Computer Programming: Skills & Concepts (CP)
Recursion (including mergesort)

Ajitha Rajan

Tuesday 14 November 2017
Today’s lecture

- Recursion: functions that call themselves.
- Examples: factorial and fibonacci.
- The long integer type.
- MergeSort (recursive version).
- Allocating memory dynamically with calloc.
Computing Factorial

Task: write a function that computes factorial

\[ n! = \begin{cases} 
  n \times (n - 1)! & \text{if } n > 1 \\
  1 & \text{if } n = 1 
\end{cases} \]
Factorial with for loop

```c
long factorial(int n) {
    long fact = 1;
    int i;
    for(i=1; i<=n; i++) {
        fact = fact * i;
    }
    return fact;
}
```

We use `long` (meaning ‘long int’) rather than `int`, because as `n` increases, `factorial(n)` can get very large - for example `13!` is too large for an `int` variable (on DICE).

On DICE, `long` uses 64 bits - can store values up to $\pm 9.22337204 \times 10^{18}$. **System dependent:** On old machines, `long` might be 32 bits only.
Factorial with recursion

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

The function `factorial` calls itself!
Factorial with recursion

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

The function factorial calls itself!
A function is a ‘black box’ to which you give arguments, and it returns a result. Recursive calls are no different from any other call!
Factorial with recursion

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

The function `factorial` calls itself!
A function is a ‘black box’ to which you give arguments, and it returns a result. Recursive calls are no different from any other call!
`factorial(n)` needs to know what \((n-1)!\) is, so it just calls the black box `factorial(n-1)`. 
long factorial(int n) {
    if (n<=1) {
        return 1;
    }
    return n * factorial(n-1);
}

The function factorial calls itself!
A function is a ‘black box’ to which you give arguments, and it returns a result. Recursive calls are no different from any other call!
factorial(n) needs to know what \((n - 1)!\) is, so it just calls the black box factorial(n-1).
factorial(1) doesn't need to call anything.
Execution of recursion

If you look at how the sequence of execution goes, it’s like this:

```java
factorial(5)
    return 5 * factorial(4);
    return 4 * factorial(3);
        return 3 * factorial(2);
            return 2 * factorial(1);
                return 1;
            return 2 * 1;
        return 3 * 2
    return 4 * 6
return 5 * 24
```

120

but just think in terms of the black box.
Fibonacci numbers

The Fibonacci numbers are the sequence 0, 1, 1, 2, 3, 5, 8, 13, 21, ...

\[ F(n) = \begin{cases} 
F(n - 1) + F(n - 2) & \text{if } n > 1 \\
1 & \text{if } n = 1 \\
0 & \text{if } n = 0
\end{cases} \]

\[ \frac{F(n+1)}{F(n)} \] converges to the golden ratio \( \frac{1 + \sqrt{5}}{2} = 1.618034. \)
Recursive computation of Fibonacci numbers

```c
long fibonacci(int n) {
    if (n==0)
        return 0;
    if (n==1)
        return 1;
    return fibonacci(n-1) + fibonacci(n-2);
}
```

- How many function calls does it roughly take to compute `fibonacci(10)` or `fibonacci(100)`?

- **Running time?**
  - Is this faster/slower than the `for` and `while` versions from lecture 6?
  - Interesting comparison using `clock()` :-).
    more interesting than comparing linear against binary search.
**Merge**

**Idea:**
Suppose we have two arrays $a$, $b$ of length $n$; and $m$ respectively, and that these arrays are already sorted. Then the merge of $a$ and $b$ is the sorted array of length $n+m$ we get by walking through both arrays jointly, taking the smallest item at each step.

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

$\uparrow i=0$ $\uparrow j=0$ $\uparrow k=0$
Merge

Idea:

Suppose we have two arrays $a$, $b$ of length $n$; and $m$ respectively, and that these arrays are already sorted. Then the merge of $a$ and $b$ is the sorted array of length $n+m$ we get by walking through both arrays jointly, taking the smallest item at each step.

\[
\begin{array}{c|c|c}
\hline
a & b & c \\
\hline
2 & 7 & 18 \\
\hline
2 & 7 & 18 \\
\hline
\end{array}
\]

$\uparrow i=0$

\[
\begin{array}{c|c|c}
\hline
2 & 7 & 18 \\
\hline
2 & 7 & 18 \\
\hline
\end{array}
\]

$\uparrow i=0$

\[
\begin{array}{c|c|c}
\hline
4 & 5 & 6 \\
\hline
4 & 5 & 6 \\
\hline
\end{array}
\]

