Algorithmic Strategies 2024/25

Week 2 – Recursion

· U C ·

Universidade de Coimbra

Outline

1. Time complexity

Time complexity

- The time complexity of recursive programs may not be easy to derive
- Guess the recurrence, e.g., by *unrolling* it, or use the Master Theorem, if applicable
- Reading material: Cormen at al, Introduction to Algorithms, chapter 4.

Time complexity of recursion

A recursive algorithm to print all binary strings of size n

Function *binaryString*(n, S)
  if $n = 0$ **then** {base case}
    print S
  else
    push$('0', S)$
    binaryString$(n-1, S)$ {1st recursive step}
    pop(S)
    push$('1', S)$
    binaryString$(n-1, S)$ {2nd recursive step}
    pop(S)
  return

What is the time complexity of this algorithm?

A first guess: Unrolling the recurrence

Let T_s denote the cost of stack operations at each recursive step.

$$
\begin{aligned}
T(n) &= 2T(n-1) + T_s \\
&= 2(2T(n-2) + T_s) + T_s \\
&= 4T(n-2) + 3T_s \\
&= 4(2T(n-3) + T_s) + 3T_s \\
&= 8T(n-3) + 7T_s \\
& \dots \\
&= 2^k T(n-k) + (2^k - 1)T_s \\
& \dots \\
&= 2^n T(0) + (2^n - 1)T_s \in O(2^{n+1})
\end{aligned}
$$

Time complexity of recursion

Master Theorem

The time complexity of some recursive programs can be defined as:

$$T(n) = \begin{cases} aT(n/b) + n^c & \text{if } n > 1 \\ d & \text{if } n = 1 \end{cases}$$

- n is the problem size
- $a \geq 1$ is the number of subproblems at each recursive step
- n/b, $b > 1$, is the size of each subproblem
- $d \geq 0$ is cost of the base case
- n^c is the cost of each recursive step

Master Theorem

$$T(n) = \begin{cases} aT(n/b) + n^c & \text{if } n > 1 \\ d & \text{if } n = 1 \end{cases}$$

- If $\log_b a < c,\ T(n) = \Theta(n^c)$
- If $\log_b a = c,\ T(n) = \Theta(n^c \log n)$
- If $\log_b a > c$, $T(n) = \Theta(n^{\log_b a})$

Time complexity of recursion

Merge Sort

```
Function MergeSort(A, low, high)
  if low = high then                                  {base case}
    return
  else
    mid = (low + high)/2
    B = MergeSort(A, low, mid)                        {1st recursive step}
    C = MergeSort(A, mid + 1, high)                   {2nd recursive step}
    return Merge(B, C)
```

What is the time complexity for an array of size n?

Time complexity of recursion

Master Theorem

$$T(n) = \begin{cases} 2T(n/2) + O(n) & \text{if } n > 1 \\ O(1) & \text{if } n = 1 \end{cases}$$

- If $\log_b a < c,\ T(n) = \Theta(n^c)$
- If $\log_b a = c,\ T(n) = \Theta(n^c \log n)$
- If $\log_b a > c$, $T(n) = \Theta(n^{\log_b a})$

The time complexity of merge sort is $\Theta(n \log n)$.

Time complexity of recursion

Binary Search

Function $binarySearch(A, low, high, v)$
    $mid = (low + high)/2$
    if $A[mid] = v$ **then** {base case}
        return mid
    if $low = mid$ **then** {base case}
        return -1
    if $A[mid] \leq v$ **then**
        return $binarySearch(A, low, mid, v)$ {1st recursive step}
    else
        return $binarySearch(A, mid + 1, high, v)$ {2nd recursive step}

What is the time complexity for an array of size n?

Time complexity of recursion

Master Theorem

$$T(n) = \begin{cases} T(n/2) + O(1) & \text{if } n > 1 \\ O(1) & \text{if } n = 1 \end{cases}$$

- If $\log_b a < c,\ T(n) = \Theta(n^c)$
- If $\log_b a = c,\ T(n) = \Theta(n^c \log n)$
- If $\log_b a > c$, $T(n) = \Theta(n^{\log_b a})$

The time complexity of binary search is $\Theta(\log n)$.

Balanced Binary Tree Traversal

```
Function binaryTreeSearch(node)
    print(node)
    if left(node) ≠ NULL then
        binaryTreeSearch(left(node))                 {1st recursive step}
    if right(node) ≠ NULL then
        binaryTreeSearch(right(node)                 {2nd recursive step}
```

What is the time complexity for a tree with n nodes?

Time complexity of recursion

Master Theorem

$$T(n) = \begin{cases} 2T(n/2) + O(1) & \text{if } n > 1 \\ O(1) & \text{if } n = 1 \end{cases}$$

- If $\log_b a < c,\ T(n) = \Theta(n^c)$
- If $\log_b a = c,\ T(n) = \Theta(n^c \log n)$
- If $\log_b a > c,\ T(n) = \Theta(n^{\log_b a})$

The time complexity of balanced binary tree traversal is $\Theta(n)$.

Strassen's Algorithm for Matrix Multiplication

Function *Strassen* $\left(A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}, n\right)$

if $n = 1$ **then**

$C = A \cdot B$

else

$M_1 = Strassen(A_{11} + A_{22}, B_{11} + B_{22}, n/2)$

$M_2 = Strassen(A_{21} + A_{22}, B_{11}, n/2)$

$M_3 = Strassen(A_{11}, B_{12} - B_{22}, n/2)$

$M_4 = Strassen(A_{22}, B_{21} - B_{11}, n/2)$

$M_5 = Strassen(A_{11} + A_{12}, B_{22}, n/2)$

$M_6 = Strassen(A_{21} - A_{11}, B_{11} + B_{12}, n/2)$

$M_7 = Strassen(A_{12} - A_{22}, B_{21} + B_{22}, n/2)$

$$C = \begin{bmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 - M_2 + M_3 + M_6 \end{bmatrix}$$

return C

Master Theorem

$$T(n) = \begin{cases} 7T(n/2) + O(n^2) & \text{if } n > 1 \\ O(1) & \text{if } n = 1 \end{cases}$$

- If $\log_b a < c,\ T(n) = \Theta(n^c)$
- If $\log_b a = c,\ T(n) = \Theta(n^c \log n)$
- If $\log_b a > c$, $T(n) = \Theta(n^{\log_b a})$

The time complexity is $\Theta(n^{\log_2 7}) \approx O(n^{2.8074})$.