MC458 — Projeto e Análise de Algoritmos I

C.C. de Souza   C.N. da Silva   O. Lee

Antes de mais nada...

- Uma versão anterior deste conjunto de slides foi preparada por Cid Carvalho de Souza e Cândida Nunes da Silva para uma instância anterior desta disciplina.
- O que vocês tem em mãos é uma versão modificada preparada para atender a meus gostos.
- Nunca é demais enfatizar que o material é apenas um guia e não deve ser usado como única fonte de estudo. Para isso consultem a bibliografia (em especial o CLR ou CLRS).

Orlando Lee

Agradecimentos (Cid e Cândida)

- Várias pessoas contribuíram direta ou indiretamente com a preparação deste material.
- Algumas destas pessoas cederam gentilmente seus arquivos digitais enquanto outras cederam gentilmente o seu tempo fazendo correções e dando sugestões.
- Uma lista destes "colaboradores" (em ordem alfabética) é dada abaixo:
  - Célia Picinin de Mello
  - José Coelho de Pina
  - Orlando Lee
  - Paulo Feofiloff
  - Pedro Rezende
  - Ricardo Dahab
  - Zanoni Dias

Divisão e Conquista

Projeto de Algoritmos por Divisão e Conquista

Projeto de Algoritmos por Divisão e Conquista

- **Dividir para conquistar:** uma tática de guerra aplicada ao projeto de algoritmos.

Projeto de Algoritmos por Divisão e Conquista

- **Dividir para conquistar:** uma tática de guerra aplicada ao projeto de algoritmos.
- Um algoritmo de divisão e conquista é aquele que resolve o problema desejado combinando as soluções parciais de (um ou mais) subproblemas, obtidas recursivamente.

Projeto de Algoritmos por Divisão e Conquista

- **Dividir para conquistar:** uma tática de guerra aplicada ao projeto de algoritmos.
- Um algoritmo de divisão e conquista é aquele que resolve o problema desejado combinando as soluções parciais de (um ou mais) subproblemas, obtidas recursivamente.
- É mais um paradigma de projeto de algoritmos baseado no princípio da indução.

- **Dividir para conquistar:** uma tática de guerra aplicada ao projeto de algoritmos.
- Um algoritmo de divisão e conquista é aquele que resolve o problema desejado combinando as soluções parciais de (um ou mais) subproblemas, obtidas recursivamente.
- É mais um paradigma de projeto de algoritmos baseado no princípio da indução.
- Informalmente, podemos dizer que o paradigma incremental representa o projeto de algoritmos por indução fraca, enquanto o paradigma de divisão e conquista representa o projeto por indução forte.

Projeto de Algoritmos por Divisão e Conquista

- **Dividir para conquistar:** uma tática de guerra aplicada ao projeto de algoritmos.
- Um algoritmo de divisão e conquista é aquele que resolve o problema desejado combinando as soluções parciais de (um ou mais) subproblemas, obtidas recursivamente.
- É mais um paradigma de projeto de algoritmos baseado no princípio da indução.
- Informalmente, podemos dizer que o paradigma incremental representa o projeto de algoritmos por indução fraca, enquanto o paradigma de divisão e conquista representa o projeto por indução forte.
- É natural, portanto, demonstrar a corretude de algoritmos de divisão e conquista por indução.

Algoritmo Genérico

DIVISÃO-E-CONQUISTA(x)

▷ **Entrada:** A instância x

▷ **Saída:** Solução y do problema em questão para x

1. **se** x é suficientemente pequeno
2. **então devolva** $Solucao(x)$
3. **senão**
4. decomponha x em instâncias menores $x_1, x_2, \cdots, x_k$
5. **para** i **de** 1 **até** k **faça**
6. $y_i \leftarrow$ DIVISÃO-E-CONQUISTA(x_i)
7. combine as soluções y_i para obter a solução y de x.
8. **devolva** y

Projeto por Divisão e Conquista - Exemplo 1
Exponenciação

Problema:

Calcular a^n, para todo real a e inteiro $n \geq 0$.

Projeto por Divisão e Conquista - Exemplo 1
Exponenciação

Problema:

Calcular a^n, para todo real a e inteiro $n \geq 0$.

Primeira solução, por indução fraca:

Projeto por Divisão e Conquista - Exemplo 1
Exponenciação

Problema:

Calcular a^n, para todo real a e inteiro $n \geq 0$.

Primeira solução, por indução fraca:

- **Caso base:** $n = 0$; $a^0 = 1$.

Projeto por Divisão e Conquista - Exemplo 1
Exponenciação

Problema:

Calcular a^n, para todo real a e inteiro $n \geq 0$.

