Divide-and-Conquer algorithms

Divide-and-Conquer algorithms

The same as recursive algorithms.

Divide-and-Conquer algorithms

The same as recursive algorithms.

A general algorithm development method:

1. Divide the problem into smaller subproblems (of the same type).
2. Solve the subproblems using recursion (i.e. call the algorithm itself, but with the smaller inputs).
3. Construct a solution to the problem based on the solution of the subproblems.

Base case: Problems of smallest size are solved directly (without recursion).

Divide and Conquer

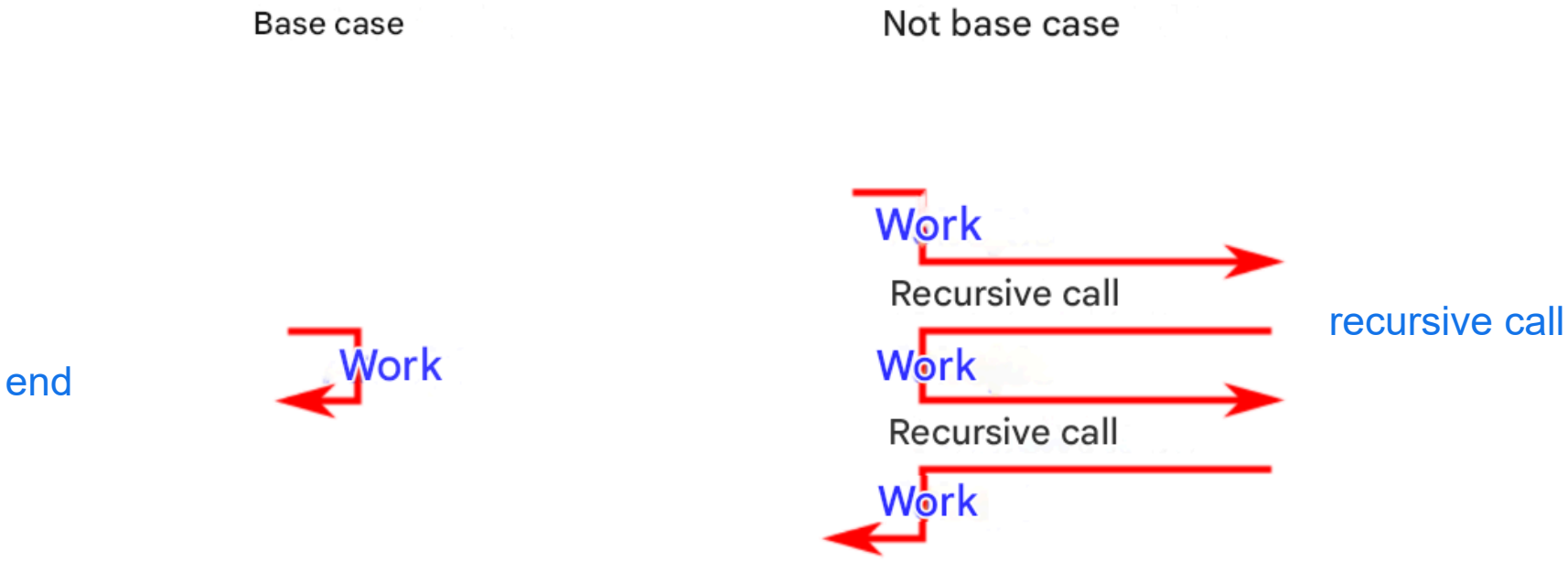
General structure of the code:

```
If  base case
    Local work (solve problem of base size)
Else
    Local work (e.g. build one or more subproblems)
    Recursive call
    Local work (e.g. use the answer to build next subproblem)
    Recursive call
    Local work (solve main problem based on answers to the sub-problems)
```

(It does not always have to be two recursive calls. Some recursive algorithms have only one, while others have more than two.).

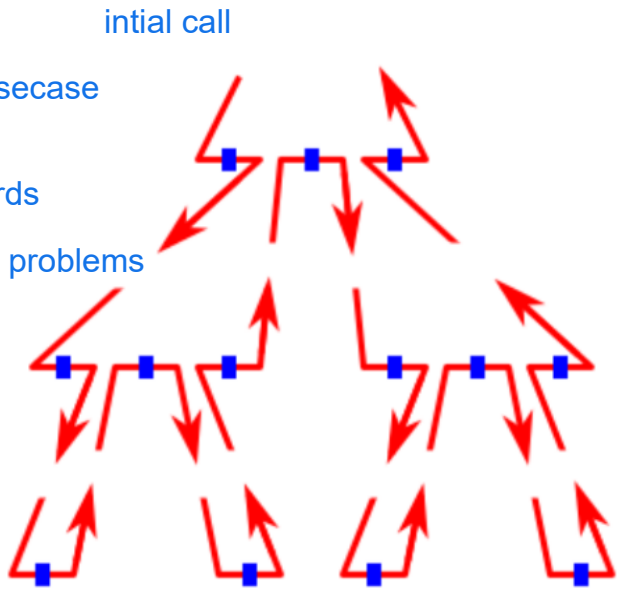
Divide-and-Conquer, flow of control

Flow of control (Locally, for a single call of the algorithm):



