Data Structures I : O notation (recursion)

UNIVERSIDAD EAFIT

Mauricio Toro
Department of Systems and Informatics
Universidad EAFIT

Cocktail of the day: Margarita

Disclaimer: Keep alcohol out of the hands of minors.

Cocktail of the day: Margarita

- 35 ml Tequila
- 20 ml Cointreau
- 15 ml lime juice

Microsoft Office documents are trees

https://msdn.microsoft.com/en-us/library/bb266220(v=office.12).aspx

1. Number of instructions: $T(n)$
2. Asymptotic analysis: O notation
3. Rule of sums
4. Rule of products

Sum the elements of an array

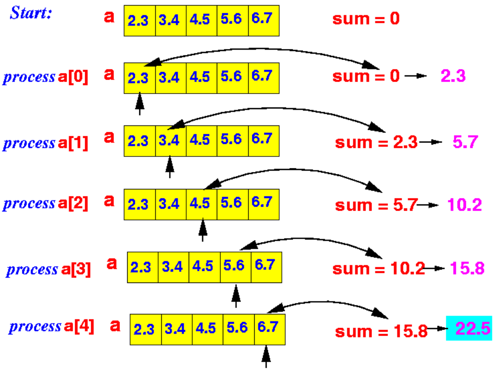


Figure: Array sum

Sum the elements of an array (2)

```
Proceso ArraySum
  Definir i, n, sum, A Como Entero;
  Leer n;
  sum <- 0;
  Dimension A[n];
  Para i <- 0 hasta n-1 con paso 1 Hacer
    sum <- sum + A[1];
  FinPara
  Escribir sum;
FinProceso
```

Sum the elements of an array (3)

```
Proceso ArraySum
  Definir i, n, sum, A Como Entero;
  Leer n;
  sum <- 0;
  Dimension A[n];
  Para i <- 0 hasta n-1 con paso 1 Hacer
    sum <- sum + A[1];
  FinPara
  Escribir sum;
FinProceso
```

Number of instructions $T(n) = ?$

Sum the elements of an array (4)

```
Proceso ArraySum
  Definir i, n, sum, A Como Entero;       // 4
  Leer n;                                 // 1
  sum <- 0;                               // 1
  Dimension A[n];                         // 1
  Para i <- 0 hasta n-1 con paso 1 Hacer  // 1*n
    sum <- sum + A[1];                    // 3*n
  FinPara
  Escribir sum;                           // 1
FinProceso
```

$T(n) = 7n + 5$

Sum the elements of an array (5)

```
Proceso ArraySum
  Definir i, n, sum, A Como Entero;     // 4
  Leer n;                               // 1
  sum <- 0;                             // 1
  Dimension A[n];                       // 1
  Para i <- 0 hasta n-1 con paso 1 Hacer  // 1*n
    sum <- sum + A[1];                  // 3*n
  FinPara
  Escribir sum;                         // 1
FinProceso
```

$T(n) = 7n + 5$ is $O(n)$

Recursive sum of an array

```
SubProceso sum <- ArraySum( A, n )
  Definir i, sum Como Entero;
  Si n = 0 Entonces
    sum <- A[0];
  Sino
    sum <- A[n] + ArraySum(A, n-1);
  FinSi
FinSubProceso
```

$T(n) = ?$

Recursive sum of an array (2)

```
SubProceso sum <- ArraySum( A, n )
  Definir i, sum Como Entero;          // 2
  Si n = 0 Entonces                    // 1
    sum <- A[0];                       // 2
  Sino
    sum <- A[n] + ArraySum(A, n-1); // 4 + T(n-1)
```

$$T(n) = \begin{cases} 5 & \textit{if} \quad n = 0 \\ 7 + T(n-1) & \textit{if} \quad n > 0 \end{cases}$$

Recursive sum of an array (3)

$$T(n) = \begin{cases} 5 & \textit{if} \quad n = 0 \\ 7 + T(n-1) & \textit{if} \quad n > 0 \end{cases}$$

is $O(n)$

Recurrence relations

- An order d linear homogeneous recurrence relation with constant coefficients is an equation of the form:
- $T(n) = c_1 a_{n-1} + c_2 a_{n-2} + \ldots + c_d a_{n-d}$
- For example, an equation of order 1 is

$$T(n) = \begin{cases} 5 & \text{if} \quad n = 0 \\ 7 + T(n-1) & \text{if} \quad n > 0 \end{cases}$$