Primeira solução, por indução fraca:

- **Caso base:** $n = 0$; $a^0 = 1$.
- **Hipótese de indução:** *Suponha que, para qualquer inteiro* $n > 0$ *e real* a, *sei calcular* a^{n-1}.

Problema:

Calcular a^n, para todo real a e inteiro $n \geq 0$.

Primeira solução, por indução fraca:

- **Caso base:** $n = 0$; $a^0 = 1$.
- **Hipótese de indução:** *Suponha que, para qualquer inteiro* $n > 0$ *e real* a*, sei calcular* a^{n-1}.
- **Passo da indução:** Queremos agora calcular a^n, para $n > 0$. Por hipótese de indução, sei calcular a^{n-1}. Então, calculamos a^n multiplicando a^{n-1} por a.

Exemplo 1 - Solução 1 - Algoritmo

▷ **Entrada:** Um número real a e um inteiro $n \geq 0$.
▷ **Saída:** O valor de a^n.

Exponenciação(a, n)

1. **se** $n = 0$
2. **então** $an \leftarrow 1$ ▷ caso base
3. **senão**
4. $an' \leftarrow$ Exponenciação$(a, n-1)$
5. $an \leftarrow an' * a$
6. **devolva** an

Exemplo 1 - Solução 1 - Complexidade

Exemplo 1 - Solução 1 - Complexidade

Seja $T(n)$ o número de operações executadas pelo algoritmo para calcular a^n.

Seja $T(n)$ o número de operações executadas pelo algoritmo para calcular a^n.

Então a relação de recorrência deste algoritmo é:

$$T(n) = \begin{cases} \Theta(1), & n = 0 \\ T(n-1) + \Theta(1), & n > 0. \end{cases}$$

Exemplo 1 - Solução 1 - Complexidade

Seja $T(n)$ o número de operações executadas pelo algoritmo para calcular a^n.

Então a relação de recorrência deste algoritmo é:

$$T(n) = \begin{cases} \Theta(1), & n = 0 \\ T(n-1) + \Theta(1), & n > 0. \end{cases}$$

Neste caso, não é difícil ver que

$$T(n) = \Theta(n).$$

Exemplo 1 - Solução 1 - Complexidade

Seja $T(n)$ o número de operações executadas pelo algoritmo para calcular a^n.

Então a relação de recorrência deste algoritmo é:

$$T(n) = \begin{cases} \Theta(1), & n = 0 \\ T(n-1) + \Theta(1), & n > 0. \end{cases}$$

Neste caso, não é difícil ver que

$$T(n) = \Theta(n).$$

Este algoritmo é linear no tamanho da entrada?

Exemplo 1 - Solução 1 - Complexidade

Seja $T(n)$ o número de operações executadas pelo algoritmo para calcular a^n.

Então a relação de recorrência deste algoritmo é:

$$T(n) = \begin{cases} \Theta(1), & n = 0 \\ T(n-1) + \Theta(1), & n > 0. \end{cases}$$

Neste caso, não é difícil ver que

$$T(n) = \Theta(n).$$

Este algoritmo é linear no tamanho da entrada?
Não! O tamanho da entrada é $\lg a + \lg n$.

Exemplo 1 - Solução 2 - Divisão e Conquista

Agora projetaremos um algoritmo para o problema usando indução forte (divisão e conquista).

Exemplo 1 - Solução 2 - Divisão e Conquista

Agora projetaremos um algoritmo para o problema usando indução forte (divisão e conquista).

Segunda solução, por indução forte:

Exemplo 1 - Solução 2 - Divisão e Conquista

Agora projetaremos um algoritmo para o problema usando indução forte (divisão e conquista).

Segunda solução, por indução forte:

- **Caso base:** $n = 0$; $a^0 = 1$.

Exemplo 1 - Solução 2 - Divisão e Conquista

Agora projetaremos um algoritmo para o problema usando indução forte (divisão e conquista).

Segunda solução, por indução forte:

- **Caso base:** $n = 0$; $a^0 = 1$.
- **Hipótese de indução:** *Suponha que, para qualquer inteiro* $n > 0$ *e real* a, *sei calcular* a^k, *para todo* $k : 0 \leq k < n$.

Agora projetaremos um algoritmo para o problema usando indução forte (divisão e conquista).

