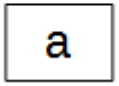
Cota Inferior – exemplos de árvores de decisão

Encontrar o mínimo em um arranjo de n elementos

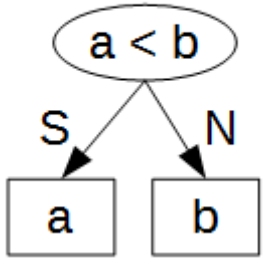
n=1

<a>

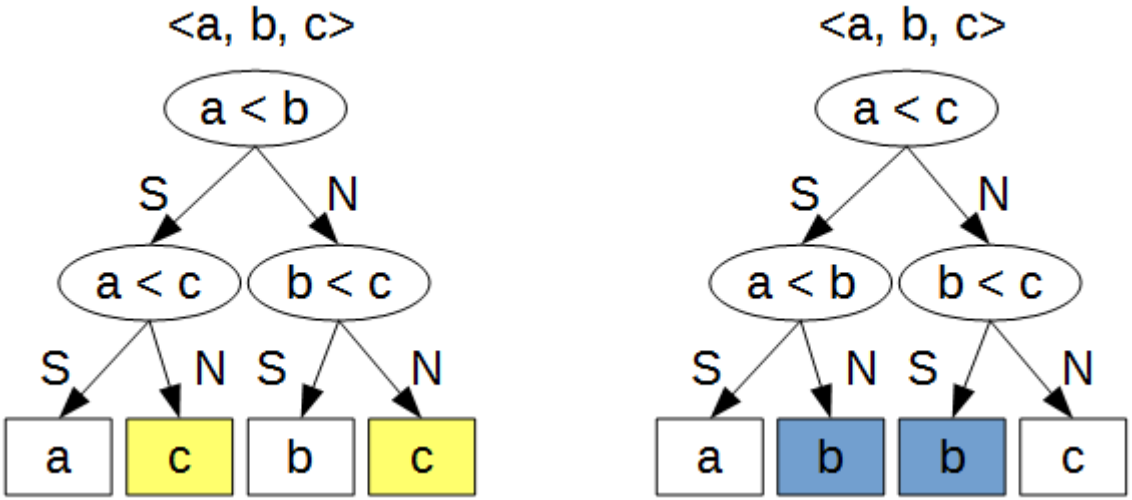


n=2

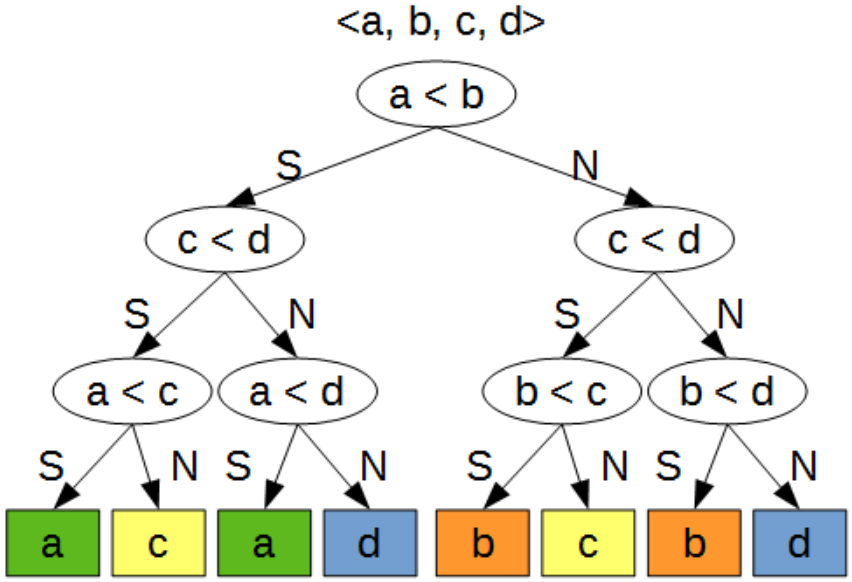
<a, b>



n=3 (exemplos)



n=4



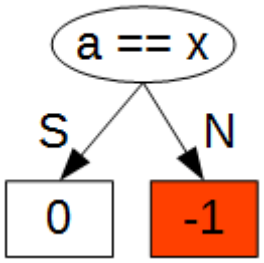
Encontrar o valor mínimo	
Elementos	**Altura mínima da árvore**
1	0
2	1
3	2
4	3
...	...
n	n-1

