

AED - Resumo (Algoritmos)



Com o apoio moral de:



Alessio

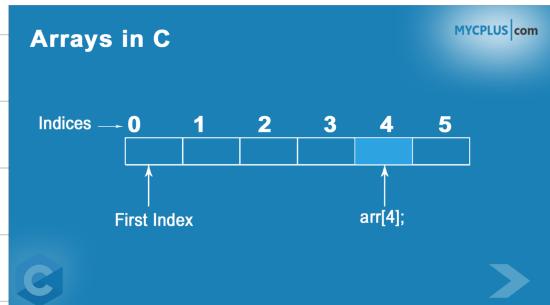


Fábio

Arrays:

```
double x_array[3];
int n_array[] = {1, 2, 3, 4};
int 2d_array[3][3];
```

- O tamanho não pode ser alterado



- O index é igual ao Python



Volta Python :)

Ponteiros e Arrays...

```
int my_array[3] = {1, 2, 3};
int* p_my_array = my_array;           // Pointer to first element !!
my_array[0] = 10; => Atende o array [0] para 10
*my_array = 20; => " " " " 20
*p_my_array = 30; => " " " " 30
printf("New value of first element: %d\n", my_array[0]); // Output ?
```

1º elemento (my_array[0])

{30, 2, 3}

Ciclo for

Sintaxe:

```
for(valor_inicial; condição_final; valor_incremento)
{
    instruções;
}
```

```
1. //Exemplo: Programa usando loop for em C
2. #include <stdio.h>
3. #include <conio.h>
4. int main(void)
5. {
6.     int contador; //variável de controle do loop
7.     for(contador = 1; contador <= 10; contador++)
8.     {
9.         printf("%d ", contador);
10.    }
11.
12.
13.    getch();
14.
15. }
```

Output:

[1,2,3,4,...,10]

} língua gram normal

} língua gram C

Calcular média com ciclo for)

```
1. #include <stdio.h>
2. #include <conio.h>
3. int main(void)
4. {
5.     //Declaração das variáveis
6.     float nota1, nota2, media;
7.     int i;
8.
9.     printf("---- Calculando a media para 10 alunos ----\n\n");
10.
11.    //Entrada de dados
12.    for (i = 1; i<=10; i++) (X)
13.    {
14.        printf("Digite a primeira nota do aluno %d\n",i);
15.        scanf("%f",&nota1); ) Input
16.
17.        printf("Digite a segunda nota do aluno %d\n",i);
18.        scanf("%f",&nota2); ) Input
19.
20.        //Processamento
21.        media = (nota1 + nota2) / 2; ) Calculo da média
22.
23.        //Saída
24.        printf("\nMedia do aluno %d = %.1f\n",i,media);
25.        printf("-----\n\n"); ) Print
26.    }
27.    getch();
28.    return(0);
29. }
```

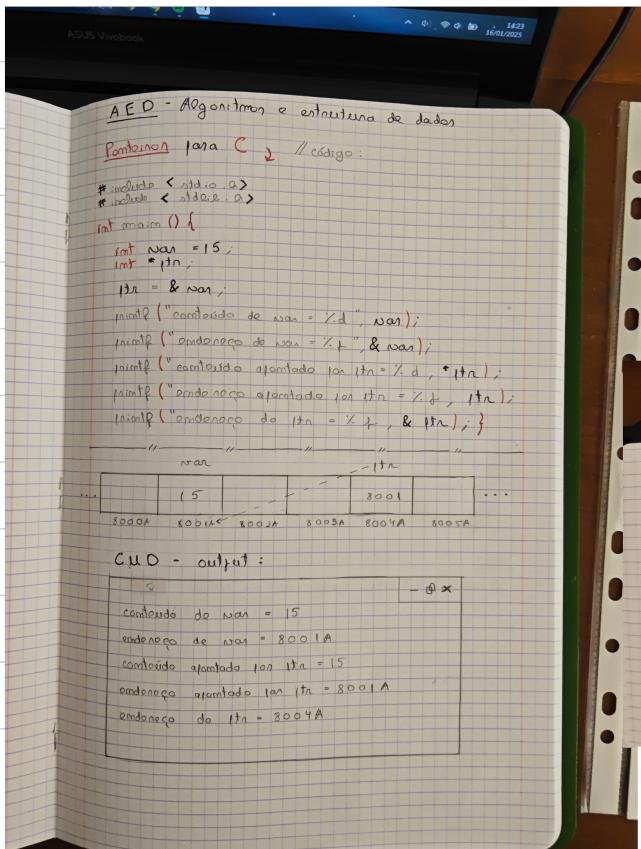
~~(X)~~ Este ciclo for vai ser executado até $i \leq 10$, logo quando $i = 11$, não vai ser executado

