Data Structures II: Nonlinear rec. equations

Mauricio Toro
Department of Systems and Informatics
Universidad EAFIT

Cocktail of the day: Alexander

Disclaimer: Keep alcohol out of the hands of minors.

Cocktail of the day: Alexander

- 30 ml Cognac
- 30 ml Crème de Cacao
- 30 ml Fresh cream

Artificial vision uses graphs

https://www.youtube.com/watch?v=EzjkBwZtxp4

Review: O notation

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

https://www.khanacademy.org/computing/computer-science/cryptography/modern-crypt/p/time-complexity-exploration

O cheat sheet

http://http://bigocheatsheet.com/

Recursive sum of an array

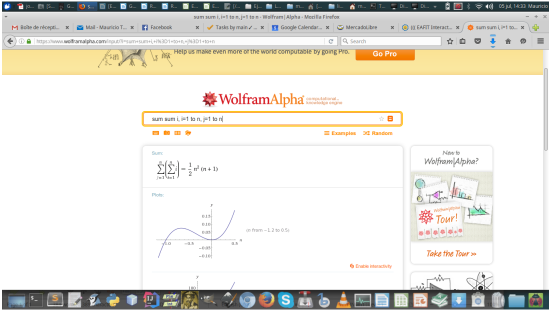
```
SubProceso sum <- ArraySum( A, n )
  Definir i, sum Como Entero;
  Si n = A.length Entonces
    sum <- A[n];
  Sino
    sum <- A[n] + ArraySum(A, n+1);
  FinSi
FinSubProceso
```

$T(n) = ?$

UNIVERSIDAD EAFIT

How you need help with sums?

Use this tool



https://www.wolframalpha.com/

Recursive sum of an array (2)

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

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

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

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

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

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

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

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

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

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 $O(n)$.

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 $O(n)$

UNIVERSIDAD EAFIT®

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

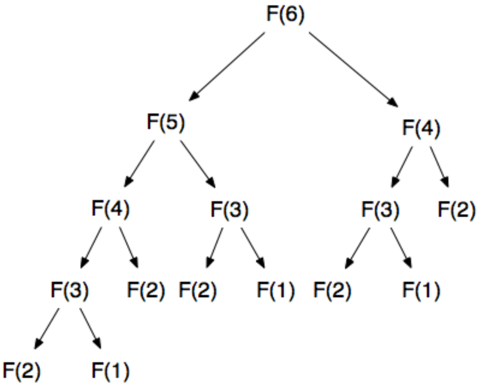


Figure: Execution of a case-2 algorithm

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

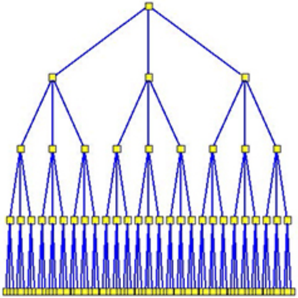
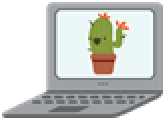


Figure: Execution of a case-3 algorithm for $k = 3$

Recursive binary search



https://www.youtube.com/watch?x-yt-ts=1421914688&x-yt-cl=84503534&feature=player_embedded&v=iDVH3oCTc2c#t=0

Recursive binary search

```
SubProceso index <- BinarySearch( A, k, l, h)
  Definir index, m Como Entero;
  m <- l+(h-l)/2;
  Si (h-l) <= 0 Entonces index <- -1;
  Sino Si A[m] = k Entonces index <- m;
  Sino Si A[m] > k Entonces
      index <- BinarySearch(A, k, m+1, h);
  Sino
      index <- BinarySearch(A, k, l, m-1);
```

$T(n) = ?$

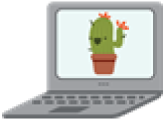
Recursive binary search (2)

```
SubProceso index <- BinarySearch( A, k, l, h)
  Definir index, m Como Entero;              //C1
  m <- l+(h-l)/2;                            //C2
  Si (h-l) <= 0 Entonces index <- -1;        //C3
  Sino
    Si A[m] = k Entonces index <- m;         //C4
    Sino Si A[m] > k Entonces                //C5
      index <-BinarySearch(A, k, m+1, h);//C6 + T(n/2)
    Sino
      index <-BinarySearch(A, k, l, m-1);//C6 + T(n/2)
```

$T(n) = C + T(n/2)$ if $n > 1$

Master theorem

In what follows they describe the 3 cases:
http://www.csanimated.com/animation.php?t=Master_theorem



Master theorem, case II

- Given $T(n) = aT(\frac{n}{b}) + f(n)$, where $a \geq 1, b > 1$.
- If it is true, for some constant $k \geq 0$, that:
    - $f(n)$ is $\Theta(n^c log^k n)$, where $c = log_b a$
- Then
    - $T(n)$ is $\Theta(n^c log^{k+1} n)$
- Therefore, it is also true that
    - $T(n)$ is $O(n^c log^{k+1} n)$