Segunda solução, por indução forte:

- **Caso base:** $n = 0$; $a^0 = 1$.
- **Hipótese de indução:** *Suponha que, para qualquer inteiro* $n > 0$ *e real* a, *sei calcular* a^k, *para todo* $k : 0 \leq k < n$.
- **Passo da indução:** Queremos agora calcular a^n, para $n > 0$. Por hipótese de indução sei calcular $a^{\lfloor \frac{n}{2} \rfloor}$. Então, calculamos a^n da seguinte forma:

$$a^n = \begin{cases} \left(a^{\lfloor \frac{n}{2} \rfloor}\right)^2, & \text{se } n \text{ par;} \\ a \cdot \left(a^{\lfloor \frac{n}{2} \rfloor}\right)^2, & \text{se } n \text{ ímpar.} \end{cases}$$

Exemplo 1 - Solução 2 - Algoritmo

$\triangleright$ **Entrada:** Um número real a e um inteiro $n \geq 0$.

$\triangleright$ **Saída:** O valor de a^n.

$\textsc{ExponenciaçãoDC}(a, n)$

se $n = 0$

então $an \leftarrow 1$ $\triangleright$ caso base

senão $\triangleright$ divisão

$an' \leftarrow \textsc{ExponenciaçãoDC}(a, \lfloor \frac{n}{2} \rfloor)$ $\triangleright$ conquista

$an \leftarrow an' * an'$

se $n \bmod 2 = 1$

$an \leftarrow an * a$

devolva an

Exemplo 1 - Solução 2 - Complexidade

Exemplo 1 - Solução 2 - Complexidade

- Seja $T(n)$ o número de operações executadas pelo algoritmo de divisão e conquista para calcular a^n.

Exemplo 1 - Solução 2 - Complexidade

- Seja $T(n)$ o número de operações executadas pelo algoritmo de divisão e conquista para calcular a^n.
- Então a relação de recorrência deste algoritmo é:

$$T(n) = \begin{cases} \Theta(1), & n = 0 \\ T(\lfloor \frac{n}{2} \rfloor) + \Theta(1), & n > 0, \end{cases}$$

Exemplo 1 - Solução 2 - Complexidade

- Seja $T(n)$ o número de operações executadas pelo algoritmo de divisão e conquista para calcular a^n.
- Então a relação de recorrência deste algoritmo é:

$$T(n) = \begin{cases} \Theta(1), & n = 0 \\ T(\lfloor \frac{n}{2} \rfloor) + \Theta(1), & n > 0, \end{cases}$$

- Não é difícil ver que $T(n) = \Theta(\log n)$. Por quê?

Projeto por Divisão e Conquista - Exemplo 2
Busca Binária

Problema:

Dados um vetor ordenado A com n números reais e um real x, determinar a posição $1 \leq i \leq n$ tal que $A[i] = x$, ou que não existe tal i.

Projeto por Divisão e Conquista - Exemplo 2
Busca Binária

Problema:

Dados um vetor ordenado A com n números reais e um real x, determinar a posição $1 \leq i \leq n$ tal que $A[i] = x$, ou que não existe tal i.

- O projeto de um algoritmo para este problema usando indução simples nos leva a um algoritmo de busca linear de complexidade de pior caso $\Theta(n)$. (Como seria a indução?)

Projeto por Divisão e Conquista - Exemplo 2
Busca Binária

Problema:

Dados um vetor ordenado A com n números reais e um real x, determinar a posição $1 \leq i \leq n$ tal que $A[i] = x$, ou que não existe tal i.

- O projeto de um algoritmo para este problema usando indução simples nos leva a um algoritmo de busca linear de complexidade de pior caso $\Theta(n)$. (Como seria a indução?)
- Se utilizarmos indução forte para projetar o algoritmo, obtemos o algoritmo de divisão-e-conquista chamado de busca binária.

Problema:

Sejam $A[p\,..\,r]$ um vetor **ordenado** com $n = r - p + 1$ números reais e x um número real. Queremos projetar um algoritmo que devolve um índice i tal que $p \leq i \leq r$ e $A[i] = x$, ou $i = -1$ se tal índice não existir.

Projeto por Divisão e Conquista - Exemplo 2
Busca Binária

Problema:

Sejam $A[p\,..\,r]$ um vetor **ordenado** com $n = r - p + 1$ números reais e x um número real. Queremos projetar um algoritmo que devolve um índice i tal que $p \leq i \leq r$ e $A[i] = x$, ou $i = -1$ se tal índice não existir.

Solução usando indução forte:

Problema:

Sejam $A[p..r]$ um vetor **ordenado** com $n = r - p + 1$ números reais e x um número real. Queremos projetar um algoritmo que devolve um índice i tal que $p \leq i \leq r$ e $A[i] = x$, ou $i = -1$ se tal índice não existir.

Solução usando indução forte:

- **Hipótese de indução:** Suponha que sabemos resolver o problema para vetores com no máximo $n - 1$ elementos.