Pontoon

10



- Os conteúdos da memória são todos identificados, formando um endereço de memória



Nota

- & serve para identificar que estamos a falar do endereço da memória

* fr: o afastado por, contando do ordenado da comissão
que lhe afasta

Hn : o ondonojo da manançal

& ptm: o endereço do fonteiro

11. cōago da tricomeia lágima

25

conteúdo do rxn = 73

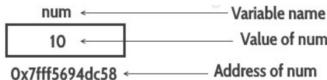


 YouTube WR Kits

Como saber o endereço de uma variável?

& – Obter o endereço de uma variável

```
int main(void) {  
    int num = 10;  
    printf("Value of variable num is: %d", num);  
  
    /* To print the address of a variable we use the %p  
     * format specifier and the ampersand (&) sign just  
     * before the variable name like &num  
     */  
    printf("\nAddress of variable num is: %p", &num);  
  
    return 0;  
}
```



outut: #0x7ffff5694dc58

Como aceder à variável apontada?

* – Aceder à variável apontada

```
int a = 10;  
int b = 99;  
printf("Value of a + b is: %d\n", a + b);  
  
int* p_a = &a;  
int* p_b = &b;  
int sum = *p_a + *p_b; ← 10 + 99 = 109  
  
*p_a = 1000;  
*p_b = 9999;  
  
printf("Value of sum is: %d\n", sum); // Output ? => 109  
printf("Value of a + b is: %d\n", a + b); // Output ? => 10 999
```

Ajuda diretamente os valores de **a** e **b**, porque os ponteiros manipulam diretamente o conteúdo dos endereços

Algoritmos - Análise



Tempo de execução



Complexidade Temporal

- N° de operações fundamentais em relação ao tamanho da entrada (n)
 - ↳ (comparações, somas, etc...)



Espaço de memória necessário

a b c a ...

Complexidade Espacial

Exemplos:



Exemplo 1

- Determinar o maior de 3 valores a, b, c ✓
- Quantas comparações são efetuadas ? 3
- Quantas atribuições são efetuadas ? 1
- A ordem dos dados é importante ? Sim
- Escrever os invariantes

"Ganhou: a"
 \downarrow
 $\text{maior} = \max(a, b, c)$

```
if (a > b) { }  
  if (a > c)  
    maior = a;  
  else  
    maior = c;  
} else { }  
  if (b > c)  
    maior = b;  
  else  
    maior = c;  
}
```

Se $a > b$, o algoritmo acaba logo,
caso contrário vai comparar +1 vez

Exemplo 2

- Determinar o maior de 3 valores a, b, c
- Quantas comparações são efetuadas? 2 ✓
- Quantas atribuições são efetuadas? ⊗ ✓
- A ordem dos dados é importante?
- Escrever os invariantes

```
• inicialização : maior = a  
• 1º if      : maior = max (a, b)  
• 2º if      : maior = max (a, b, c)
```

⊗ Vou defender do imfut, no 1º caso
o programa pode fazer até 3 atribuições, 1 no inicio, 2 depois
 $\begin{array}{l} \text{maior} = a \\ \text{maior} = b \\ \text{maior} = c \end{array}$

```
maior = a;  
if (b > maior) 1  
    maior = b;  
if (c > maior) 12  
    maior = c;  
...
```

Não, porque não remane
comparar com todos

Exemplo 3

```
inicializar array contador[256];  
de i = 0 até 256:  
    contador[i] = 0;  
enquanto não fim de ficheiro:  
    ler próximo carater;  
    incrementar contador[próximo carater];
```