Cota inferior: encontrar o mínimo em um arranjo $\epsilon \Omega(n)$

Busca sequencial em um arranjo de n elementos

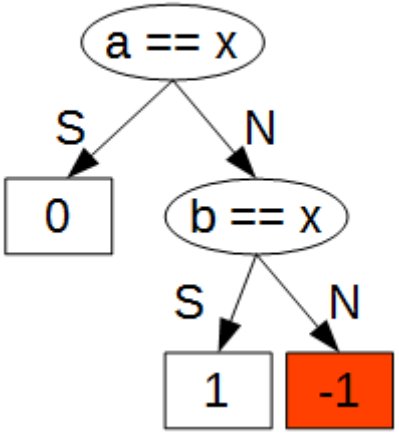
n=1

<a> [x]

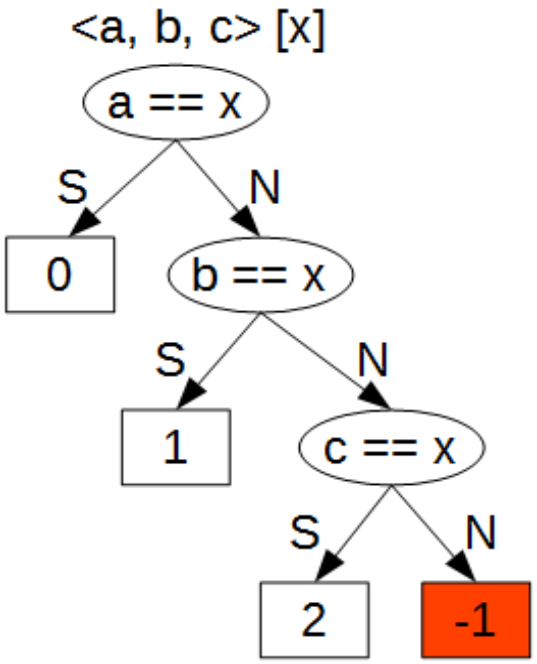


n=2

<a, b> [x]



n=3



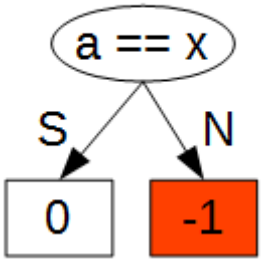
Encontrar um elemento em um arranjo (busca sequencial)	
N	**Altura mínima da árvore**
1	1
2	2
3	3
4	4
...	...
n	n

Cota inferior: encontrar um elemento em um arranjo não ordenado/organizado: $\epsilon \Omega(n)$

Busca binária em um arranjo de n elementos

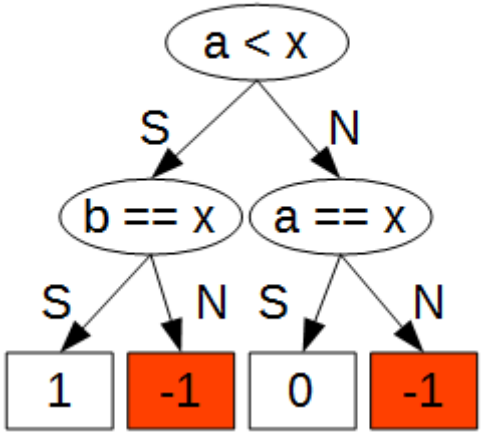
n=1

<a> [x]