$\uparrow j=0$

\[
\begin{array}{c|c|c|c|c|c}
\hline
2 & 7 & 18 \\
\hline
2 & 7 & 18 \\
\hline
2 & 7 & 18 \\
\hline
4 & 5 & 6 \\
\hline
4 & 5 & 6 \\
\hline
4 & 5 & 6 \\
\hline
\end{array}
\]

$\uparrow k=0$

$\uparrow k=0$

$\uparrow k=0$
Merge

Idea:
Suppose we have two arrays \( a, b \) of length \( n \) and \( m \) respectively, and that these arrays are already sorted. Then the merge of \( a \) and \( b \) is the sorted array of length \( n+m \) we get by walking through both arrays jointly, taking the smallest item at each step.

### Example

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>7</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>18</td>
<td></td>
</tr>
</tbody>
</table>

\( i=0 \)

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

\( j=0 \)

\( k=0 \)
Idea:
Suppose we have two arrays $a$, $b$ of length $n$; and $m$ respectively, and that these arrays are already sorted. Then the merge of $a$ and $b$ is the sorted array of length $n+m$ we get by walking through both arrays jointly, taking the smallest item at each step.

```
<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>7</td>
<td>18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

$\uparrow i=0$

$\uparrow j=0$

$\uparrow k=0$

$\uparrow i=0$

$\uparrow j=0$

$\uparrow k=0$

$\uparrow i=1$

$\uparrow j=0$

$\uparrow k=1$

$\uparrow i=1$

$\uparrow j=0$

$\uparrow k=1$
<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>7</td>
<td>18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$i=1$


<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$j=1$


<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$k=2$


<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$k=3$


<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$k=4$
\[
\begin{align*}
\text{Row } i = 1: & \quad \begin{array}{c}
2 \quad 7 \quad 18 \\
\uparrow i = 1
\end{array} & \quad \begin{array}{c}
4 \quad 5 \quad 6 \\
\uparrow j = 1
\end{array} & \quad \begin{array}{c}
2 \quad 4 \quad 5 \quad 6 \\
\uparrow k = 2
\end{array} \\
\text{Row } i = 1: & \quad \begin{array}{c}
2 \quad 7 \quad 18 \\
\uparrow i = 1
\end{array} & \quad \begin{array}{c}
4 \quad 5 \quad 6 \\
\uparrow j = 2
\end{array} & \quad \begin{array}{c}
2 \quad 4 \quad 5 \quad 6 \\
\uparrow k = 3
\end{array} \\
\text{Row } i = 1: & \quad \begin{array}{c}
2 \quad 7 \quad 18 \\
\uparrow i = 1
\end{array} & \quad \begin{array}{c}
4 \quad 5 \quad 6 \\
\uparrow j = 3
\end{array} & \quad \begin{array}{c}
2 \quad 4 \quad 5 \quad 6 \\
\uparrow k = 4
\end{array}
\end{align*}
\]
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 2 | 7 | 18 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 2 | 7 | 18 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 2 | 7 | 18 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 2 | 7 | 18 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 2 | 7 | 18 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |

$\uparrow i = 1$

$\uparrow j = 1$

$\uparrow k = 2$

$\uparrow j = 2$

$\uparrow k = 3$

$\uparrow j = 3$

$\uparrow k = 4$

$\uparrow j = 3$

$\uparrow k = 4$

$\uparrow j = 3$

$\uparrow k = 4$

$\uparrow j = 3$

$\uparrow k = 4$

$\uparrow j = 3$

$\uparrow k = 4$

$\uparrow j = 3$

$\uparrow k = 4$

$\uparrow j = 3$

$\uparrow k = 4$

$\uparrow j = 3$

$\uparrow k = 4$

$\uparrow j = 3$

$\uparrow k = 4$

$\uparrow j = 3$

$\uparrow k = 4$

$\uparrow j = 3$

$\uparrow k = 4$

$\uparrow j = 3$

$\uparrow k = 4$

$\uparrow j = 3$

$\uparrow k = 4$

$\uparrow j = 3$

$\uparrow k = 4$

$\uparrow j = 3$

$\uparrow k = 4$

$\uparrow j = 3$

$\uparrow k = 4$
<table>
<thead>
<tr>
<th>i=1</th>
<th>j=1</th>
<th>k=2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 7 18</td>
<td>4 5 6</td>
<td>2 4</td>
</tr>
<tr>
<td>i=1</td>
<td>j=2</td>
<td>k=3</td>
</tr>
<tr>
<td>2 7 18</td>
<td>4 5 6</td>
<td>2 4 5</td>
</tr>
<tr>
<td>i=1</td>
<td>j=3</td>
<td>k=4</td>
</tr>
<tr>
<td>2 7 18</td>