Recurrence relations

- An order d linear homogeneous recurrence relation with constant coefficients is an equation of the form:
- $T(n) = c_1 a_{n-1} + c_2 a_{n-2} + \ldots + c_d a_{n-d}$
- For example, an equation of order 1 is

$$T(n) = \begin{cases} 5 & if \quad n = 0 \\ 7 + T(n-1) & if \quad n > 0 \end{cases}$$

Solving recurrence relations

- $T(n) = 7 + T(n-1)$
- $T(n) = 7 + (7 + T(n-2))$, by induction
- $T(n) = 7 + (7 + (7 + T(n-3)))$, by induction
- $T(n) = \underbrace{7 + (7 + (7+}_{7\times 3} + T(n-3)))$
- $T(n) = \underbrace{7 + 7 + ... + 7}_{7\times n} + T(n-n)))$, by induction
- $T(n) = 7n + T(0)$ and $T(0) = 5$
- $T(n) = 7n + 5$, by replacing $T(0)$ by 5

Solving recurrence relations

- $T(n) = 7 + T(n-1)$
- $T(n) = 7 + (7 + T(n-2))$, by induction
- $T(n) = 7 + (7 + (7 + T(n-3)))$, by induction
- $T(n) = \underbrace{7 + (7 + (7+}_{7\times 3} + T(n-3)))$
- $T(n) = \underbrace{7 + 7 + \ldots + 7}_{7\times n} + T(n-n)))$, by induction
- $T(n) = 7n + T(0)$ and $T(0) = 5$
- $T(n) = 7n + 5$, by replacing $T(0)$ by 5

Solving recurrence relations

- $T(n) = 7 + T(n-1)$
- $T(n) = 7 + (7 + T(n-2))$, by induction
- $T(n) = 7 + (7 + (7 + T(n-3)))$, by induction
- $T(n) = \underbrace{7 + (7 + (7+}_{7\times 3} + T(n-3)))$
- $T(n) = \underbrace{7 + 7 + ... + 7}_{7\times n} + T(n-n)))$, by induction
- $T(n) = 7n + T(0)$ and $T(0) = 5$
- $T(n) = 7n + 5$, by replacing $T(0)$ by 5

- $T(n) = 7 + T(n-1)$
- $T(n) = 7 + (7 + T(n-2))$, by induction
- $T(n) = 7 + (7 + (7 + T(n-3)))$, by induction
- $T(n) = \underbrace{7 + (7 + (7+}_{7\times 3} + T(n-3)))$
- $T(n) = \underbrace{7 + 7 + \ldots + 7}_{7\times n} + T(n-n)))$, by induction
- $T(n) = 7n + T(0)$ and $T(0) = 5$
- $T(n) = 7n + 5$, by replacing $T(0)$ by 5

Solving recurrence relations

- $T(n) = 7 + T(n-1)$
- $T(n) = 7 + (7 + T(n-2))$, by induction
- $T(n) = 7 + (7 + (7 + T(n-3)))$, by induction
- $T(n) = \underbrace{7 + (7 + (7+}_{7\times 3} + T(n-3)))$
- $T(n) = \underbrace{7 + 7 + \ldots + 7}_{7\times n} + T(n-n)))$, by induction
- $T(n) = 7n + T(0)$ and $T(0) = 5$
- $T(n) = 7n + 5$, by replacing $T(0)$ by 5

Solving recurrence relations

- $T(n) = 7 + T(n-1)$
- $T(n) = 7 + (7 + T(n-2))$, by induction
- $T(n) = 7 + (7 + (7 + T(n-3)))$, by induction
- $T(n) = \underbrace{7 + (7 + (7+}_{7\times 3} + T(n-3)))$
- $T(n) = \underbrace{7 + 7 + ... + 7}_{7\times n} + T(n-n)))$, by induction
- $T(n) = 7n + T(0)$ and $T(0) = 5$
- $T(n) = 7n + 5$, by replacing $T(0)$ by 5