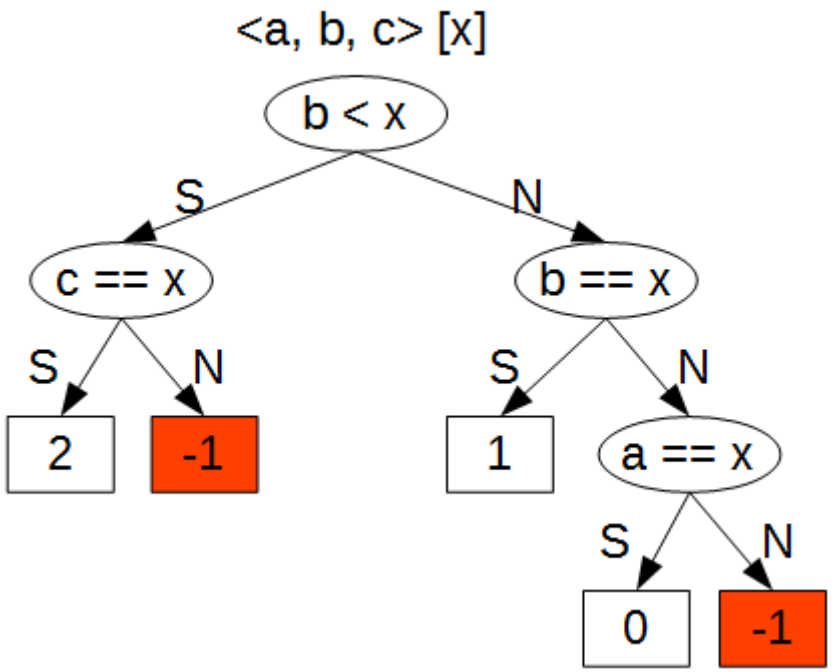


n=2

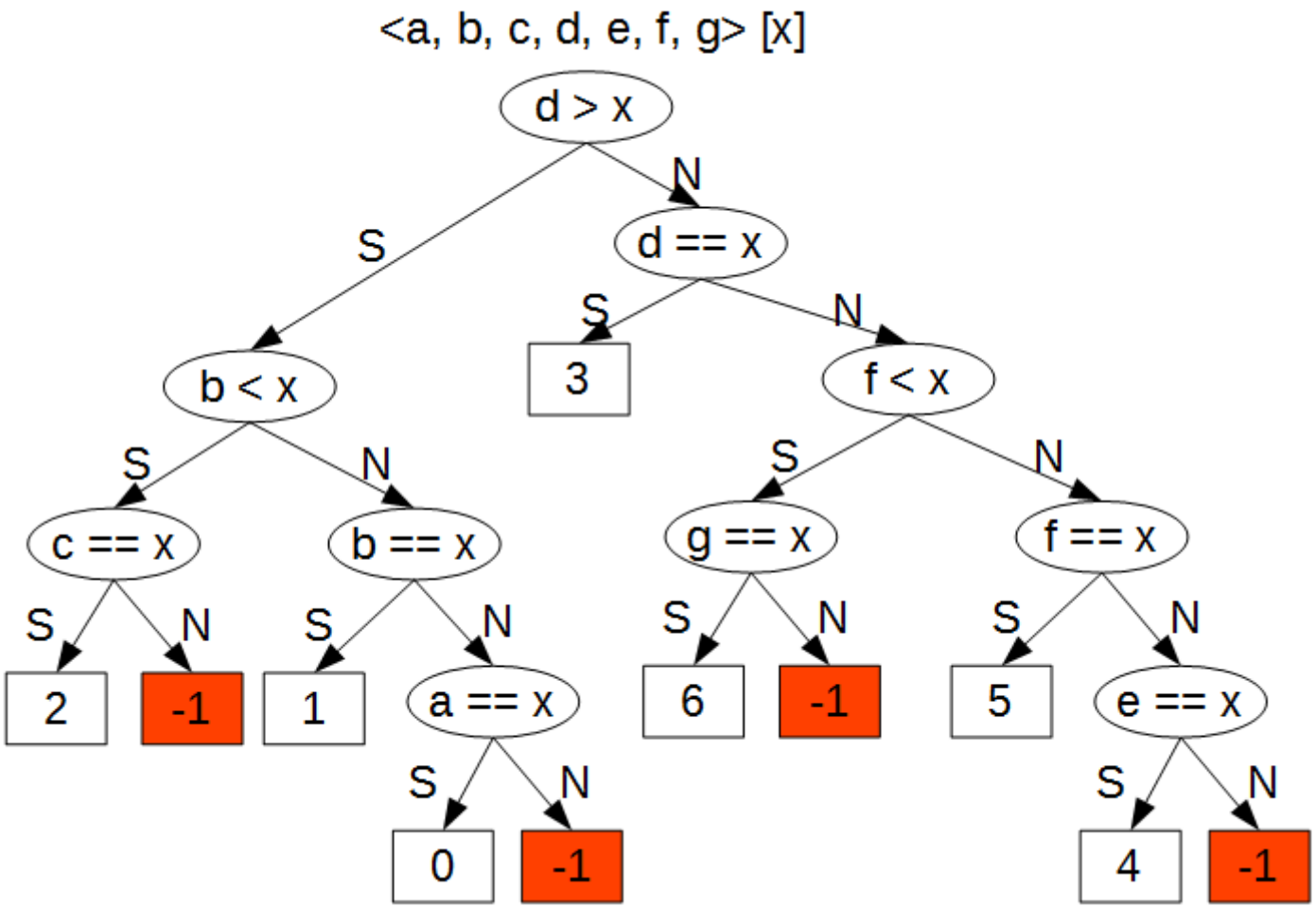
<a, b> [x]



