Recursion
(Divide and Conquer strategies)

Divide and conquer

- Strategy to solve problems
  - Break problem into smaller problems
  - Solve smaller problems
  - Combine results
- Strategy can be applied “recursively” to smaller problems
  - Continue dividing problem until solution is trivial

BEFORE START

- So, what is n!?

- Factorial of n *(denoted n!)* is a product of integer numbers from 1 to n.

    - ▫ For instance, 6! = 1 * 2 * 3 * 4 * 5 * 6 = 720.

BEFORE START

- Recursion is one of techniques to calculate factorial.

BEFORE START

- Indeed, 6! = 5! * 6.

- To calculate factorial of n, we should calculate it for $(n-1)$.

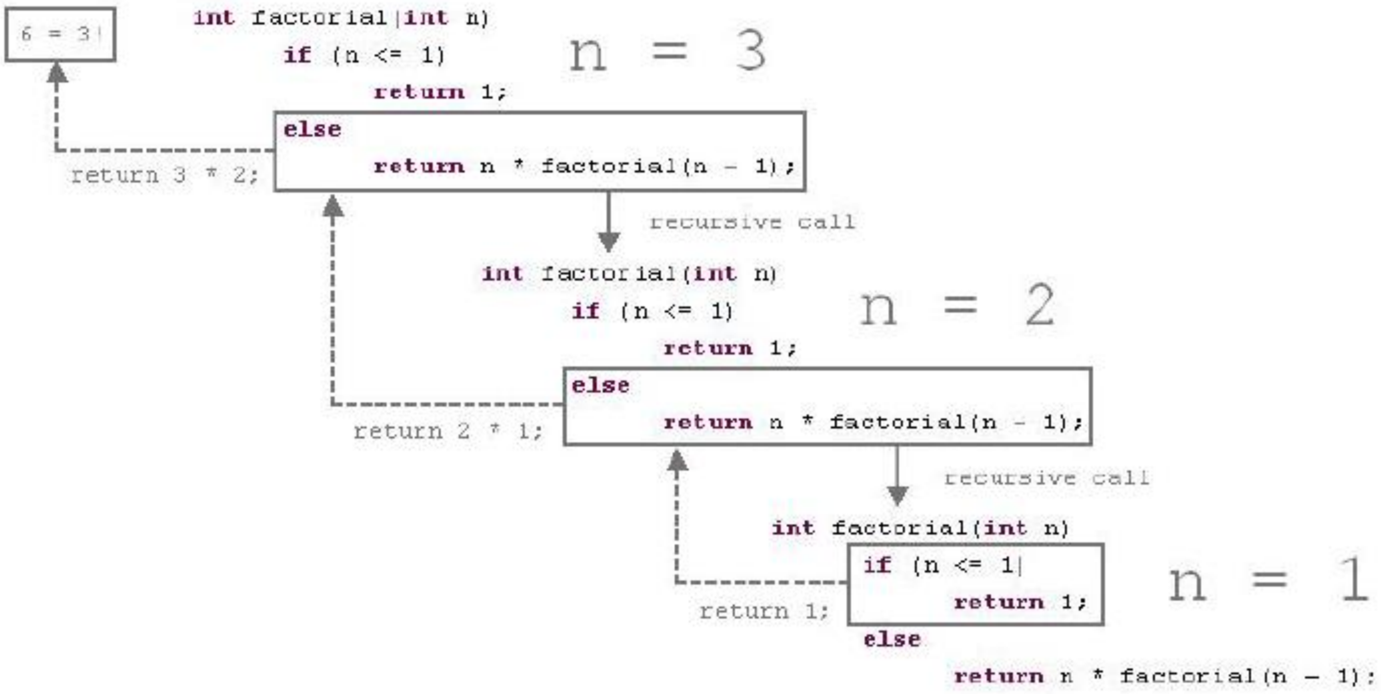
- To calculate factorial of $(n-1)$ algorithm should find $(n-2)!$ and so on.