Projeto por Divisão e Conquista - Exemplo 2
Busca Binária

Problema:

Sejam $A[p \,..\, r]$ um vetor **ordenado** com $n = r - p + 1$ números reais e x um número real. Queremos projetar um algoritmo que devolve um índice i tal que $p \leq i \leq r$ e $A[i] = x$, ou $i = -1$ se tal índice não existir.

Solução usando indução forte:

- **Hipótese de indução:** Suponha que sabemos resolver o problema para vetores com no máximo $n - 1$ elementos.
- **Caso base:** $n = 1$ (ou $p = r$). Se $A[p] = x$ então devolvemos p, senão devolvemos -1.

Projeto por Divisão e Conquista - Exemplo 2
Busca Binária

Problema:

Sejam $A[p \,..\, r]$ um vetor **ordenado** com $n = r - p + 1$ números reais e x um número real. Queremos projetar um algoritmo que devolve um índice i tal que $p \leq i \leq r$ e $A[i] = x$, ou $i = -1$ se tal índice não existir.

Solução usando indução forte:

- **Hipótese de indução:** Suponha que sabemos resolver o problema para vetores com no máximo $n - 1$ elementos.

Projeto por Divisão e Conquista - Exemplo 2
Busca Binária

Problema:

Sejam $A[p..r]$ um vetor **ordenado** com $n = r - p + 1$ números reais e x um número real. Queremos projetar um algoritmo que devolve um índice i tal que $p \leq i \leq r$ e $A[i] = x$, ou $i = -1$ se tal índice não existir.

Solução usando indução forte:

- **Hipótese de indução:** Suponha que sabemos resolver o problema para vetores com no máximo $n - 1$ elementos.
- **Passo de indução:** Queremos resolver o problema para $A[p..r]$ e x. Seja $i = \lceil (p + r)/2 \rceil$.

Projeto por Divisão e Conquista - Exemplo 2
Busca Binária

Problema:

Sejam $A[p \mathinner{.\,.} r]$ um vetor **ordenado** com $n = r - p + 1$ números reais e x um número real. Queremos projetar um algoritmo que devolve um índice i tal que $p \leq i \leq r$ e $A[i] = x$, ou $i = -1$ se tal índice não existir.

Solução usando indução forte:

- **Hipótese de indução:** Suponha que sabemos resolver o problema para vetores com no máximo $n - 1$ elementos.
- **Passo de indução:** Queremos resolver o problema para $A[p \mathinner{.\,.} r]$ e x. Seja $i = \lceil (p + r)/2 \rceil$.

  Se $x < A[i]$ então x só pode estar em $A[p \mathinner{.\,.} i - 1]$.

Projeto por Divisão e Conquista - Exemplo 2
Busca Binária

Problema:

Sejam $A[p \mathinner{..} r]$ um vetor **ordenado** com $n = r - p + 1$ números reais e x um número real. Queremos projetar um algoritmo que devolve um índice i tal que $p \leq i \leq r$ e $A[i] = x$, ou $i = -1$ se tal índice não existir.

Solução usando indução forte:

- **Hipótese de indução:** Suponha que sabemos resolver o problema para vetores com no máximo $n - 1$ elementos.
- **Passo de indução:** Queremos resolver o problema para $A[p \mathinner{..} r]$ e x.
  Seja $i = \lceil (p + r)/2 \rceil$.

  Se $x < A[i]$ então x só pode estar em $A[p \mathinner{..} i - 1]$.

  Aplicamos a HI a $A[p \mathinner{..}, i - 1]$ e x.

Projeto por Divisão e Conquista - Exemplo 2
Busca Binária

Problema:

Sejam $A[p\,..\,r]$ um vetor **ordenado** com $n = r - p + 1$ números reais e x um número real. Queremos projetar um algoritmo que devolve um índice i tal que $p \leq i \leq r$ e $A[i] = x$, ou $i = -1$ se tal índice não existir.

Solução usando indução forte:

- **Hipótese de indução:** Suponha que sabemos resolver o problema para vetores com no máximo $n - 1$ elementos.
- **Passo de indução:** Queremos resolver o problema para $A[p\,..\,r]$ e x. Seja $i = \lceil (p + r)/2 \rceil$.

Projeto por Divisão e Conquista - Exemplo 2
Busca Binária

Problema:

Sejam $A[p \,..\, r]$ um vetor **ordenado** com $n = r - p + 1$ números reais e x um número real. Queremos projetar um algoritmo que devolve um índice i tal que $p \leq i \leq r$ e $A[i] = x$, ou $i = -1$ se tal índice não existir.

Solução usando indução forte:

- **Hipótese de indução:** Suponha que sabemos resolver o problema para vetores com no máximo $n - 1$ elementos.
- **Passo de indução:** Queremos resolver o problema para $A[p \,..\, r]$ e x. Seja $i = \lceil (p + r)/2 \rceil$.

