

# IMD0030 – LINGUAGEM DE PROGRAMAÇÃO I

Aula 04 – Recursividade

---

# Recursão

- Método de programação no qual uma função chama a si própria
- Dividir e conquistar
  - problemas menores
  - combinar soluções
  - soluções mais simples e fáceis de se analisar



---

## Cálculo do fatorial (abordagem recursiva)

- $4! = 4 \cdot 3 \cdot 2 \cdot 1$
- $3! = 3 \cdot 2 \cdot 1$
- $2! = 2 \cdot 1$
- $1! = 1$
- $0! = 1$

- $4! = 4 \cdot 3!$
- $3! = 3 \cdot 2!$
- $2! = 2 \cdot 1!$
- $1! = 1 \cdot 0!$
- $0! = 1$

$$N! = N \cdot (N-1)!$$

$$0! = 1$$

---

---

# Cálculo do fatorial (abordagem recursiva)

$0! = 1$  (caso base)

$N! = N \cdot (N-1)!$

- quando a recursão deve parar
- mudar o valor do parâmetro da chamada recursiva

```
3  #include <iostream>
4  using namespace std;
5
6  int fatorial(int N)
7  {
8      if (N == 0)    critério de parada
9      {
10         return 1;
11     }
12     else
13         return N*fatorial(N-1); parâmetro da chamada recursiva
14 }
15
16
17 int main(void)
18 {
19     int n, F;
20
21     cout << "Digite um número:" << endl;
22     cin >> n;
23
24     F = fatorial(n);
25
26     cout << "O fatorial de " << n << " é: " << F << endl;
27
28     return 0;
29 }
```

---

---

# Cálculo do fatorial (abordagem recursiva)

$0! = 1$  (caso base)

$N! = N \cdot (N-1)!$

fatorial(0)
fatorial(1)
fatorial(2)
fatorial(3)
fatorial(4)
main()

return(1)
return(1*1)
return(2*1*1)
return(3*2*1*1)
return(4*3*2*1*1)
main()


# Cálculo do fatorial (abordagem recursiva)

$0! = 1$  (caso base)

$N! = N \cdot (N-1)!$

Posso usar caso base  
“ $1! = 1$ ”?

Por que?



```
3  #include <iostream>
4  using namespace std;
5
6  int fatorial(int N)
7  {
8      if (N == 0)    critério de parada
9      {
10         return 1;
11     }
12     else
13         return N*fatorial(N-1); parâmetro da chamada recursiva
14 }
15
16
17 int main(void)
18 {
19     int n, F;
20
21     cout << "Digite um número:" << endl;
22     cin >> n;
23
24     F = fatorial(n);
25
26     cout << "O fatorial de " << n << " é: " << F << endl;
27
28     return 0;
29 }
```

# Função Recursiva


Sempre deve ter **nesta sequência**:

- Caso base (critério de parada)
- Caso de recursão

Opcional

- Condição de terminação (caso de entrada errada dos dados)

```
3  #include <iostream>
4  using namespace std;
5
6  int fatorial(int N)
7  {
8      if (N == 0)    critério de parada
9      {
10         return 1;
11     }
12     else
13         return N*fatorial(N-1); parâmetro da chamada recursiva
14 }
15
16
17 int main(void)
18 {
19     int n, F;
20
21     cout << "Digite um número:" << endl;
22     cin >> n;
23
24     F = fatorial(n);
25
26     cout << "O fatorial de " << n << " é: " << F << endl;
27
28     return 0;
29 }
```



---

# Soma de N números inteiros

Utilizando recursão, faça um programa que retorne a soma de 1 a N, onde N é um número inteiro positivo fornecido pelo usuário