EXAMPLE: FACTORIAL

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

EXAMPLE: FACTORIAL

Calculation of 3! in details



```
int factorial(int n)
    if (n <= 1)
        return 1;
else
        return n * factorial(n - 1);
```

```
int factorial(int n)
    if (n <= 1)
        return 1;
else
        return n * factorial(n - 1);
```

```
int factorial(int n)
    if (n <= 1)
        return 1;
else
        return n * factorial(n - 1);
```

```
main()
{
     int sum, max=10;
     sum=fact(max);
     printf(“1*2*…*10=%3d”,sum);

 }


int fact(int N){
  if(N==1)
    return 1;
  else
   return * fact(N-1);
}
```

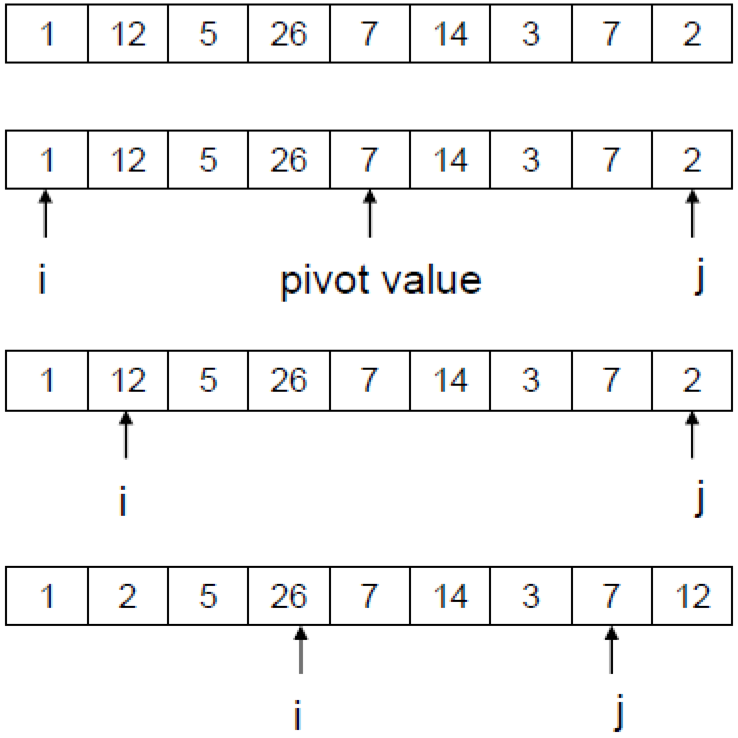
Quick Sort Revisit

QUICKSORT

- *Example:*
  - Sort {1, 12, 5, 26, 7, 14, 3, 7, 2} using quicksort.

1	12	5	26	7	14	3	7	2

QUICKSORT



1	12	5	26	7	14	3	7	2

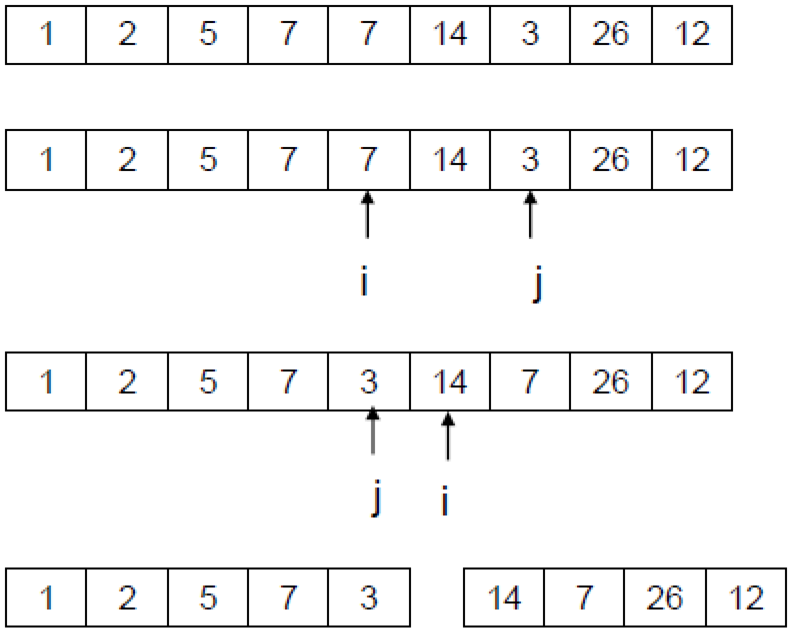
1	12	5	26	7	14	3	7	2
i				pivot value				j