  Se $x \geq A[i]$ então x só pode estar em $A[i \,..\, r]$.

Projeto por Divisão e Conquista - Exemplo 2
Busca Binária

Problema:

Sejam $A[p \,..\, r]$ um vetor **ordenado** com $n = r - p + 1$ números reais e x um número real. Queremos projetar um algoritmo que devolve um índice i tal que $p \leq i \leq r$ e $A[i] = x$, ou $i = -1$ se tal índice não existir.

Solução usando indução forte:

- **Hipótese de indução:** Suponha que sabemos resolver o problema para vetores com no máximo $n - 1$ elementos.
- **Passo de indução:** Queremos resolver o problema para $A[p \,..\, r]$ e x. Seja $i = \lceil (p + r)/2 \rceil$.

  Se $x \geq A[i]$ então x só pode estar em $A[i \,..\, r]$.

  Aplicamos a HI a $A[i \,..\, , r]$ e x.

Eis algumas observações:

Eis algumas observações:

- Este algoritmo evita o teste de igualdade até chegar na base.

Eis algumas observações:

- Este algoritmo evita o teste de igualdade até chegar na base.
- Com isso, há apenas uma comparação entre elementos em cada nível da recursão.

Eis algumas observações:

- Este algoritmo evita o teste de igualdade até chegar na base.
- Com isso, há apenas uma comparação entre elementos em cada nível da recursão.
- Por outro lado, não há modo de parar a busca antes.

Projeto por Divisão e Conquista - Exemplo 2
Busca Binária

Eis algumas observações:

- Este algoritmo evita o teste de igualdade até chegar na base.
- Com isso, há apenas uma comparação entre elementos em cada nível da recursão.
- Por outro lado, não há modo de parar a busca antes.
- Outra alternativa, é fazer o teste de igualdade no código o que implica em duas comparações entre elementos por nível de recursão.

Eis algumas observações:

- Este algoritmo evita o teste de igualdade até chegar na base.
- Com isso, há apenas uma comparação entre elementos em cada nível da recursão.
- Por outro lado, não há modo de parar a busca antes.
- Outra alternativa, é fazer o teste de igualdade no código o que implica em duas comparações entre elementos por nível de recursão.
- A escolha entre $i = \lceil (p + r)/2 \rceil$ e $i = \lfloor (p + r)/2 \rfloor$ não afeta a complexidade assintótica do algoritmo.

Eis algumas observações:

- Este algoritmo evita o teste de igualdade até chegar na base.
- Com isso, há apenas uma comparação entre elementos em cada nível da recursão.
- Por outro lado, não há modo de parar a busca antes.
- Outra alternativa, é fazer o teste de igualdade no código o que implica em duas comparações entre elementos por nível de recursão.
- A escolha entre $i = \lceil (p+r)/2 \rceil$ e $i = \lfloor (p+r)/2 \rfloor$ não afeta a complexidade assintótica do algoritmo.
- Entretanto é necessário adaptar a base ($n = 0$ ou $n = 1$) dependendo da escolha de i (chamada recursiva em um vetor vazio).

$\triangleright$ **Entrada:** Um vetor ordenado $A[p\,..\,r]$ não-vazio e um elemento x.

$\triangleright$ **Saída:** Índice $p \leq i \leq r$ tal que $A[i] = x$ ou $i = -1$.

```
BUSCA-BINÁRIA(A, p, r, x)
se p = r  ▷ n = 1
    então
        se A[p] = x então devolva p
                    senão devolva −1
    senão
        i ← ⌈(p + r)/2⌉
        se x < A[i]
            então devolva BUSCA-BINÁRIA(A, p, i − 1, x)
            senão devolva BUSCA-BINÁRIA(A, i, r, x)
```

Exemplo 2 - Complexidade

Exemplo 2 - Complexidade

- O número de operações $T(n)$ executadas na busca binária no pior caso é:

$$T(n) = \begin{cases} \Theta(1), & n = 0 \\ T(\lceil \frac{n}{2} \rceil) + \Theta(1), & n > 1, \end{cases}$$

Exemplo 2 - Complexidade

- O número de operações $T(n)$ executadas na busca binária no pior caso é:

$$T(n) = \begin{cases} \Theta(1), & n = 0 \\ T(\lceil \frac{n}{2} \rceil) + \Theta(1), & n > 1, \end{cases}$$

- A solução é $T(n) = \Theta(\log n)$.