Solving recurrence relations

- $T(n) = 7 + T(n-1)$
- $T(n) = 7 + (7 + T(n-2))$, by induction
- $T(n) = 7 + (7 + (7 + T(n-3)))$, by induction
- $T(n) = \underbrace{7 + (7 + (7+}_{7\times 3} + T(n-3)))$
- $T(n) = \underbrace{7 + 7 + ... + 7}_{7\times n} + T(n-n)))$, by induction
- $T(n) = 7n + T(0)$ and $T(0) = 5$
- $T(n) = 7n + 5$, by replacing $T(0)$ by 5

Sum elements of an array: Proof

1. $T(n) = 7n + 5$
2. $7n + 5$ is $O(7n + 5)$, by Definition of O
3. $O(7n + 5) = O(7n)$, by Rule of Sums
4. $O(7n) = O(n)$, by Rule of Products
5. Therefore, $T(n) = 7n + 5$ is $O(n)$.

Sum elements of an array: Proof

1. $T(n) = 7n + 5$
2. $7n + 5$ is $O(7n + 5)$, by Definition of O
3. $O(7n + 5) = O(7n)$, by Rule of Sums
4. $O(7n) = O(n)$, by Rule of Products
5. Therefore, $T(n) = 7n + 5$ is $O(n)$.

Sum elements of an array: Proof

1. $T(n) = 7n + 5$
2. $7n + 5$ is $O(7n + 5)$, by Definition of O
3. $O(7n + 5) = O(7n)$, by Rule of Sums
4. $O(7n) = O(n)$, by Rule of Products
5. Therefore, $T(n) = 7n + 5$ is $O(n)$.

Sum elements of an array: Proof

1. $T(n) = 7n + 5$
2. $7n + 5$ is $O(7n + 5)$, by Definition of O
3. $O(7n + 5) = O(7n)$, by Rule of Sums
4. $O(7n) = O(n)$, by Rule of Products
5. Therefore, $T(n) = 7n + 5$ is $O(n)$.

Sum elements of an array: Proof

1. $T(n) = 7n + 5$
2. $7n + 5$ is $O(7n + 5)$, by Definition of O
3. $O(7n + 5) = O(7n)$, by Rule of Sums
4. $O(7n) = O(n)$, by Rule of Products
5. Therefore, $T(n) = 7n + 5$ is $O(n)$.

Recursive maximum element of an array

```
SubProceso max <- ArrayMax( A, n )
  Definir i, max, temp Como Entero;
  max <- A[n]; // Si n = 0, max <- A[0]
  Si n != 0 Entonces
    temp <- ArrayMax(A, n-1);
    Si temp > max Entonces
      max <- temp;
```

$T(n) = ?$

Recursive maximum element of an array (2)

```
SubProceso max <- ArrayMax( A, n )
  Definir i, max, temp Como Entero;   // 3
  max <- A[n];                        // 2
  Si n != 0 Entonces                  // 1
    temp <- ArrayMax(A, n-1);         // 1 + T(n-1)
    Si temp > max Entonces            // 1
      max <- temp;                    // 1
```

$$T(n) = \begin{cases} 6 & \text{if} \quad n = 0 \\ 9 + T(n-1) & \text{if} \quad n > 0 \end{cases}$$

Recursive maximum element of an array (3)

```
SubProceso max <- ArrayMax( A, n )
  Definir i, max, temp Como Entero;   // 3
  max <- A[n];                        // 2
  Si n != 0 Entonces                  // 1
    temp <- ArrayMax(A, n-1);         // 1 + T(n-1)
    Si temp > max Entonces            // 1
      max <- temp;                    // 1
```

$$T(n) = \begin{cases} 6 & \textit{if} \quad n = 0 \\ 9 + T(n-1) & \textit{if} \quad n > 0 \end{cases} = 9n + 6$$

Recursive maximum element: Proof

- $T(n) = 9 + T(n-1)$
- $T(n) = 9 + (9 + T(n-2))$, by induction
- $T(n) = 9 + (9 + (9 + T(n-3)))$, by induction
- $T(n) = \underbrace{9 + (9 + (9+}_{9\times 3} + T(n-3)))$
- $T(n) = \underbrace{9 + 9 + ... + 9}_{9\times n} + T(n-n)))$, by induction
- $T(n) = 9n + T(0)$ and $T(0) = 6$
- $T(n) = 9n + 6$, by replacing $T(0)$ by 6