1	12	5	26	7	14	3	7	2
	i							j

1	2	5	26	7	14	3	7	12
			i				j	

- Unsorted
- Pivot value = 7
- 12 >= 7 >= 2. swap
- 26 >= 7 >= 7, swap

QUICKSORT



1	2	5	7	7	14	3	26	12

1	2	5	7	7	14	3	26	12

- 7 >= 7 >= 3, swap

1	2	5	7	3	14	7	26	12

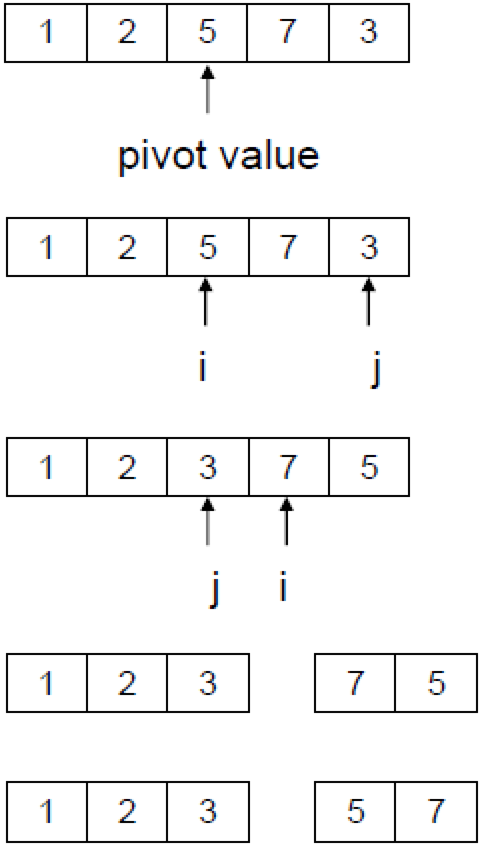
- i > j, stop partition

1	2	5	7	3

14	7	26	12

- run quicksort recursively

QUICKSORT



1	2	5	7	3

1	2	5	7	3

1	2	3	7	5

1	2	3

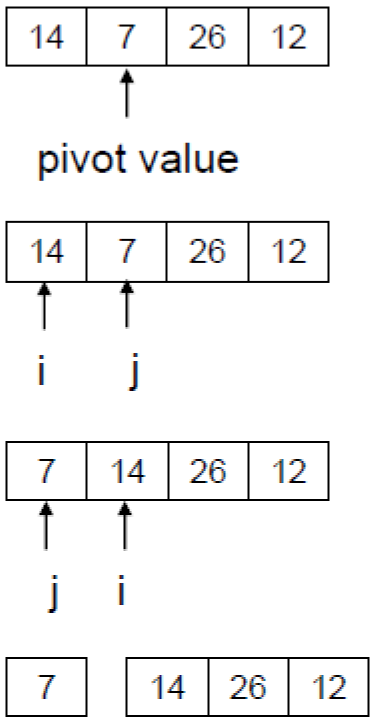
7	5

1	2	3

5	7

- Pivot value = 5
- 5 >= 5 >= 3. swap
- i > j, stop partition
- run quicksort recursively