---



---

# Máximo Divisor Comum

Algoritmo de Euclides

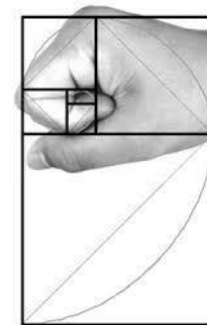
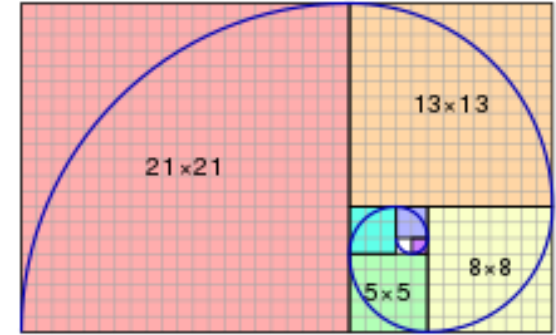
$$\text{mdc}(p, q) = \begin{cases} p, & \text{se } q = 0 \\ \text{mdc}(q, p \% q) & \text{caso contrário} \end{cases}$$

---

# Fibonacci

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...

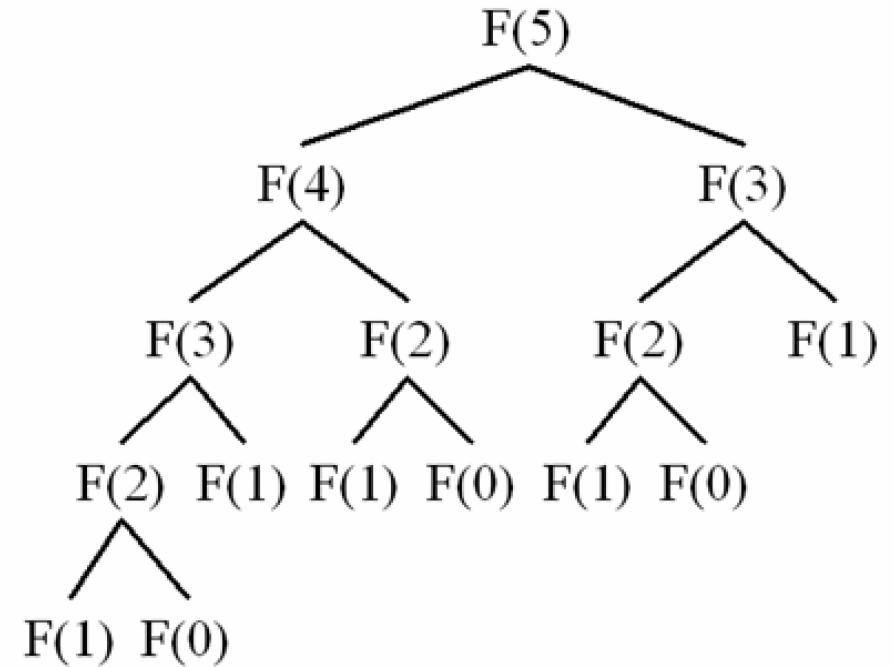
- $F_0 = 0, F_1 = 1$
- $F_n = F_{n-1} + F_{n-2}$



---

# Fibonacci

- $F_{31}$ : 4.356.617 chamadas
- $F_{32}$ : 7.049.155 chamadas



---

# Recursão vs Iteração

- quando utilizar recursão?
- quais os pontos críticos na implementação de um programa recursivo?

<b>Recursão</b>	<b>Iteração</b>
Estruturas de seleção <i>if</i> , <i>if-else</i> ou <i>switch</i> .	Estruturas de repetição <i>for</i> , <i>while</i> ou <i>do-while</i> .
Repetição por chamadas de funções repetidas.	Repetição explícita.
Termina quando um caso base é reconhecido.	Termina quando teste do laço falha.
Pode ocorrer infinitamente.	Pode ocorrer infinitamente.
Lento.	Rápido.
Soluções simples e de fácil manutenção.	Soluções complicadas e de difícil manutenção.