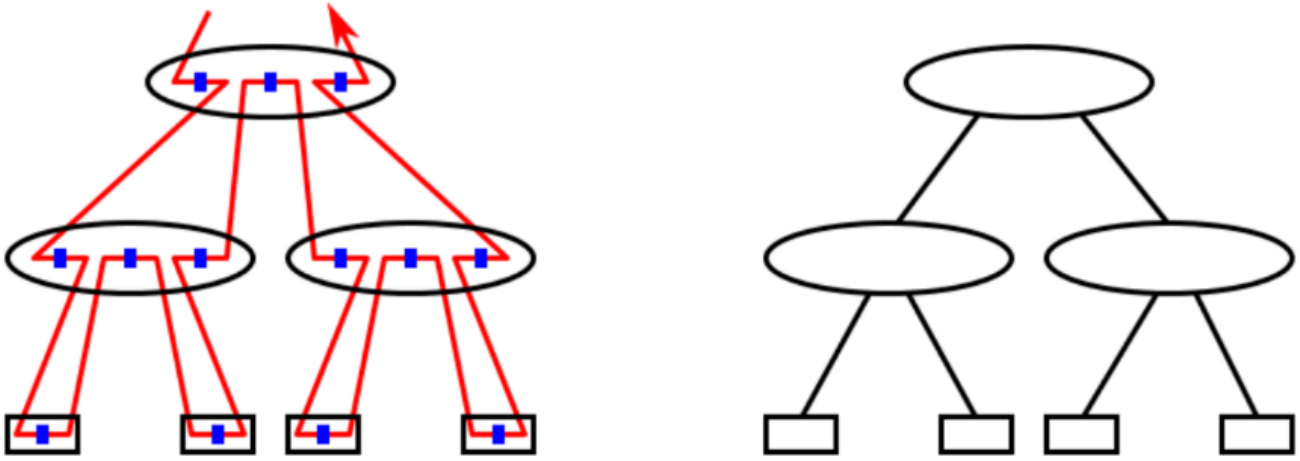
Divide-and-Conquer, Executed work

Global flow of control:



Recursion trees

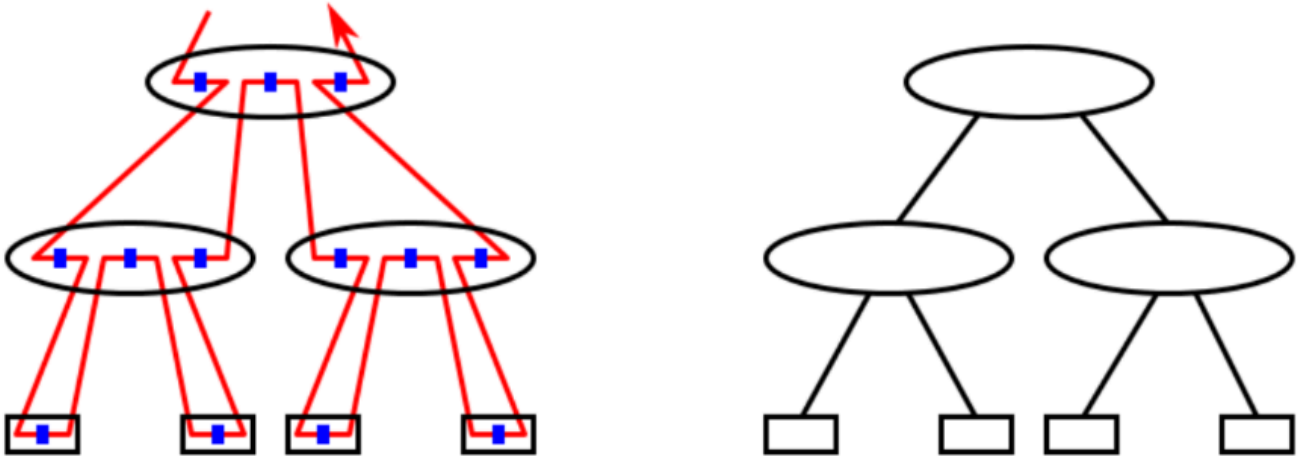
Global flow of control =recursion trees:



A node = a single call of the algorithm.

Recursion trees

Global flow of control =recursion trees:



A node = a single call of the algorithm.

Remember: All calls along a path from the root to active calls are "in progress" but paused. Their local variables and other state are stored (by the operating system) on a stack, so the execution of different calls does not get mixed up.

- Calling a child in the recursion tree = **push** onto the stack.
- Completion of a child's execution = **pop** from the stack.

stack, like a stack of plates, list in first out, LIFO