Algorithmic Techniques

#### **TOPICS**

1. Recursion
2. Backtracking
3. Divide and Conquer

# Recursion

Any function that calls itself is said to be recursive. A recursive method solves a problem by calling a copy of itself to work on a smaller problem. Recursive code tend to be shorter and elegant. But every recursive step requires extra memory (every iterative step does not) and hence recursive solutions tend to be less efficient than its iterative counterpart.

**Basic Format:** def func():  
 if base condition:  
 return value  
 else:  
 return func(changed parameter)

# Backtracking

Backtracking is a more refined and optimized version of **brute force** that intelligently explores the search space, avoiding unnecessary work whenever possible. This is done by generating a **state space trees** and searching for a solution via **Depth First Search**. Backtracking speeds the exhaustive search by cutting tree branches selectively (pruning).

# Divide and conquer

Take a (big) problem and divide it into smaller parts. Now solve these parts. This is **divide**. Now take the output from these smaller problems and combine them to get the final output. This is the **conquer** step.

Basic Format

def DAC(P):

if P is a small problem:

Execute(P)

else:

Divide P into P1,P2 ... Pk

Apply DAC(P1), DAC(P2) ... DAC(Pk)

Combine [DAC(P1), DAC(P2) ... DAC(Pk)]