---

# Recursão vs Iteração

Recursão	Iteração
Estruturas de seleção <i>if</i> , <i>if-else</i> ou <i>switch</i> .	Estruturas de repetição <i>for</i> , <i>while</i> ou <i>do-while</i> .
Repetição por chamadas de funções repetidas.	Repetição explícita.
Termina quando um caso base é reconhecido.	Termina quando teste do laço falha.
Pode ocorrer infinitamente.	Pode ocorrer infinitamente.
Lento.	Rápido.
Soluções simples e de fácil manutenção.	Soluções complicadas e de difícil manutenção.

[www.ic.unicamp.br/~oliveira/doc/mc102\\_2s2004/Aula19.pdf](http://www.ic.unicamp.br/~oliveira/doc/mc102_2s2004/Aula19.pdf)

- Recursão pode ficar MUITO LENTO ao ponto de parecer “travamento”.  
Por que?
  - Resposta: disciplina Arquitetura + Sistemas Operacionais

---

# Tipos de recursão

- recursão direta
    - função chama a mesma função
  - recursão indireta
    - uma função “a” é dita indireta se chama uma função “b” que por sua vez chama a função “a”
  - recursão em cauda
    - não existe processamento a ser feito depois de encerrada a chamada recursiva (não há retorno de informação)
-

---

# Recursão em cauda

```
void imprime(int n) {  
    if (n == 0)  
        return;  
    cout << n;  
    return imprime(n-1);  
}
```

```
int main(void) {  
    imprime(4);  
    return 0;  
}
```

---

# Recursão em cauda



```
int fact(int n){  
    if (n == 0)  
        return 1;  
  
    return n*fact(n-1);  
}  
  
int main(){  
    cout << fact(5);  
    return 0;  
}
```

```
int fact(int n, int a){  
    if (n == 0)  
        return a;  
  
    return fact(n-1, n*a);  
}  
  
int main(){  
    cout << fact(5,1);  
    return 0;  
}
```

---



---

# Exercício Sala de Aula

1) Implemente Integral definida utilizando regra do trapézio. Forneça:

- Função  $f(x)$ ;
- Limites inferior e superior “a” e “b”;
- Número de trapézios “n”.