- Contar o nº de ocorrências de caracteres num ficheiro
- Inicialização : quantas atribuições ao array? 256 atribuições → ASCII
- Leitura do ficheiro : quantas vezes incrementamos o array? = nº caracteres
- Qual é o factor que determina o desempenho?
- O esforço da fase de inicialização é importante?
Sim → O tamanho do ficheiro e velocidade de leitura
↳ começar no 0

Exemplos de algoritmos

Binary Search

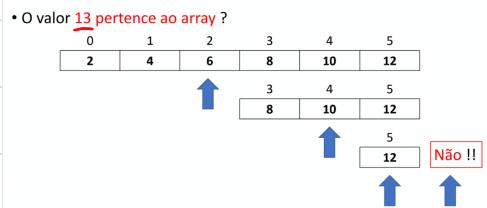
→ De modo geral como funciona?

- O valor 2 pertence ao array?

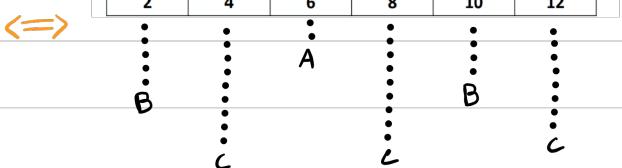
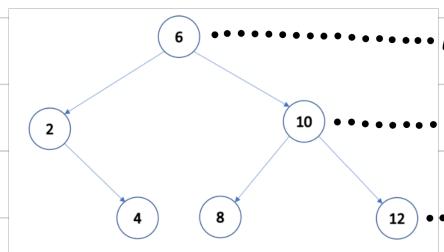


Objetivo: Encontrar um valor alvo (target) num array (sorted array)
Em cada iteração, metade dos elementos são eliminados.

→ E se o valor não estiver no array?



→ Pode - se perceber melhor como uma árvore binária?



Código em C do Binary search



Array tamanho do array N° procurado (?)

```
int binSearch( int a[], int n, int x ) {  
    int left = 0; int right = n - 1;  
    while( left <= right ) {  
        int middle = (left + right) / 2;  
        if(a[middle] == x) return middle;  
        if(a[middle] > x) right = middle - 1;  
        else left = middle + 1;  
    }  
    return -1;  
}
```

cálculo da posição meio do array

// Divisão inteira

Caso se o meio for igual ao N° procurado (x)
Encontrar a janela de memória

Dá return -1, se não for encontrado dentro do array [a]

Exemplo:

0 1 2 3 4 5 6 7 8 9 10 11 12 13

[1, 4, 5, 7, 9, 12, 15, 18, 19, 22, 25, 29, 40, 50]

left

right

① Encontra o meio do array

average of (0, 13) = 6 \Leftrightarrow int middle = (left + right) / 2;

6

[1, 4, 5, 7, 9, 12, 15, 18, 19, 22, 25, 29, 40, 50]

left

mid

right

2 Comparações

$15 > \text{target}$, logo
(12)

$[1, 4, 5, 7, 9, 12, 15, 18, 19, 22, 25, 29, 40, 50]$

left

right

3 Atualizar o array e repetir o processo até encontrar.

$[1, 4, 5, 7, 9, 12, 15, 18, 19, 22, 25, 29, 40, 50]$

left mid right

• • •

$[1, 4, 5, 7, 9, 12, 15, 18, 19, 22, 25, 29, 40, 50]$

left, right, mid

$\Rightarrow \text{return } 5$, o index

de 12

"target"



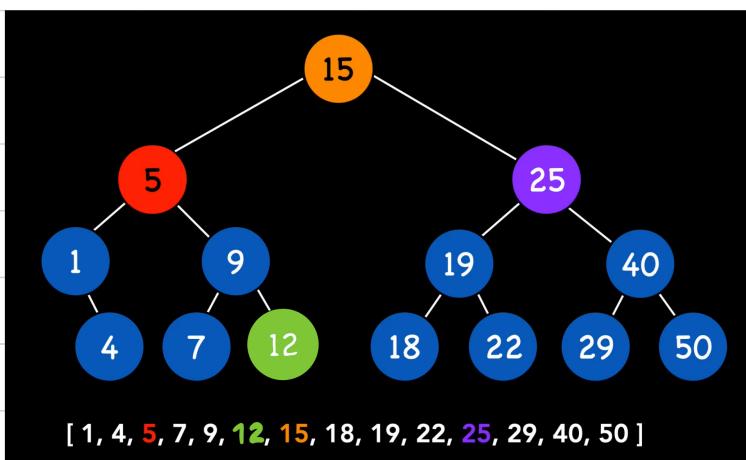
Complexidade
Temporal



$O(\log m)$



$\log_2(m) + 1$



Selection Sort)