Master theorem, case II

- Given $T(n) = aT(\frac{n}{b}) + f(n)$, where $a \geq 1, b > 1$.
- If it is true, for some constant $k \geq 0$, that:
  - $f(n)$ is $\Theta(n^c log^k n)$, where $c = log_b a$
- Then
  - $T(n)$ is $\Theta(n^c log^{k+1} n)$
- Therefore, it is also true that
  - $T(n)$ is $O(n^c log^{k+1} n)$

Master theorem, case II

- Given $T(n) = aT(\frac{n}{b}) + f(n)$, where $a \geq 1, b > 1$.
- If it is true, for some constant $k \geq 0$, that:
  - $f(n)$ is $\Theta(n^c log^k n)$, where $c = log_b a$
- Then
  - $T(n)$ is $\Theta(n^c log^{k+1} n)$
- Therefore, it is also true that
  - $T(n)$ is $O(n^c log^{k+1} n)$

Recursive binary search: Proof

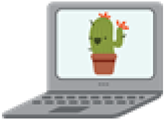
- $T(n) = C + T(n/2)$ has the form $aT(\frac{n}{b}) + f(n)$
- where $a = 1, b = 2$,
- $f(n)$ is $\Theta(n^c log^k n)$, where $c = 0, k = 0$
- Therefore, by Master theorem
- $T(n)$ is $\Theta(n^0 log^{0+1} n)$, and
- $T(n)$ is $O(log\ n)$, by Definition of Θ

Recursive binary search: Proof

- $T(n) = C + T(n/2)$ has the form $aT(\frac{n}{b}) + f(n)$
- where $a = 1, b = 2,$
- $f(n)$ is $\Theta(n^c log^k n)$, where $c = 0, k = 0$
- Therefore, by Master theorem
- $T(n)$ is $\Theta(n^0 log^{0+1} n)$, and
- $T(n)$ is $O(log\ n)$, by Definition of Θ

Recursive Merge Sort

https://www.youtube.com/watch?v=ZRPoEKHXTJg



Recursive Merge Sort

```
SubProceso MergeSort( A, l, h)
  Definir m Como Entero;
  m <- l+(h-l)/2;
  Si (h-l) > 1 Entonces
    MergeSort(A,l,m-1);
    MergeSort(A,m,h);
    Merge(A, l,m-1, m, h);
  FinSi
FinSubProceso
```

$T(n) = ?$

Recursive Merge Sort (2)

```
SubProceso MergeSort( A, l, h)
  Definir m Como Entero;    //O(1)
  m <- l+(h-l)/2;           //O(1)
  Si (h-l) > 1 Entonces     //O(1)
    MergeSort(A,l,m-1);     //T(n/2)
    MergeSort(A,m,h);       //T(n/2)
    Merge(A, l,m-1, m, h); //O(n)
  FinSi
FinSubProceso
```

$T(n) = O(n) + 2T(n/2)$ if $n > 1$ ...why?

Recursive merge sort: Proof

- $T(n) = O(n) + 2T(n/2)$ has the form $aT(\frac{n}{b}) + f(n)$
- where $a = 2, b = 2,$
- $f(n)$ is $\Theta(n^c log^k n)$, where $c = 1, k = 0$
- Therefore, by Master theorem
- $T(n)$ is $\Theta(n^1 log^{0+1} n)$, and
- $T(n)$ is $O(n.log(n))$, by Definition of Θ

Recursive merge sort: Proof

- $T(n) = O(n) + 2T(n/2)$ has the form $aT(\frac{n}{b}) + f(n)$
- where $a = 2, b = 2,$
- $f(n)$ is $\Theta(n^c log^k n)$, where $c = 1, k = 0$
- Therefore, by Master theorem
- $T(n)$ is $\Theta(n^1 log^{0+1} n)$, and
- $T(n)$ is $O(n.log(n))$, by Definition of Θ

Quiz questions

- Binary search has a complexity of $O(log(n))$
- Merge sort has a complexity of $O(n.log(n))$
- Some nonlinear recurrence equations can be solved with the Master theorem

References

- Please learn how to reference images, trademarks, videos and fragments of code.
- Avoid plagiarism

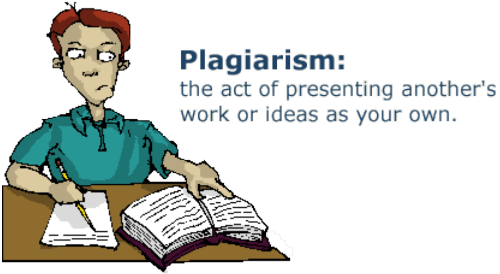


Figure: Figure about plagiarism, University of Malta [Uni09]

References

University of Malta.

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

[Online; accessed 29-November-2013].

Further Reading

- Directed graphs
  - Alfred Aho, Estructuras de Datos y Algoritmos. Capítulo 6: Grafos dirigidos. Páginas 267 - 276.