Explicação a seguir

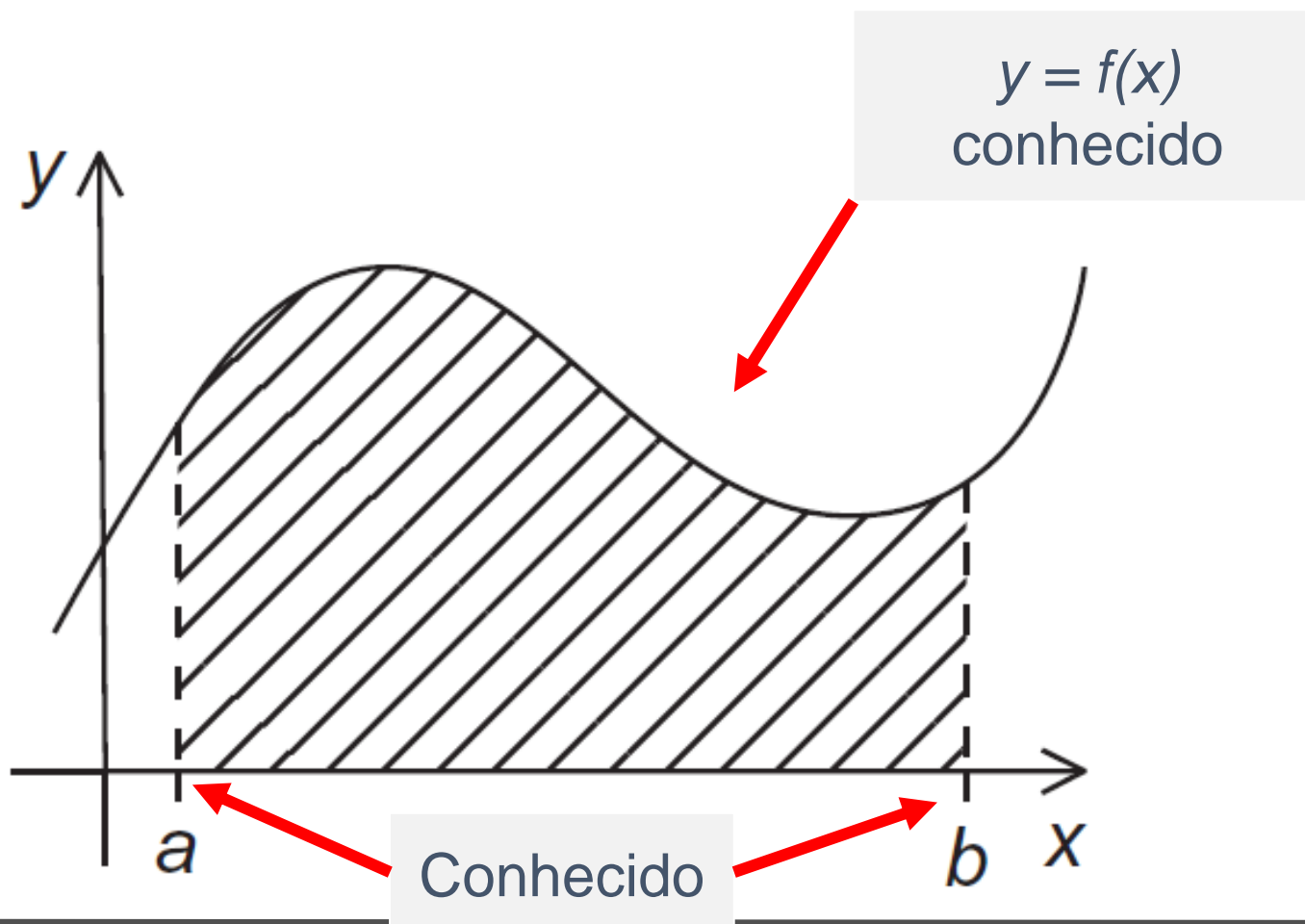
---

---

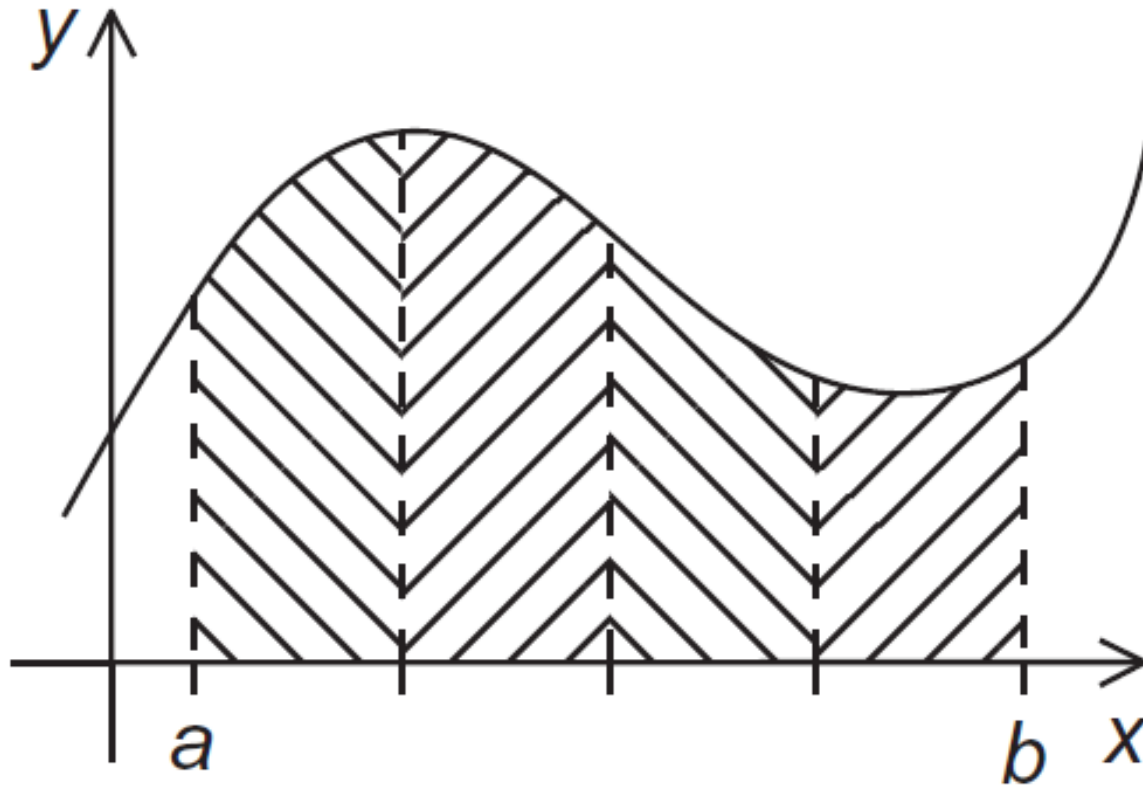
# Integral definida por Regra do Trapézio