Recursive maximum element: Proof

- $T(n) = 9 + T(n-1)$
- $T(n) = 9 + (9 + T(n-2))$, by induction
- $T(n) = 9 + (9 + (9 + T(n-3)))$, by induction
- $T(n) = \underbrace{9 + (9 + (9+}_{9\times 3} + T(n-3)))$
- $T(n) = \underbrace{9 + 9 + ... + 9}_{9\times n} + T(n-n)))$, by induction
- $T(n) = 9n + T(0)$ and $T(0) = 6$
- $T(n) = 9n + 6$, by replacing $T(0)$ by 6

Recursive maximum element: Proof

- $T(n) = 9 + T(n-1)$
- $T(n) = 9 + (9 + T(n-2))$, by induction
- $T(n) = 9 + (9 + (9 + T(n-3)))$, by induction
- $T(n) = \underbrace{9 + (9 + (9+}_{9\times 3} + T(n-3)))$
- $T(n) = \underbrace{9 + 9 + ... + 9}_{9\times n} + T(n-n)))$, by induction
- $T(n) = 9n + T(0)$ and $T(0) = 6$
- $T(n) = 9n + 6$, by replacing $T(0)$ by 6

Recursive maximum element: Proof

- $T(n) = 9 + T(n-1)$
- $T(n) = 9 + (9 + T(n-2))$, by induction
- $T(n) = 9 + (9 + (9 + T(n-3)))$, by induction
- $T(n) = \underbrace{9 + (9 + (9+}_{9\times 3} + T(n-3)))$
- $T(n) = \underbrace{9 + 9 + ... + 9}_{9\times n} + T(n-n)))$, by induction
- $T(n) = 9n + T(0)$ and $T(0) = 6$
- $T(n) = 9n + 6$, by replacing $T(0)$ by 6

UNIVERSIDAD EAFIT

Recursive maximum element: Proof

- $T(n) = 9 + T(n-1)$
- $T(n) = 9 + (9 + T(n-2))$, by induction
- $T(n) = 9 + (9 + (9 + T(n-3)))$, by induction
- $T(n) = \underbrace{9 + (9 + (9+}_{9\times 3} + T(n-3)))$
- $T(n) = \underbrace{9 + 9 + ... + 9}_{9\times n} + T(n-n)))$, by induction
- $T(n) = 9n + T(0)$ and $T(0) = 6$
- $T(n) = 9n + 6$, by replacing $T(0)$ by 6

Recursive maximum element: Proof

- $T(n) = 9 + T(n-1)$
- $T(n) = 9 + (9 + T(n-2))$, by induction
- $T(n) = 9 + (9 + (9 + T(n-3)))$, by induction
- $T(n) = \underbrace{9 + (9 + (9+}_{9 \times 3} + T(n-3)))$
- $T(n) = \underbrace{9 + 9 + \ldots + 9}_{9 \times n} + T(n-n)))$, by induction
- $T(n) = 9n + T(0)$ and $T(0) = 6$
- $T(n) = 9n + 6$, by replacing $T(0)$ by 6

Recursive maximum element: Proof

- $T(n) = 9 + T(n-1)$
- $T(n) = 9 + (9 + T(n-2))$, by induction
- $T(n) = 9 + (9 + (9 + T(n-3)))$, by induction
- $T(n) = \underbrace{9 + (9 + (9+}_{9\times 3} + T(n-3)))$
- $T(n) = \underbrace{9 + 9 + \ldots + 9}_{9\times n} + T(n-n)))$, by induction
- $T(n) = 9n + T(0)$ and $T(0) = 6$
- $T(n) = 9n + 6$, by replacing $T(0)$ by 6

Recursive maximum element of an array (4)

$$T(n) = \begin{cases} 6 & \textit{if} \quad n = 0 \\ 9 + T(n-1) & \textit{if} \quad n > 0 \end{cases} = 9n + 6$$

is $O(n)$

Sum elements of an array: Proof (2)

1. $T(n) = 9n + 6$
2. $9n + 6$ is $O(9n + 6)$, by Definition of O
3. $O(9n + 6) = O(9n)$, by Rule of Sums
4. $O(9n) = O(n)$, by Rule of Products
5. Therefore, $T(n) = 9n + 6$ is $O(n)$.

Sum elements of an array: Proof (2)

1. $T(n) = 9n + 6$
2. $9n + 6$ is $O(9n + 6)$, by Definition of O
3. $O(9n + 6) = O(9n)$, by Rule of Sums
4. $O(9n) = O(n)$, by Rule of Products
5. Therefore, $T(n) = 9n + 6$ is $O(n)$.

