the search has taken
    * @return true if the target was found
    */
   private boolean iSearch(int target, int lo, int hi, int steps) {
   if (lo>hi) return false;
   int mid = (lo+hi)/2;
   System.out.println(target + " " + data[mid] + " " + lo + " " + hi
+ " " + steps);
   if (data[mid]==target) return true;
   if (data[mid]<target)
       return iSearch(target, mid+1, hi, steps+1);
   else
       return iSearch(target, lo, mid-1, steps+1);
   }
```

BinrySearch in 206HW6

Binary Search Analysis

- Each recursive call divides the array in half
- If the array is of size n, it divides (and searches) at most lgn times before the current half is of size 1
- $O(lgn)$

Backtracking with Recursion

- Previous examples all progressed linearly to success/failure
  - Tail recursion
  - Easy to rewrite using loops
- So consider doing binary like search on an unsorted array
  - Need to backtrack and try other directions on failure.
  - Can be very difficult to rewrite using loops

Backtracker

```
    private boolean iSearch(int target, int lo, int hi,
int depth)
    {
   steps++;
   System.out.print(steps);
   if (lo>hi) { System.out.println(); return false; }
   int mid = (lo+hi)/2;
   System.out.println(" " + target + " " + data[mid] + "
+ lo + " " + hi + " " + depth);
   if (data[mid]==target) return true;
   if (iSearch(target, mid+1, hi, depth+1))
      return true;
   return iSearch(target, lo, mid-1, depth+1);
    }
```

backtrackingsearch in 206hw6

Towers Of Hanoi

```
public void solve(int n, String start, String auxiliary, String end) {
    if (n == 1) {
        System.out.println(start + " -> " + end);
    } else {
        solve(n - 1, start, end, auxiliary);
        System.out.println(start + " -> " + end);
        solve(n - 1, auxiliary, start, end);
    }
}
```