Part 2: Analysis of Recursive Algorithms
Outline

Induction on Recurrences

Complexity of Recursive Algorithms
Recursively Defined Functions

Define:

\[ f(n) = \begin{cases} 
2 & n = 0 \\
7 & n = 1 \\
2f(n - 2) + f(n - 1) & n > 1 
\end{cases} \]

Write out a few values of \( f(n) \). We want to show \( f(n) < 2^g(n) \). Conjecture for \( g(n) \)?
\[ f(n) < 2^{n+2} \]
Define:

\[ f(n) = \begin{cases} 
  3 & \text{if } n = 0 \\
  6 & \text{if } n = 1 \\
  5f(n - 1) - 6f(n - 2) + 2 & \text{if } n > 1
\end{cases} \]

Prove that \( f(n) = 1 + 2^n + 3^n \) for all \( n \geq 0 \).
Reminders about Algorithm Complexity

- Running time is measured by counting steps in an algorithm
- Sufficient to count chunks of instructions (sequences that are always executed together in constant time) as one step
- Measure as a function $T(n)$ of input “size” $n$ (number of input elements)
- For now, just concerned with worst-case (maximum over all inputs of the same size)
- Often, there is no simple algebraic expression for $T(n)$
Reminders about Algorithm Complexity

Asymptotic Notation

- Upper bound
- Lower bound
- Tight bound
Recursive Algorithms

Recursive Binary Search

RecBinSearch(x, A, b, e):
1. if b == e:
2. if x <= A[b]: return b
3. else: return e + 1
else:
4. m = (b + e) // 2 # midpoint
5. if x <= A[m]:
6. return RecBinSearch(x, A, b, m)
else:
7. return RecBinSearch(x, A, m+1, e)
Complexity of RecBinSearch
Solving Recurrence Relations

Example: Recursive Factorial

```
Fact(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * Fact(n-1)
```
Solving Recurrence Relations

Unwinding $T(n)$