Exemplo 2 - Complexidade

- O número de operações $T(n)$ executadas na busca binária no pior caso é:

$$T(n) = \begin{cases} \Theta(1), & n = 0 \\ T(\lceil \frac{n}{2} \rceil) + \Theta(1), & n > 1, \end{cases}$$

- A solução é $T(n) = \Theta(\log n)$.
- O algoritmo de busca binária (divisão e conquista) tem complexidade de pior caso $\Theta(\log n)$, que é assintoticamente melhor que o algoritmo de busca linear (incremental).

Exemplo 2 - Complexidade

- O número de operações $T(n)$ executadas na busca binária no pior caso é:

$$T(n) = \begin{cases} \Theta(1), & n = 0 \\ T(\lceil \frac{n}{2} \rceil) + \Theta(1), & n > 1, \end{cases}$$

- A solução é $T(n) = \Theta(\log n)$.
- O algoritmo de busca binária (divisão e conquista) tem complexidade de pior caso $\Theta(\log n)$, que é assintoticamente melhor que o algoritmo de busca linear (incremental).
- E se o vetor não estivesse ordenado, qual paradigma nos levaria a um algoritmo assintoticamente melhor?

Aplicações de busca binária

Leiam na seção 6.2 do livro de U. Manber, Algorithms: A Creative Approach, Addison-Wesley (1989), os tópicos:

- Busca binária em uma seqüência ciclicamente ordenada;
- Busca binária para encontrar um índice especial;
- Busca binária em seqüências de tamanhos desconhecidos;
- O problema da subseqüência gaguejante.

Exemplo 3 - Máximo e Mínimo

Problema:

Dado um conjunto S de $n \geq 2$ números reais, determinar o maior e o menor elemento de S.

Exemplo 3 - Máximo e Mínimo

Problema:

Dado um conjunto S de $n \geq 2$ números reais, determinar o maior e o menor elemento de S.

- Um algoritmo incremental simples para esse problema faz $2n - 3$ **comparações entre elementos**. Como?

Exemplo 3 - Máximo e Mínimo

Problema:

Dado um conjunto S de $n \geq 2$ números reais, determinar o maior e o menor elemento de S.

- Um algoritmo incremental simples para esse problema faz $2n - 3$ **comparações entre elementos**. Como?
  Uma comparação inicial entre dois elementos e $2(n - 2)$ comparações para os demais $n - 2$ elementos.

Exemplo 3 - Máximo e Mínimo

Problema:

Dado um conjunto S de $n \geq 2$ números reais, determinar o maior e o menor elemento de S.

- Um algoritmo incremental simples para esse problema faz $2n - 3$ **comparações entre elementos**. Como?
  Uma comparação inicial entre dois elementos e $2(n - 2)$ comparações para os demais $n - 2$ elementos.
- Será que um algoritmo de divisão e conquista seria melhor?

Problema:

Dado um conjunto S de $n \geq 2$ números reais, determinar o maior e o menor elemento de S.

- Um algoritmo incremental simples para esse problema faz $2n - 3$ **comparações entre elementos**. Como?
  Uma comparação inicial entre dois elementos e $2(n-2)$ comparações para os demais $n-2$ elementos.
- Será que um algoritmo de divisão e conquista seria melhor?
- Um possível algoritmo de divisão e conquista seria:

Exemplo 3 - Máximo e Mínimo

Problema:

Dado um conjunto S de $n \geq 2$ números reais, determinar o maior e o menor elemento de S.

- Um algoritmo incremental simples para esse problema faz $2n - 3$ **comparações entre elementos**. Como?
  Uma comparação inicial entre dois elementos e $2(n-2)$ comparações para os demais $n-2$ elementos.
- Será que um algoritmo de divisão e conquista seria melhor?
- Um possível algoritmo de divisão e conquista seria:
  - Divida S em dois subconjuntos de (aproximadamente) mesmo tamanho S_1 e S_2 e solucione os subproblemas.

Exemplo 3 - Máximo e Mínimo

Problema:

Dado um conjunto S de $n \geq 2$ números reais, determinar o maior e o menor elemento de S.

- Um algoritmo incremental simples para esse problema faz $2n - 3$ **comparações entre elementos**. Como?
  Uma comparação inicial entre dois elementos e $2(n-2)$ comparações para os demais $n-2$ elementos.
- Será que um algoritmo de divisão e conquista seria melhor?
- Um possível algoritmo de divisão e conquista seria:
  - Divida S em dois subconjuntos de (aproximadamente) mesmo tamanho S_1 e S_2 e solucione os subproblemas.
  - O máximo de S é o máximo dos máximos de S_1 e S_2 e o mínimo de S é o mínimo dos mínimos de S_1 e S_2.

Exemplo 3 - Algoritmo

▷ **Entrada:** Um vetor $A[p\,..\,r]$ com $n = r - p + 1 \geq 1$.