Objetivo: Ordenar um array

→ Durante cada iteração, vai se selecionar o elemento mais pequeno da parte desordenada e mover para a parte ordenada

III - parte ordenada

↑ - elemento atual

↑ - elemento mínimo

2	8	5	3	9	4	1
---	---	---	---	---	---	---

2	8	5	3	9	4	1
---	---	---	---	---	---	---



2	8	5	3	9	4	1
---	---	---	---	---	---	---

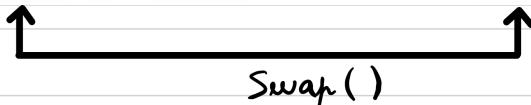


$1 < 2$, logo vamos atualizar o array

2	8	5	3	9	4	1
---	---	---	---	---	---	---



2	8	5	3	9	4	1
---	---	---	---	---	---	---



1	8	5	3	9	4	2
---	---	---	---	---	---	---

1	8	5	3	9	4	2
---	---	---	---	---	---	---



- (continuar o processo até o array estiver **ordenado (sorted)**)
-
-

1	2	3	4	5	8	9
---	---	---	---	---	---	---



Complexidade
Temporal



$O(m^2)$

```
void selectionSort( int a[], int n ) {
    for( int k = n - 1; k > 0; k-- ) {
        int indMax = 0;
        for( int i = 1; i <= k; i++ ) {
            if( a[i] >= a[indMax] ) indMax = i;
        }
        if( indMax != k ) swap( &a[indMax], &a[k] );
    }
}
```

- Seleção em Sort - N° de comparações)

→ N° fixo de comparações (máx imposto se o array está ordenado ou não.)

$$\sum_{k=1}^{n-1} k = \frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2}$$

$O(m^2)$

(Simplificam para originalmente $O(m^2)$)

Best Case (Ω)



Quando a entrada
já está **ordenada**



$$\begin{array}{ccc} \text{(Comparações)} & O(m^2) & = \\ \text{Trocas} & O(m) & \neq \end{array} \quad \begin{array}{ccc} \text{(Comparações)} & O(m^2) & \\ \text{Trocas} & O(m) & \end{array}$$

Worst Case O



Quando a entrada
está na ordem **inversa**



- O N° de Trocas vai ser diferente $0 > \Omega$
- O N° de Comparações vai ser igual

Ejercicios:

Considere o array ordenado de 8 elementos. Usando o algoritmo de procura binária:

0	1	2	3	4	5	6	7
1	3	5	7	9	11	13	15

- a) O elemento de valor 3 é encontrado ao fim de 2 tentativas.
b) O elemento de valor 13 é encontrado ao fim de 3 tentativas.
c) Ambas estão corretas.
d) Nenhuma está correta.

~ → "target"

Considere o seguinte *array* de 6 elementos que é ordenado, por ordem crescente, usando o algoritmo “**Selectionsort**”.

0	1	2	3	4	5
6	5	4	3	2	1

- a) São efetuadas **15 comparações** entre elementos do *array*.
 - b) São efetuadas **3 trocas** entre elementos do *array*.
 - c) Ambas estão corretas.
 - d) Nenhuma está correta.

Considere o array ordenado de 15 elementos. Usando o algoritmo de procura binária:

- a) O elemento de valor 11 é encontrado ao fim de 3 tentativas. ✓
 - b) O elemento de valor 1 é encontrado ao fim de 4 tentativas.
 - c) Ao fim 4 tentativas conclui-se que o valor 28 não pertence ao array.
 - d) Todas estão corretas.

Considere o array ordenado de 15 elementos. Usando o algoritmo de

Exemplo de busca binária:

- a) O elemento de valor 11 é encontrado ao fim de 3 tentativas.
b) O elemento de valor 1 é encontrado ao fim de 4 tentativas. ✓
c) Ao fim 4 tentativas conclui-se que o valor 28 não pertence ao array.
d) Todas estão corretas.