<td>4 5 6</td>
<td>2 4 5 6</td>
</tr>
<tr>
<td>i=2</td>
<td>j=3</td>
<td>k=5</td>
</tr>
<tr>
<td>2 7 18</td>
<td>4 5 6</td>
<td>2 4 5 6 7</td>
</tr>
</tbody>
</table>

CP Lect 18 – slide 10 – Tuesday 14 November 2017
void merge(int a[], int b[], int c[], int m, int n) {
    int i=0, j=0, k=0;
    while (i < m || j < n) {
        /* In the following condition, if j >= n, C knows the condition is true, and it *doesn’t* check the rest, so it doesn’t matter that b[j] is off the end of the array. So the order of these is important */
        if (j >= n || (i < m && a[i] <= b[j])) {
            /* either run out of b, or the smallest elt is in a */
            c[k++] = a[i++];
        } else {
            /* either run out of a, or the smallest elt is in b */
            c[k++] = b[j++];
        }
    }
}
sorting using merge

- merge can create an *overall sort* from two smaller arrays which were individually sorted.

- Could we use merge as a helper function to perform the task of sorting a given array (*where the initial arrays is not at all sorted*)?
  - **Divide-and-Conquer** is the process of solving a big problem, by utilizing the solutions to smaller versions of that problem.
  - For sorting, we could *divide* our (unsorted) input array in two pieces, then *sort* those two smaller subarrays individually (*recursion*) and finally get the overall sort using merge.
sorting using merge

mergesort (int key [], int n)

key

\( j = \frac{n}{2} \) ("divide")

\( \phi \)

\( \frac{n}{2} - 1 \)

\( \frac{n}{2} \)

\( n-1 \)

mergesort (key, j)

sorted!

merge (the two sorted arrays)

mergesort (key+j, n-j)

sorted!
MergeSort through recursion

Typical implementation of MergeSort is recursive:

- The sort of the array is the result of sorting each half of the array and then merging those two sorted subarrays.
- Merge function takes two sorted arrays and creates the ‘merge’ of those arrays.

C issues:
We will need to create a ‘scratch array’ to pass to merge (as the c parameter) to write the result of ‘merging’ the two smaller (already sorted) subarrays.

- ‘scratch’ array needs same length as input array.
- Need to dynamically allocate space.
- In C, use calloc to get space of a ‘dynamic’ (not fixed) amount.
  ```c
  void *calloc(size_t num, size_t size)
  ```
- Returns a ‘pointer to void’ … just means a pointer/address of no particular type.
  The pointer will be NULL if there was not enough available space.
recursive mergesort

int mergesort(int key[], int n){
    int j, *w;
    if (n <= 1) { return 1; } /* base case, sorted */
    w = calloc(n, sizeof(int)); /* space for temporary array */
    if (w == NULL) { return 0; } /* calloc failed */
    j = n/2;
    /* do the subcalls and check they succeed */
    if ( mergesort(key, j) 
        && mergesort(key+j, n-j) ) {
        merge(key, key+j, w, j, n-j);
        for (j = 0; j < n; ++j)
            key[j] = w[j];
        free(w); /* Free up the dynamic memory no longer in use. */
        return 1;
    } else { /* a subcall failed */
        free(w);
        return 0;
    }
}

CP Lect 18 – slide 15 – Tuesday 14 November 2017
Wrap-up and Reading

- Running-time of mergesort is proportional to $n \log(n)$. Compares favourably with BubbleSort ($n^2$).
- Read more about \textit{recursion} in Sections 5.14, 5.15 of Kelley & Pohl.
- Read more about \textit{calloc} in Section 6.8 of Kelley & Pohl.
- Our implementation of \texttt{mergesort} is on the course webpage. \texttt{mergerec.c} also has a \texttt{wrt} function for printing out small arrays, and a \texttt{main} for testing/timing on arrays of various sizes.
- Kelley & Pohl have a ‘bottom-up’ version of MergeSort for \textit{array lengths} a power-of-2.
  - More troublesome/fiddly than the recursive version
  - Can adapt their ‘bottom-up’ version of MergeSort to work for general array lengths. \textit{If you want a challenge try this} (but test it rigorously)