▷ **Saída:** O máximo e o mínimo de $A[p\,..\,r]$.

$\textsc{MaxMin}(A, p, r)$

```
se p = r
    então devolva A[p], A[p]
    senão
        i ← ⌈(p + r)/2⌉
        max_E, min_E ← MaxMin(A, p, i − 1)
        max_D, min_D ← MaxMin(A, i, r)
        max_A ← max{max_E, max_D}
        min_A ← min{min_E, min_D}
        devolva max_A, min_A
```

Exemplo 3 - Complexidade

Exemplo 3 - Complexidade

- Estamos interessados no número (exato) $T(n)$ de **comparações entre elementos** feitos pelo algoritmo.

Exemplo 3 - Complexidade

- Estamos interessados no número (exato) $T(n)$ de **comparações entre elementos** feitos pelo algoritmo.
- Qual é o valor de $T(n)$?

$$T(n) = \begin{cases} 0, & n = 1, \\ T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + 2, & n > 1. \end{cases}$$

Exemplo 3 - Complexidade

- Estamos interessados no número (exato) $T(n)$ de **comparações entre elementos** feitos pelo algoritmo.
- Qual é o valor de $T(n)$?

$$T(n) = \begin{cases} 0, & n = 1, \\ T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + 2, & n > 1. \end{cases}$$

- A solução exata é $T(n) = 2(n-1)$ como mostraremos por indução.

Exemplo 3 - Complexidade

$$T(n) = \begin{cases} 0, & n = 1, \\ T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + 2, & n > 1. \end{cases}$$

Exemplo 3 - Complexidade

$$T(n) = \begin{cases} 0, & n = 1, \\ T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + 2, & n > 1. \end{cases}$$

- **Caso base:** $n = 1$. Então $T(1) = 0 = 2(n-1)$.

Exemplo 3 - Complexidade

$$T(n) = \begin{cases} 0, & n = 1, \\ T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + 2, & n > 1. \end{cases}$$

- **Caso base:** $n = 1$. Então $T(1) = 0 = 2(n-1)$.
- **Hipótese de indução:** suponha que $T(k) = 2(k-1)$ para $k < n$.

Exemplo 3 - Complexidade

$$T(n) = \begin{cases} 0, & n = 1, \\ T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + 2, & n > 1. \end{cases}$$

- **Caso base:** $n = 1$. Então $T(1) = 0 = 2(n-1)$.
- **Hipótese de indução:** suponha que $T(k) = 2(k-1)$ para $k < n$.
- **Passo de indução:** queremos mostrar que $T(n) = 2(n-1)$.

$$\begin{aligned} T(n) &= T(\left\lfloor \frac{n}{2} \right\rfloor) + T(\left\lceil \frac{n}{2} \right\rceil) + 2 \\ &= 2(\left\lfloor \frac{n}{2} \right\rfloor - 1) + 2(\left\lceil \frac{n}{2} \right\rceil - 1) + 2 \text{ (por HI)} \\ &= 2(\left\lfloor \frac{n}{2} \right\rfloor + \left\lceil \frac{n}{2} \right\rceil) - 2 \\ &= 2(n-1) \end{aligned}$$

e o resultado segue.

Exemplo 3 - Algoritmo

$$T(n) = \begin{cases} 0, & n = 1, \\ T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + 2, & n > 1, \end{cases}$$

Note que para $n = 2$ o algoritmo faz duas comparações quando uma é suficiente!

Exemplo 3 - Algoritmo

$$T(n) = \begin{cases} 0, & n = 1, \\ T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + 2, & n > 1, \end{cases}$$

Note que para $n = 2$ o algoritmo faz duas comparações quando uma é suficiente!

É possível resolver o problema com menos comparações mudando o caso base! Para isso, supomos que $n \geq 2$ e tomamos como casos bases $n = 2, 3$. Para $n = 2$ basta uma comparação e para $n = 3$ bastam três comparações.

Exemplo 3 - Algoritmo

$\triangleright$ **Entrada:** Um vetor $A[p \mathinner{..} r]$ com $n = r - p + 1 \geq 2$.
$\triangleright$ **Saída:** O máximo e o mínimo de $A[p \mathinner{..} r]$.