n=3



n=7


<a, b, c, d, e, f, g> [x]
d > x
S
N
d == x
S
N
3
b < x
f < x
S
N
S
N
c == x
b == x
g == x
f == x
S
N
S
N
S
N
S
N
2
-1
1
a == x
6
-1
5
e == x
S
N
S
N
0
-1
4
-1

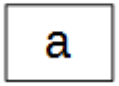
Encontrar um elemento em um arranjo (busca binária)	
N	**Altura mínima da árvore**
1	1
3	3
7	5
15	7
31	9
...	...
n (igual a 2^y-1)	$2*\lfloor log_2 \, n \rfloor+1$

Cota inferior: encontrar um elemento em um arranjo ordenado, por comparações: ϵ Ω(log n)

Ordenação por comparação em um arranjo de n elementos

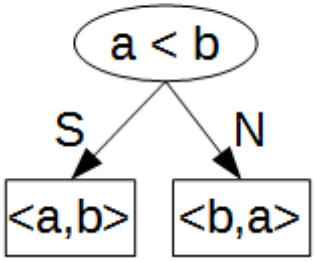
n=1

<a>

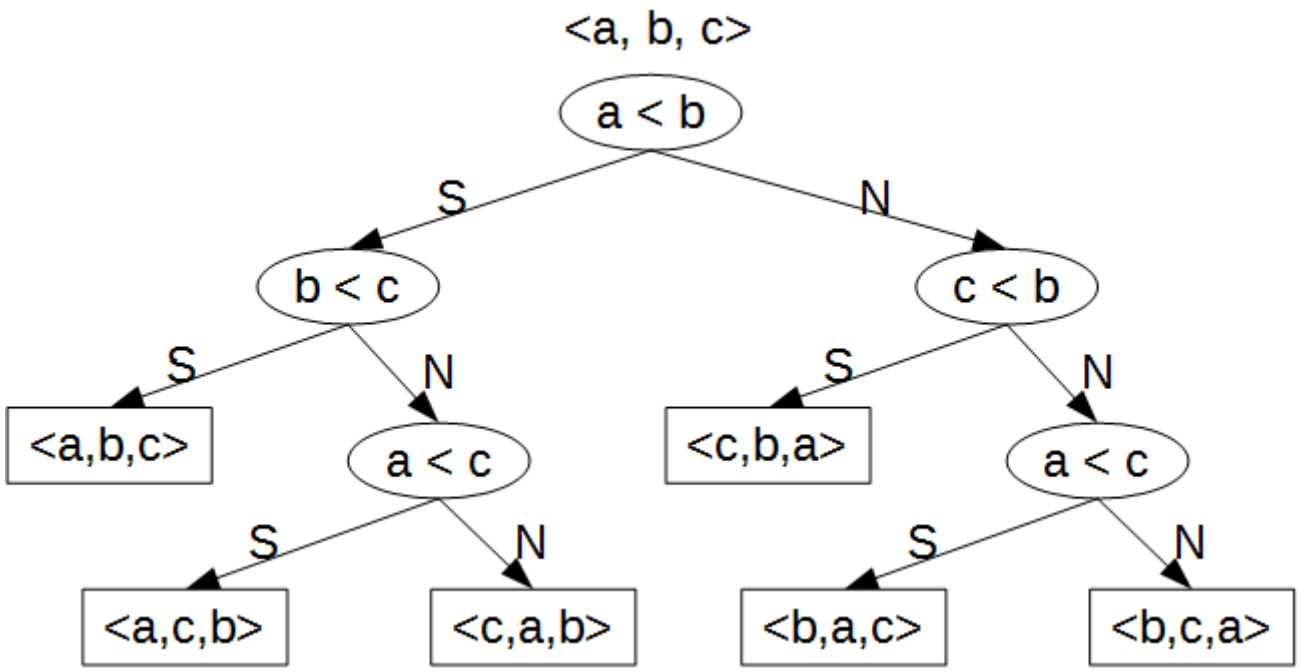


n=2

<a, b>



n=3



Ordenar um conjunto por comparação	
N	**Altura mínima da árvore**
1	**0**
2	**1**
3	**3**
…	**…**
?	**?**

A identificação das menores árvores de decisão para cada problema e a fórmula que generaliza a altura da árvore em relação ao número de elementos é uma **tarefa difícil**.

Por isso não utilizaremos esta técnica e sim uma simplificação derivada dela: sabemos que cada resposta possível produzirá, **no mínimo, uma folha** na árvore binária. Também sabemos que **uma árvore binária com altura h terá no máximo 2^h folhas**.

Imaginaremos uma **árvore binária de decisão completa**, com apenas uma folha para cada resposta possível (sem repetições de respostas) e estabeleceremos um limite inferior para o problema. Esta abordagem pode ser menos precisa, mas também correta, do que o uso das árvores de decisão.

Por exemplo, se um problema tem **n respostas diferentes** e a altura de uma árvore binária de decisão com n folhas **(n respostas)** será de no mínimo $\lceil log_2 n \rceil$ podemos concluir que este problema tem sua cota inferior em $\lceil log_2 n \rceil$, isto é, a complexidade de qualquer algoritmo que resolva este problema **ϵ Ω(log n)** (talvez não seja possível resolver o problema com essa complexidade, mas com certeza não é possível resolver com complexidade menor).