Considerando o array ordenado de 15 elementos. Usando o algoritmo de procura binária:

- a) O elemento de valor 11 é encontrado ao fim de 3 tentativas.
 - b) O elemento de valor 1 é encontrado ao fim de 4 tentativas.
 - c) Ao fim 4 tentativas conclui-se que o valor 28 não pertence ao array.

→ 5 comparações
 1 troca

$$1] \underset{\uparrow}{5} 4 3 2 6$$

. Comparações = $5 + 4 + 3 + 2 + 1$
 = 15

→ 4 comparações
 1 troca

$$12] \underset{\uparrow}{4} 3 5 6$$

. Trocas = 3

→ 3 comparações
 1 troca

$$123] \underset{\uparrow}{4} 5 6$$

→ 2 comparações
 0 troca

$$1234] \underset{\uparrow}{5} 6$$

→ 1 comparação
 0 troca

$$12345] \underset{\uparrow}{6}$$

Bubble Sort)

- Percorro o array da **esquerda para a direita** e troca elementos adjacentes, se estiverem **fornados de ordem**
$$(1, \downarrow 3, 2, 8, 10, \dots)$$
- A **última ocorrência** do maior elemento **fica na sua posição final**

Exemplo :

Ondemar!

0	1	2	3	4
7	2	6	4	3



0	1	2	3	4
7	2	6	4	3

7	2	6	4	3
2	7	6	4	3

7 > 2 ?, se **SIM**, trocam! (\Rightarrow)

```
if( a[i] > a[i + 1] ) {  
    swap( &a[i], &a[i + 1] );
```

(Parte do código do Bubble Sort, código completo mais à frente)

:

0	1	2	3	4
7	2	6	4	3

7	2	6	4	3
2	7	6	4	3
2	6	7	4	3
2	6	4	7	3
2	6	4	3	7

1ª iteração

- 4 comparações

+

- 4 trocas

- Após chegar ao fim, é preciso voltar a fazer outra iteração

2	6	4	3	7
---	---	---	---	---

2>6, NÃO, NÃO TROCAR

2	6	4	3	7
2	6	4	3	7
2	4	6	3	7

6>4, SIM

2	4	6	3	7
2	4	3	6	7

✓

6>3, SIM

✓

- 3 comparações

+

- 2 trocas

∴ (Falta a 3ª iteração)

0	1	2	3	4
2	3	4	6	7

4ª iteração

2	3	4	6	7
2	3	4	6	7

- 1 comparação

+

- 0 trocas

```

void bubbleSort( int a[], int n ) {
    int k = n; int stop = 0;
    while( stop == 0 ) {
        stop = 1; k--;
        for( int i = 0; i < k; i++ )
            if( a[i] > a[i + 1] ) {
                swap( &a[i], &a[i + 1] );
                stop = 0;
            }
    }
}

```

N° de comparações)

- Melhor Caso ? - Array ordenado

$\rightarrow O(m)$

$$B_c(m) = m - 1$$

- Pior Caso ?

- Array pela ordem inversa, sem elementos repetidos

$\rightarrow O(m^2)$

$$W_c(m) = \frac{(m-1) \cdot m}{2}$$

$$= \frac{m^2}{2} - \frac{m}{2}$$

N° de trocas)

- Melhor Caso ? - Array ordenado

$\rightarrow O(1) \rightarrow B_c(m) = 0$

- Pior Caso ? - Array pela ordem inversa, sem elementos repetidos
- 1 troca para cada comparação

$\rightarrow O(m^2)$

$$W_c(m) = \frac{(m-1) \cdot m}{2}$$

$$= \frac{m^2}{2} - \frac{m}{2}$$

Insertion Sort

- O elemento $a[0]$ constitui um subconjunto de um só elemento
- Insere ordenadamente o elemento $a[1]$ neste subconjunto, o que origina um subconjunto com 2 elementos

Exemplo :

0	1	2	3	4
7	2	6	4	3

1ª Iteração

- 1 comparação
- +
- 1 deslocamento

7	2	6	4	3
2	7	6	4	3

2ª Iteração

- 2 comparações
- +
- 1 deslocamento

2	7	6	4	3
2	6	7	4	3

$4 < 7 ? \checkmark \text{ } \textcolor{blue}{\cancel{0}} \text{ } \textcolor{red}{\cancel{X}}$

