Examples of “Divide-and-Conquer”
Example: Efficiently computing the power function

- The fibonacci example showed that sometimes recursion can be extremely inefficient compared.

- However, sometimes a recursive-approach can lead to tremendous gains in efficiency.

- We will see several important examples of this.

- Problem: Given a real number $a$ and a nonnegative integer $n$, compute $a^n$.  


Computing \( a^n \) efficiently.

- It is easy to write a loop that performs \( n-1 \) multiplications to compute \( a^n \).

- However, we can compute \( a^n \) much more efficiently.

**Example:** To compute \( a^{32} \), we could compute \( a^{16} \) first and then use one multiplication to square it. To compute \( a^{16} \), we would compute \( a^8 \) and then use one multiplication to square it...
It takes 5 multiplications to compute $a^{32}$

For each recursive call, we perform just one multiplication, but manage to reduce the problem size to $\frac{1}{2}$ its previous size.

This is a common theme in many efficient algorithms: if we can do just a little work and manage to shrink the problem size down to $\frac{1}{2}$ its original size, then we have an efficient solution.
What about an odd power?

- **Example:** Compute $a^{39}$

- We could compute $a^{19}$, square it (using one multiplication) to get $a^{38}$ and then use another multiplication to compute $a^{39}$.

- Thus using 2 multiplications we can reduce the power to less than $1/2$ of what it was earlier.
def power(a, n):
    # Base Cases
    if n == 0:
        return 1
    if n == 1:
        return a
    # Recursive Case: even n
    if n % 2 == 0:
        temp = power(a, n/2)
        return temp*temp
    # Recursive Case: odd n
    if n % 2 == 1:
        temp = power(a, n/2)
        return temp*temp*a
The *Divide-and-Conquer* Paradigm

- This is an important algorithmic technique to efficiently solving computational problems.

- *It is commonly used for*
  - Efficient sorting
  - Multiplying large numbers
  - Multiplying matrices
  - Finding a closest pair of points in Euclidean space

- *It is usually implemented using recursion.*
Binary Search
An example of Divide-and-Conquer

- In `binarySearch(L, k)`, we make one comparison: `k` compared to `L[mid]`.
- Based on the outcome of this comparison, we either stop, search the left half, or search the right half.
- Thus the problem of searching for `k` in `L` is reduced to search for `k` in `L[:mid]` or `L[mid+1:]`. 
Divide-and-Conquer Paradigm

- **Divide step:** Partition the problem into sub-problems.

- **Conquer step:** Solve each sub-problem separately.

- **Combine step:** Combine the solutions of the sub-problems into a solution of the original problem.
Sorting via Divide-and-Conquer

- Algorithms such as *selection sort*, *insertion sort*, *bubble sort*, etc. are all extremely slow for large lists.

- This is because they take about $N^2$ time on a list of size $N$.

- Algorithms that are based on “divide-and-conquer” such as *merge sort* or *quick sort* are much faster.

- These algorithms run in about $N \log N$ time on a list of size $N$. 