Sum elements of an array: Proof (2)

1. $T(n) = 9n + 6$
2. $9n + 6$ is $O(9n + 6)$, by Definition of O
3. $O(9n + 6) = O(9n)$, by Rule of Sums
4. $O(9n) = O(n)$, by Rule of Products
5. Therefore, $T(n) = 9n + 6$ is $O(n)$.

Sum elements of an array: Proof (2)

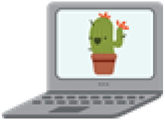
1. $T(n) = 9n + 6$
2. $9n + 6$ is $O(9n + 6)$, by Definition of O
3. $O(9n + 6) = O(9n)$, by Rule of Sums
4. $O(9n) = O(n)$, by Rule of Products
5. Therefore, $T(n) = 9n + 6$ is $O(n)$.

Sum elements of an array: Proof (2)

1. $T(n) = 9n + 6$
2. $9n + 6$ is $O(9n + 6)$, by Definition of O
3. $O(9n + 6) = O(9n)$, by Rule of Sums
4. $O(9n) = O(n)$, by Rule of Products
5. Therefore, $T(n) = 9n + 6$ is $O(n)$.

Recursive Fibonacci series

http://visualgo.net/recursion.html



https://plus.maths.org/content/life-and-numbers-fibonacci

Recursive Fibonacci series

```
SubProceso result <- Fibo( n )
  Definir result Como Entero;
  Si n <= 1 Entonces
    result <- n;
  Sino
    result <- Fibo(n-1) + Fibo(n-2);
```

$T(n) = ?$

Recursive Fibonacci series (2)

```
SubProceso result <- Fibo( n )
  Definir result Como Entero;  //1
  Si n <= 1 Entonces           //1
    result <- n;               //1
  Sino
    result<-Fibo(n-1)+Fibo(n-2); //2+T(n-1)+T(n-2)
```

$$T(n) = \begin{cases} 3 & \textit{if} \quad n < 1 \\ 4 + T(n-1) + T(n-2) & \textit{if} \quad n > 1 \end{cases}$$

Recursive Fibonacci series (2)

```
SubProceso result <- Fibo( n )
  Definir result Como Entero;  //1
  Si n <= 1 Entonces           //1
    result <- n;               //1
  Sino
    result<-Fibo(n-1)+Fibo(n-2); //2+T(n-1)+T(n-2)
```

$$T(n) = \begin{cases} 3 & if \quad n < 1 \\ 4 + T(n-1) + T(n-2) & if \quad n > 1 \end{cases} = (4+3)2^n+4$$

Recursive Fibonacci series : Proof (1)

- $T(n) = 4 + \underbrace{T(n-1) + T(n-2)}_{2^1 \text{ function calls}}$, by induction
- $T(n) = \underbrace{4 \times 3}_{4\times(2^2+1)} + \underbrace{T(n-2) + T(n-3) + T(n-3) + T(n-4)}_{2^2 \text{ function calls}}$
- $T(n) = \underbrace{4 \times 9}_{4\times(2^3+1)} + \underbrace{T(n-2) + T(n-4) + T(n-5) + \ldots}_{2^3 \text{ function calls}}$
- $T(n) = 4(2^n + 1) + T(n-n)2^n$, by induction
- $T(n) = (4+3)2^n + 4$, by replacing $T(0)$ by 3

Recursive Fibonacci series : Proof (1)

- $T(n) = 4 + \underbrace{T(n-1) + T(n-2)}_{2^1 \text{ function calls}}$, by induction
- $T(n) = \underbrace{4 \times 3}_{4\times(2^2+1)} + \underbrace{T(n-2) + T(n-3) + T(n-3) + T(n-4)}_{2^2 \text{ function calls}}$
- $T(n) = \underbrace{4 \times 9}_{4\times(2^3+1)} + \underbrace{T(n-2) + T(n-4) + T(n-5) + \ldots}_{2^3 \text{ function calls}}$
- $T(n) = 4(2^n + 1) + T(n-n)2^n$, by induction
- $T(n) = (4+3)2^n + 4$, by replacing $T(0)$ by 3

Recursive Fibonacci series : Proof (1)