3ª Iteração

2

6

7

4

3



2

6

4

7

3



$4 < 6 ? \checkmark \text{ } \textcolor{blue}{\cancel{0}} \text{ } \textcolor{red}{\cancel{X}}$

2

4

6

7

3

$4 < 2, \textcolor{red}{X} \text{ } \textcolor{blue}{\cancel{0}}$

- 3 comparações 0

+

- 2 deslocamentos X

4º Iteração

2

4

6

7

3

2

4

6

3

7

2

4

3

6

7

2

3

4

6

7

- 4 comparações

+

- 3 deslocamentos

✓ Ordenado

Nota: Este algoritmo também pode começar pela direita (\leftarrow)

```
void insertionSort( int a[], int n ) {
    for( int i = 1; i < n; i++ )
        if( a[i] < a[i - 1] )
            insertElement( a, i, a[i] )
}
```

```
void insertElement( int sorted[], int n, int elem ) {
    // Array sorted está ordenado
    // Há espaço para acrescentar mais um elemento
    int i;
    for( i = n - 1; (i >= 0) && (elem < sorted[i]); i-- )
        sorted[i + 1] = sorted[i];
    sorted[i + 1] = elem;
}
```

Nº de comparações:

• Melhor caso $\rightarrow B_c(m) = m - 1 \rightarrow O(m)$

↳ É ser um array ordenado

↳ A função auxiliar auxiliar nunca é chamada.

• Pior caso $\rightarrow W_c(m) = \frac{m-1}{2} \times (m+2) \rightarrow O(m^2)$

↳

↳ A função auxiliar é sempre chamada!

↳ tem sempre o comportamento do pior caso

A função InsertElement() precisa de ser executada ao máximo em cada iteração

- Considera-se que em cada iteração a função auxiliar tem sempre o comportamento do caso médio ($i/2 + 1$)

$$A_c(n) \approx \sum_{i=1}^{n-1} \left[1 + \left(\frac{i}{2} + 1 \right) \right] = \left[2(n-1) + \frac{n(n-1)}{4} \right]$$

$$A_c(n) \approx \frac{n^2}{4} + \frac{7n}{4}$$

Resumo - Selection, Bubble, Insertion Sort

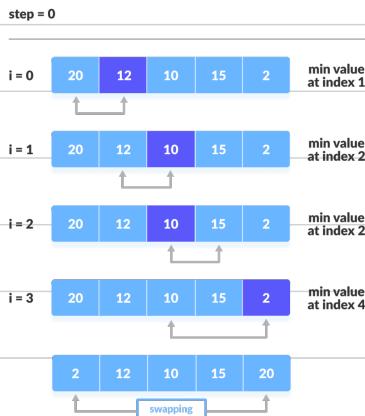
Comparações - Algoritmos Quadráticos ↴

	Pior Caso	Caso Médio	Melhor Caso
Selection Sort	$\approx \frac{n^2}{2}$	$\approx \frac{n^2}{2}$	$\approx \frac{n^2}{2}$
Bubble Sort	$\approx \frac{n^2}{2}$	$\approx \frac{n^2}{3}$	$n - 1$
Insertion Sort	$\approx \frac{n^2}{2}$	$\approx \frac{n^2}{4}$	$n - 1$

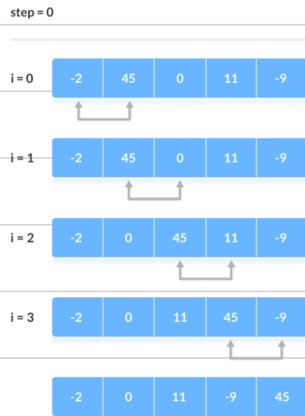
Trocas / Deslocamentos - Algoritmos Quadráticos ↴

	Pior Caso	Caso Médio	Melhor Caso
Selection Sort	$n - 1$	$\approx n - \ln n$	0
Bubble Sort	$\approx \frac{n^2}{2}$	$\approx \frac{n^2}{6}$	0
Insertion Sort	$\approx \frac{n^2}{2}$	$\approx \frac{n^2}{8}$	0