$\text{MAXMIN}(A, p, r)$
1. **se** $p \leq r - 2$ **então** $\triangleright\, n \leq 3$
2. $\quad\triangleright$ Complete o código!
3. $i \leftarrow \lceil (p + r)/2 \rceil$
4. $\max_E, \min_E \leftarrow \text{MAXMIN}(A, p, i - 1)$
5. $\max_D, \min_D \leftarrow \text{MAXMIN}(A, i, r)$
6. $\max_A \leftarrow \max\{\max_E, \max_D\}$
7. $\min_A \leftarrow \min\{\min_E, \min_D\}$
10. **devolva** $\max_A, \min_A$

Exemplo 3 - Complexidade

Exemplo 3 - Complexidade

- Qual o número de comparações $T(n)$ efetuado por este algoritmo?

$$T(n) = \begin{cases} 1, & n = 2, \\ 3, & n = 3, \\ T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + 2, & n > 3. \end{cases}$$

Exemplo 3 - Complexidade

- Qual o número de comparações $T(n)$ efetuado por este algoritmo?

$$T(n) = \begin{cases} 1, & n = 2, \\ 3, & n = 3, \\ T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + 2, & n > 3. \end{cases}$$

- Supondo que $n \geq 2$ é uma potência de 2, temos:

$$T(n) = \begin{cases} 1, & n = 2, \\ 2T(\frac{n}{2}) + 2, & n = 4, 8, \ldots, 2^k, \ldots. \end{cases}$$

Exemplo 3 - Complexidade

- Qual o número de comparações $T(n)$ efetuado por este algoritmo?

$$T(n) = \begin{cases} 1, & n = 2, \\ 3, & n = 3, \\ T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + 2, & n > 3. \end{cases}$$

- Supondo que $n \geq 2$ é uma potência de 2, temos:

$$T(n) = \begin{cases} 1, & n = 2, \\ 2T(\frac{n}{2}) + 2, & n = 4, 8, \ldots, 2^k, \ldots. \end{cases}$$

- Neste caso, podemos provar que $T(n) = \frac{3}{2}n - 2$ usando o método da substituição!

Exemplo 3 - Complexidade

- **Caso Base:** $T(2) = 1 = \frac{3}{2}2 - 2$.

Exemplo 3 - Complexidade

- **Caso Base:** $T(2) = 1 = \frac{3}{2}2 - 2$.
- **Hipótese de Indução:** Suponha que $T(n) = \frac{3}{2}n - 2$ onde $n = 2^{k-1}$ para algum $k \geq 2$.

Exemplo 3 - Complexidade

- **Caso Base:** $T(2) = 1 = \frac{3}{2}2 - 2$.
- **Hipótese de Indução:** Suponha que $T(n) = \frac{3}{2}n - 2$ onde $n = 2^{k-1}$ para algum $k \geq 2$.
- **Passo de Indução:** Queremos provar que $T(n) = \frac{3}{2}n - 2$ onde $n = 2^k$ para algum $k \geq 2$.

$$\begin{aligned} T(n) &= 2T(\tfrac{n}{2}) + 2 \\ &= 2(\tfrac{3}{4}n - 2) + 2 \text{ (por HI)} \\ &= \tfrac{3}{2}n - 2. \end{aligned}$$

Exemplo 3 - Complexidade

- **Caso Base:** $T(2) = 1 = \frac{3}{2}2 - 2$.
- **Hipótese de Indução:** Suponha que $T(n) = \frac{3}{2}n - 2$ onde $n = 2^{k-1}$ para algum $k \geq 2$.
- **Passo de Indução:** Queremos provar que $T(n) = \frac{3}{2}n - 2$ onde $n = 2^k$ para algum $k \geq 2$.

$$\begin{aligned} T(n) &= 2T(\tfrac{n}{2}) + 2 \\ &= 2(\tfrac{3}{4}n - 2) + 2 \text{ (por HI)} \\ &= \tfrac{3}{2}n - 2. \end{aligned}$$

- É possível provar que $T(n) = \lceil 3n/2 \rceil - 2$ para n qualquer, mas é um pouco mais complicado (Exercício!)
  Há uma versão iterativa que faz o mesmo número de comparações.

Exemplo 3 - Complexidade

Exemplo 3 - Complexidade

- Assintoticamente, os dois algoritmos para este problema são equivalentes, ambos $\Theta(n)$.

Exemplo 3 - Complexidade

- Assintoticamente, os dois algoritmos para este problema são equivalentes, ambos $\Theta(n)$.
- No entanto, o algoritmo de divisão e conquista faz menos comparações. A estrutura hierárquica de comparações no retorno da recursão evita comparações desnecessárias.

Leitura adicional

Dois exemplos de problemas de estatísticas de ordem nas seções 6.5.1 e 6.11.2 do livro de U. Manber, Algorithms: A Creative Approach, Addison-Wesley (1989):

- Determinação do maior e do menor elemento de um conjunto de n números reais (outra solução);
  veja também a Seção 9.1 do CLRS;
- Determinação dos dois maiores elementos de um conjunto de n números reais.