- $T(n) = 4 + \underbrace{T(n-1) + T(n-2)}_{2^1 \text{ function calls}}$, by induction
- $T(n) = \underbrace{4 \times 3}_{4\times(2^2+1)} + \underbrace{T(n-2) + T(n-3) + T(n-3) + T(n-4)}_{2^2 \text{ function calls}}$
- $T(n) = \underbrace{4 \times 9}_{4\times(2^3+1)} + \underbrace{T(n-2) + T(n-4) + T(n-5) + \ldots}_{2^3 \text{ function calls}}$
- $T(n) = 4(2^n + 1) + T(n-n)2^n$, by induction
- $T(n) = (4+3)2^n + 4$, by replacing $T(0)$ by 3

Recursive Fibonacci series : Proof (1)

- $T(n) = 4 + \underbrace{T(n-1) + T(n-2)}_{2^1 \text{ function calls}}$, by induction
- $T(n) = \underbrace{4 \times 3}_{4\times(2^2+1)} + \underbrace{T(n-2) + T(n-3) + T(n-3) + T(n-4)}_{2^2 \text{ function calls}}$
- $T(n) = \underbrace{4 \times 9}_{4\times(2^3+1)} + \underbrace{T(n-2) + T(n-4) + T(n-5) + \ldots}_{2^3 \text{ function calls}}$
- $T(n) = 4(2^n + 1) + T(n-n)2^n$, by induction
- $T(n) = (4+3)2^n + 4$, by replacing $T(0)$ by 3

Recursive Fibonacci series : Proof (1)

- $T(n) = 4 + \underbrace{T(n-1) + T(n-2)}_{2^1 \text{ function calls}}$, by induction
- $T(n) = \underbrace{4 \times 3}_{4\times(2^2+1)} + \underbrace{T(n-2) + T(n-3) + T(n-3) + T(n-4)}_{2^2 \text{ function calls}}$
- $T(n) = \underbrace{4 \times 9}_{4\times(2^3+1)} + \underbrace{T(n-2) + T(n-4) + T(n-5) + \ldots}_{2^3 \text{ function calls}}$
- $T(n) = 4(2^n + 1) + T(n-n)2^n$, by induction
- $T(n) = (4+3)2^n + 4$, by replacing $T(0)$ by 3

Recursive Fibonacci series (3)

```
SubProceso result <- Fibo( n )
  Definir result Como Entero;  //1
  Si n <= 1 Entonces           //1
    result <- n;               //1
  Sino
    result<-Fibo(n-1)+Fibo(n-2); //2+T(n-1)+T(n-2)
```

$$T(n) = \begin{cases} 3 & if \quad n < 1 \\ 4 + T(n-1) + T(n-2) & if \quad n > 1 \end{cases} = (4+3)2^n + 4$$

is $O(2^n)$

Recursive Fibonacci series : Proof (2)

1. $T(n) = (3+4)2^n + 4$
2. $(3+4)2^n + 4$ is $O((3+4)2^n + 4)$, by Definition of O
3. $O((3+4)2^n + 4) = O((3+4)2^n)$, by Rule of Sums
4. $O((3+4)2^n) = O(2^n)$, by Rule of Products
5. Therefore, $T(n) = (3+4)2^n + 4$ is $O(2^n)$.

Recursive Fibonacci series : Proof (2)

1. $T(n) = (3+4)2^n + 4$
2. $(3+4)2^n + 4$ is $O((3+4)2^n + 4)$, by Definition of O
3. $O((3+4)2^n + 4) = O((3+4)2^n)$, by Rule of Sums
4. $O((3+4)2^n) = O(2^n)$, by Rule of Products
5. Therefore, $T(n) = (3+4)2^n + 4$ is $O(2^n)$.

Recursive Fibonacci series : Proof (2)

1. $T(n) = (3+4)2^n + 4$
2. $(3+4)2^n + 4$ is $O((3+4)2^n + 4)$, by Definition of O
3. $O((3+4)2^n + 4) = O((3+4)2^n)$, by Rule of Sums
4. $O((3+4)2^n) = O(2^n)$, by Rule of Products
5. Therefore, $T(n) = (3+4)2^n + 4$ is $O(2^n)$.