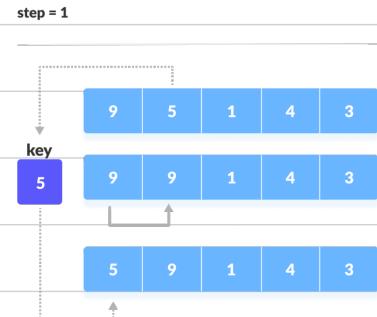
Selection Sort ↴



Bubble Sort ↴

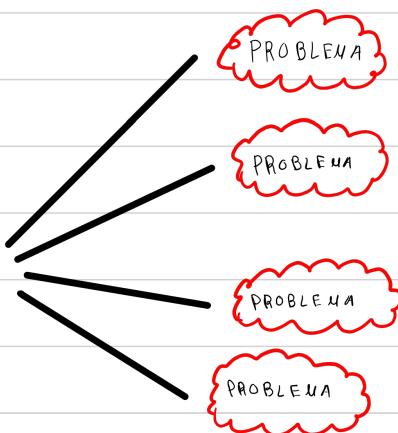
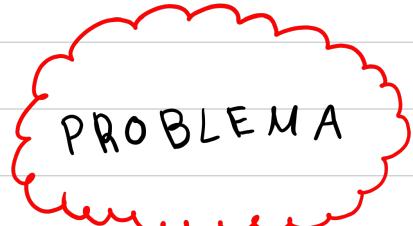


Insertion Sort ↴



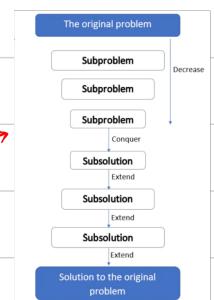
Algoritmos Recursivos

Ideia: ↴

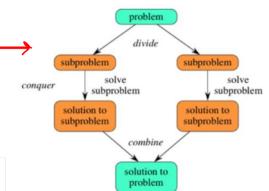


- Sub-dividir em problemas mais pequenos do mesmo tipo, com o objetivo da diminuição da "dificuldade"

$$\begin{aligned} m! &= m \times (m-1)! \\ 0! &= 1 \end{aligned} \quad \left. \begin{array}{l} \\ \end{array} \right\} \rightarrow \text{Notações}$$



- Diminuir-para-Reinar / Decrease-and-Conquer
 - Resolver 1 só subproblema em cada passo do processo recursivo
 - Lista / cadeia de chamadas recursivas
- Dividir-para-Reinar / Divide-and-Conquer
 - Resolver 2 ou mais subproblemas em cada passo do processo recursivo
 - Árvore de chamadas recursivas



1. Decrease-and-Conquer

Identifican → Resolver → Combinar as soluções
 uma instância menor → as instâncias menores → para obter uma solução / resposta final

Complexidade: $T(n) = T(n/2) + f(n)$

2. Procura Binária (Versão Recursiva)



- Divide o array a meio e determina o próximo sub-array com base na comparação

$B_c \rightarrow$ Encontram o elemento na primeira comparação $\rightarrow O(1)$

$W_c \rightarrow$ Necessidade de $O(\log_2 m)$ iterações

Exemplo de um código

```
int pesqBinRec(int* v, int esq, int dir, int valor) {
    unsigned int meio;
    if (esq > dir) return -1;
    meio = (esq + dir) / 2;
    contadorComps++;
    if (v[meio] == valor) {
        return meio;
    }
    contadorComps++;
    if (v[meio] > valor) {
        return pesqBinRec(v, esq, meio - 1, valor);
    }
    return pesqBinRec(v, meio + 1, dir, valor);
}
```

