Recursion

Josh Wilcox (jw14g24@soton.ac.uk)

February 5, 2025

Contents

1 What is Recursion

- Strategy that reduces solving a problem into solving smaller problems **of the same time**
  - Repeated until a trivial case is reached
- Can be used to define a function using *references to itself*
  - Factorial - $n! = n \cdot (n-1)!$
  - Known as **self-referential** definitions

1.1 Structure of Recursion

- **Base Case** - Or boundary case, where solving the problem is trivial
- **Recursive Case** - Self-referential part driving the problem towards the base case
- Should be reminiscent of proof by induction
  - Many mathematical functions are defined recursively
  - Recursive mathematical function's properties can typically be proved by induction

2 Programming Recursively

- Most modern programming languages allow you to program recursively
- They allow methods to be defined in terms of themselves

2.1 Factorial

```java
public static long factorial(long n) throws IllegalArgumentException
{
    if (n<0){
        throw new IllegalArgumentException();
    }
    else if (n==0){ // The base case - will always be run
        return 1;
    }
    else{
        return n* factorial(n-1); // Recursive definition of factorial
    }
}


```

2.2 Integer Power

- Use the observation that $(x^n)^2 = x^{2n}$

```java
public static double power(double x, long n)
{
    return n < 0 ? 1 / power(x,-n) // Negative Power
    : n == 0 ? 1 // Special Case
    : n == 1 ? x // Base Case
    : n%2 == 0 ? (x = power(x, n/2)) * x // Even power
```

```java
        : x * power(x, n-1); // Odd power
}
```

$$0.95^{25} = 0.95 \times (0.95)^{24} = 0.95 \times \left((0.95)^{12}\right)^2 = 0.95 \times \left(\left((0.95)^6\right)^2\right)^2$$
$$= 0.95 \times \left(\left(\left((0.95)^3\right)^2\right)^2\right)^2 = 0.95 \times \left(\left(\left(0.95 \times (0.95)^2\right)^2\right)^2\right)^2$$

2.3 Greatest Common Divisor

$$gcd(A, B) = gcd(B, AmodB)$$
$$gcd(A, 0) = A$$

```java
public static long gcd(long a, long b)
{
    if (b==0)
        return a;
    else
        return gcd(b, a%b);
}
```

3 Writing Recursive Programs

- Always **start with the base case**
- Recursive case can call itself possibly many times
  - Ensure recursive calls use a 'smaller' problem - converges to the desired result
  - Assume the smaller problem can be solved

3.1 When not to use Recursion

- Do not recurse when there is a lot of overlap in the smaller problems

4 Analysis of Recursion

- Recursion can be used to compute the running time of a recursive program
- This can be done by denoting the time taken to solve a problem of size n by $T(n)$