Encontrar o mínimo em um arranjo de n elementos

Soluções possíveis: **n**

Cota inferior por esta estratégia: **Ω(log n)**

Busca sequencial em um arranjo de n elementos

Soluções possíveis: **n + 1**

Cota inferior por esta estratégia: **Ω(log n)**

Busca binária em um arranjo de n elementos

Soluções possíveis: **n + 1**

Cota inferior por esta estratégia: **Ω(log n)**

Ordenação por comparação em um arranjo de n elementos

Soluções possíveis: **n!** (todas as permutações de n elementos)

Cota inferior por esta estratégia: **Ω(log n!)**

Mas quanto vale $\log_2 n!$?

$\log_2 n! = \sum_{i=1}^{n} \log_2 i$

$\log_2 n! \geq \sum_{i=\lceil \frac{n}{2} \rceil}^{n} \log_2 i$

$\log_2 n! \geq \sum_{i=\lceil \frac{n}{2} \rceil}^{n} \log_2 \frac{n}{2}$

$\log_2 n! \geq \frac{n}{2} * \log_2 \frac{n}{2}$

$\log_2 n! \geq \frac{n}{2} * (\log_2 n - \log_2 2)$

$\log_2 n! \geq \frac{n}{2} * \log_2 n - \frac{n}{2}$ para n ≥ 2 temos:

$\log_2 n! \geq \frac{n}{2} * \log_2 n$

Sendo $\log_2 n! \geq \frac{n}{2} * \log_2 n$ sabemos que $\log_2 n! \in \Omega(n*\log n)$

Consequentemente, a cota inferior para a ordenação por comparação por esta estratégia é: **Ω(n log n)**

Portanto, os algoritmos ***Heap Sort*** e ***Merge Sort*** são **ótimos** em termos de ordenação por comparação, pois, no pior caso, $\in \Theta(n*\log n)$ e não é possível ordenar por comparações com complexidade inferior a n*log n:

Problema $\in \Omega(n*\log n)$

Algoritmos $\in O(n*\log n)$ -> algoritmos ótimos