Recursive Fibonacci series : Proof (2)

1. $T(n) = (3+4)2^n + 4$
2. $(3+4)2^n + 4$ is $O((3+4)2^n + 4)$, by Definition of O
3. $O((3+4)2^n + 4) = O((3+4)2^n)$, by Rule of Sums
4. $O((3+4)2^n) = O(2^n)$, by Rule of Products
5. Therefore, $T(n) = (3+4)2^n + 4$ is $O(2^n)$.

Recursive Fibonacci series : Proof (2)

1. $T(n) = (3+4)2^n + 4$
2. $(3+4)2^n + 4$ is $O((3+4)2^n + 4)$, by Definition of O
3. $O((3+4)2^n + 4) = O((3+4)2^n)$, by Rule of Sums
4. $O((3+4)2^n) = O(2^n)$, by Rule of Products
5. Therefore, $T(n) = (3+4)2^n + 4$ is $O(2^n)$.

Case 1: $T(n) = T(n-a) + C$

- $T(n) = T(n-1) + C$
- $T(n) = T(n-3) + C$
- Example: Recursion 1, factorial, array sum
- $T(n)$ is $O(n)$

Case 1: $T(n) = T(n-a) + C$

- $T(n) = T(n-1) + C$
- $T(n) = T(n-3) + C$
- Example: Recursion 1, factorial, array sum
- $T(n)$ is $O(n)$

Case 2: $T(n) = T(n-a) + T(n-b)$

- $T(n) = T(n-1) + T(n-2)$
- $T(n) = 2T(n-1)$
- Example: Recursion 2, Fibonacci, Hannoi Towers
- $T(n)$ is $O(2^n)$

Case 2: $T(n) = T(n-a) + T(n-b)$

- $T(n) = T(n-1) + T(n-2)$
- $T(n) = 2T(n-1)$
- Example: Recursion 2, Fibonacci, Hannoi Towers
- $T(n)$ is $O(2^n)$

Case 2: $T(n) = T(n-a) + T(n-b)$

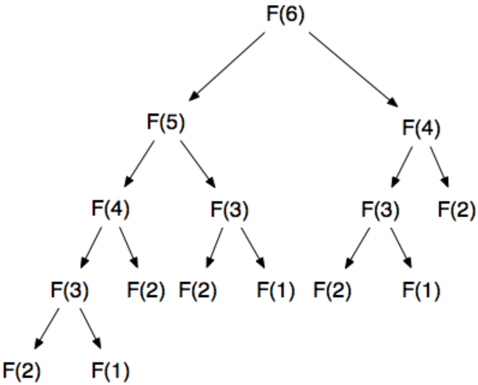


Figure: Execution of a case-2 algorithm

Case 3: $T(n) = kT(n-a)$

- $T(n) = \underbrace{T(n-a) + T(n-b) + \cdots + T(n-c)}_{k \text{ times}}$
- Example: Minimax
- $T(n)$ is $O(k^n)$

Case 3: $T(n) = kT(n-a)$

- $T(n) = \underbrace{T(n-a) + T(n-b) + \cdots + T(n-c)}_{k \text{ times}}$
- Example: Minimax
- $T(n)$ is $O(k^n)$

Case 3: $T(n) = kT(n-a)$

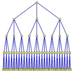


Figure: Execution of a case-3 algorithm for $k = 3$

Quiz questions

- Compute recursively the sum of the elements of an array is $O(n)$
- Compute recursively the maximum element of an array is $O(n)$
- Compute recursively the Fibonnaci series is $O(2^n)$
- Homogeneous lineal recurrence equations can be solved by induction

References

- Please check the slides after class to learn how to reference images, trademarks, videos and fragments of code.
- Avoid plagiarism

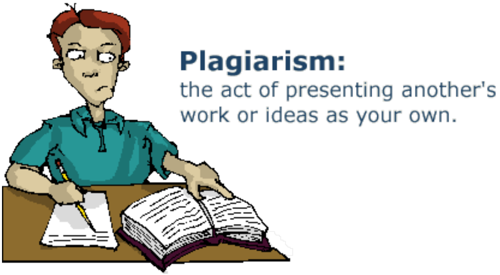


Figure: Figure about plagiarism, University of Malta [Uni09]

University of Malta.

Plagarism — The act of presenting another's work or ideas as your own, 2009.

[Online; accessed 29-November-2013].

Further reading

- Complexity of algorithms
  - Brassard y Bratley, Fundamentos de Algoritmia. Capítulo 3: Notación asintótica. Páginas 99 a 106.