Recursion

- Programming technique in which a method calls itself
- Duplicate itself but with different parameters until task is trivial
- Equivalent to mathematical induction: Recursive step + Base step
- Well-suited for the “divide and conquer” strategy
- Each level of recursion keeps building in the memory stack
- Conceptually more elegant than iterations

Various Examples

1- Summation/Triangular number: \( \text{sum}(n) = \text{sum}(n-1) + n \) with \( \text{sum}(1) = 1 \)
2- Factorials: \( \text{fact}(n) = \text{fact}(n-1) \times n \) with \( \text{fact}(1) = 1 \)
3- Anagrams: find all the \( n! \) permutations for \( n \) letters
4- Binary search: keeps dividing in half (still only one recursion)
5- Fibonacci (2 recursions- divide and conquer in action)
6- Display Linked list (combines linked-list+recursion)
Hanoi Towers

- Ancient puzzle consisting of a number of disks placed on 3 columns

- **Goal:** Move all disks from A to C

- **Rule**
  - Move one disk at a time
  - No bigger disk is on top of smaller disk at any time
Hanoi Towers

To do: Test Java applet Towers.html

- Trick to do it manually:
  - If the subtree you are trying to move has an odd number of disk, start by moving the topmost disk to the target tower for the subtree
  - If it is even, use the buffer (intermediate) tower
Hanoi Towers

- Algorithmic solution: moving ”subtree” of disks … recursively
Hanoi Towers - Recursive algorithm

```java
public void move(int n, char from, char inter, char to)
{
    if (n==1) System.out.print("move 1 from "+from+" to "+to);
    else
    {
        move(n-1,from,to,inter)
        System.out.print("move "+n+" from "+from+" to "+to);
        move(n-1,inter,from,to)
    }
}
```

Output for n=3, and A, B, C Towers

```
move(1, ‘A’, ‘B’, ‘C’)  
move 1 from A to C  
move 2 from A to B  
move(1, ‘C’, ‘A’, ‘B’)  
move 1 from C to B  
move 3 from A to C  
move(2, ‘B’, ‘A’, ‘C’)  
move(1, ‘B’, ‘C’, ‘A’)  
move 1 from B to A  
move 2 from B to C  
move(1, ‘A’, ‘B’, ‘C’)  
move 1 from A to C  
End of 2  
End of 3
```

Complicated problem solved using such small amount of code
MergeSort

- This is a much more efficient sorting technique than the ones seen before in class!
- Bubble, Selection, Insertion sorting algorithms are all \( O(N^2) \)
- MergeSort uses a divide and conquer strategy
  - Divide: split the list into two or more equal sublists
  - Conquer: Merge the sorted sub-list
  - Can be done recursively
MergeSort - overview

Divide

Conquer (sorting + merging)
The heart of MergeSort is the merging of already two sorted lists

Merging of sorted lists A and B creates a new one C

Remarks:

- The number of comparisons and copies is $O(N)$ … $N$ is the size of the new list (there is less comparisons than copies)
- Here, the merging does not work 'in-place' (additional memory needed)
### MergeSort - Examples

- **For fun:** [https://www.youtube.com/watch?v=XaqR3G_NVoo](https://www.youtube.com/watch?v=XaqR3G_NVoo)

- **To do:** Test Java applet MergeSort.html
public void recMergeSort(int[] array, int lower, int upper) {
    if (lower==upper)  // no need to sort-base step
        return;
    else {
        int mid=(lower+upper)/2;  // find midpoint
        recMergeSort(array,lower,mid);  // sort low-half
        recMergeSort(array,mid+1,upper);  // sort high-half
        merge(array,lower,mid+1,upper);  // merge them- textbook page 289
    }
}
MergeSort - Efficiency

- How many levels (round of merging)?
  - \( \log_2(N) \rightarrow O(\log N) \)
- Each round of merging needs N comparisons/copies \( \rightarrow O(N) \)
- Total is \( O(N\log N) \)

- Main drawback: additional memory needed
Most recursive strategy can be replaced by iterations

However, recursion produces elegant short programs that can handle complex tasks (Hanoi or MergeSort contain 2 recursion calls)

Recursion is similar to a stack-based approach

Some remarks about divide and conquer strategy:
  - Can indeed be solved using recursion... but
  - You should first think 'parallelism' (sub-problems can be solved in parallel)

Computer scientists (with focus on data structures) like recursion, but computational scientists (like me) avoid them