QUICKSORT



14 7 26 12

pivot value

14 7 26 12

i j

7 14 26 12

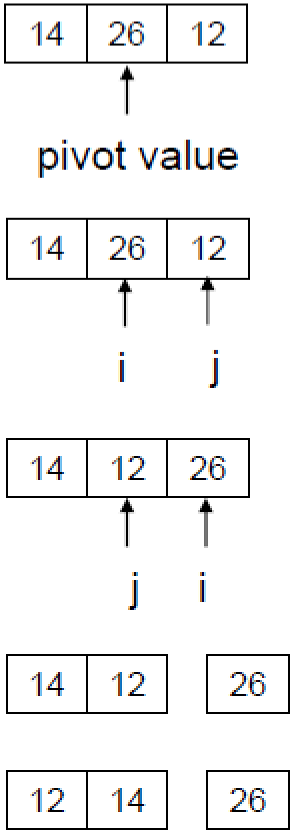
j i

7

14 26 12

- Pivot value = 7
- 14 >= 7 >= 7. swap
- i > j, stop partition
- run quicksort recursively

QUICKSORT



14	26	12

pivot value

14	26	12
	i	j

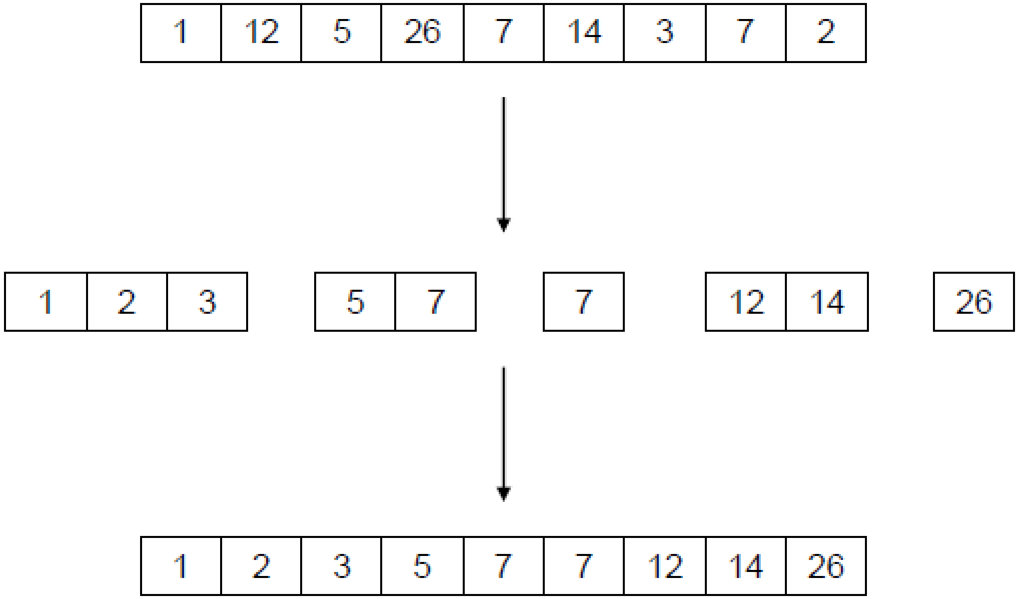
14	12	26
	j	i

14	12		26

12	14		26

- Pivot value = 26
- 26 >= 26 >= 12. swap
- i > j, stop partition
- run quicksort recursively

QUICKSORT



1 12 5 26 7 14 3 7 2

1 2 3 5 7 7 12 14 26

1 2 3 5 7 7 12 14 26

```
void quickSort(int arr[], int left, int right) {
    int i = left, j = right;
    int tmp;
    int pivot = arr[(left + right) / 2];

    /* partition */
    while (i <= j) {
        while (arr[i] < pivot)
            i++;
        while (arr[j] > pivot)
            j--;

        if (i <= j) {
            tmp = arr[i];
            arr[i] = arr[j];
            arr[j] = tmp;
            i++;
            j--;
        }
    };

    /* recursion */
    if (left < j)
        quickSort(arr, left, j);
    if (i < right)
        quickSort(arr, i, right);
}
```