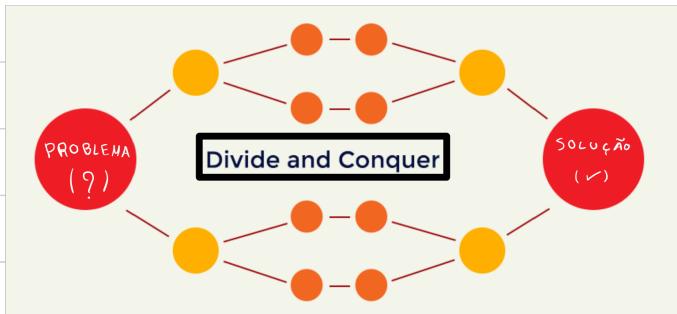
3. Divide-and-Conquer

Dividir → Conquistar → Combinar

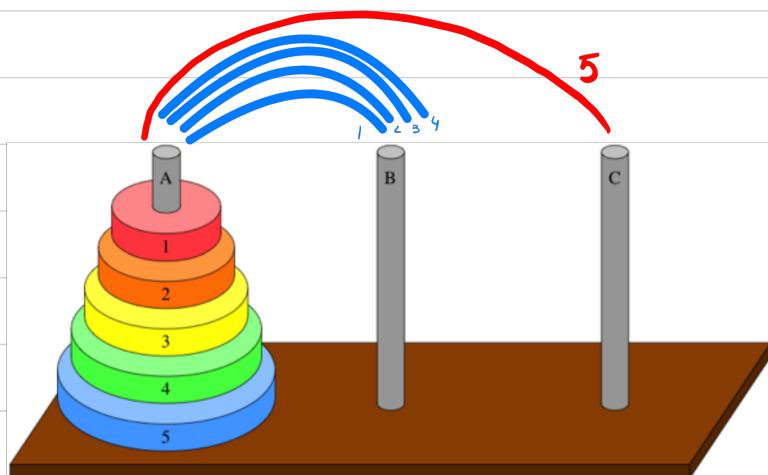
em problemas maiores

Resolver os problemas menores

Juntar as soluções dos sub-problemas



4. Torre de Hanói

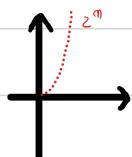


Movimentos necessários para lidar com os
2 subconjuntos de m-1 discos
(Ass. cada é feito)

$$M(m) = 2 M(m-1) + 1 \rightarrow \text{Movimento para transferir}$$

m° de movimentos $\rightarrow m^{\circ}$ do discos

$M(m) \in O(2^m) \rightarrow$ É EXPONENCIAL



5. Mergesort

Split - Divide

2	8	5	3	9	4	1	7
---	---	---	---	---	---	---	---

Ondeman!
C/ Recursão

2	8	5	3
---	---	---	---

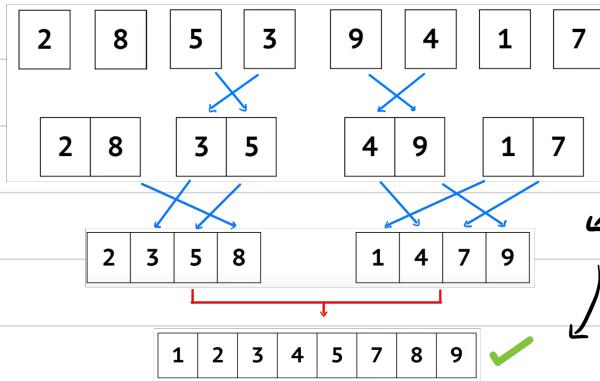
9	4	1	7
---	---	---	---

2	8	5	3
---	---	---	---

9	4	1	7
---	---	---	---

2	8	5	3	9	4	1	7
---	---	---	---	---	---	---	---

Fusão



Ondeman os arrays temporários!

Por fim, juntar e ondemar os arrays temporários e originar o array final

```

mergesort (array a)
if ( n == 1 )
    return a

arrayOne = a[0] ... a[n/2]
arrayTwo = a[n/2+1] ... a[n]

arrayOne = mergesort ( arrayOne )
arrayTwo = mergesort ( arrayTwo )

return merge ( arrayOne, arrayTwo )

```

Divide os arrays

```

merge ( array a, array b )
array c

while ( a and b have elements )
    if ( a[0] > b[0] )
        add b[0] to the end of c
        remove b[0] from b
    else
        add a[0] to the end of c
        remove a[0] from a

// At this point either a or b is empty

while ( a has elements )
    add a[0] to the end of c
    remove a[0] from a

while ( b has elements )
    add b[0] to the end of c
    remove b[0] from b

return c

```

Combina os arrays

⚠️
Complexidade
Temporal

O(m log m)