---

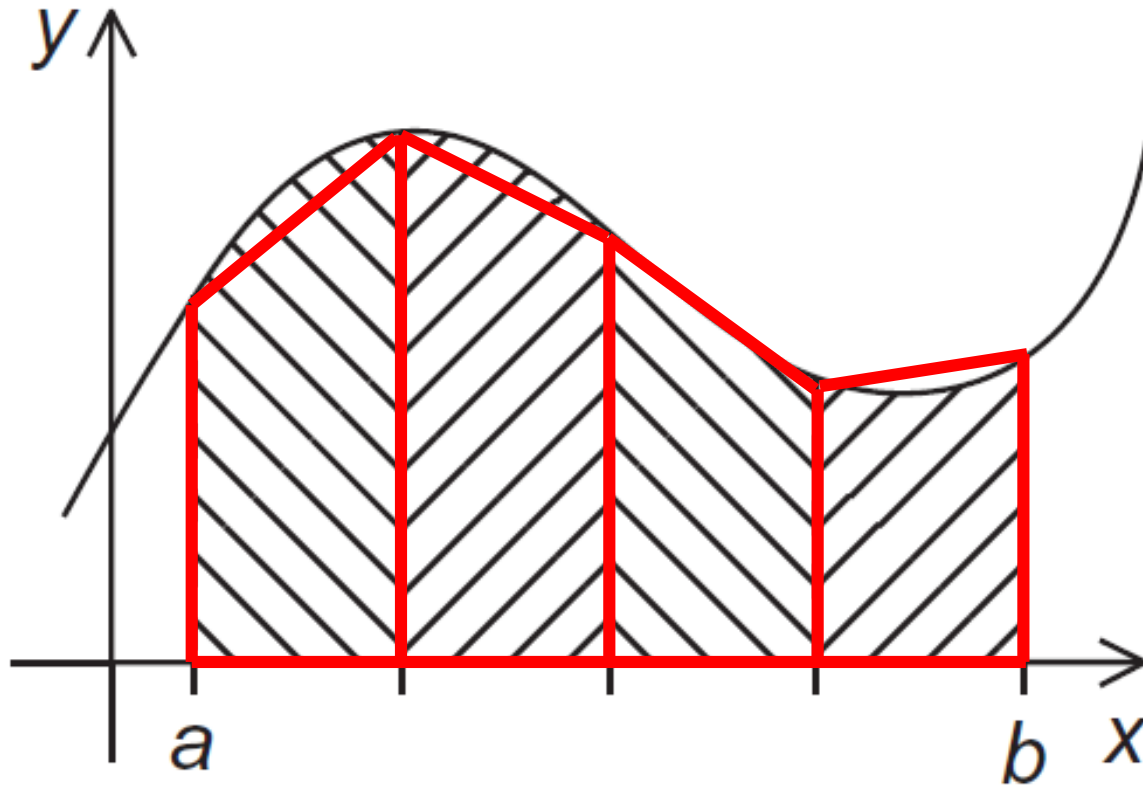
# Regra dos Trapézios



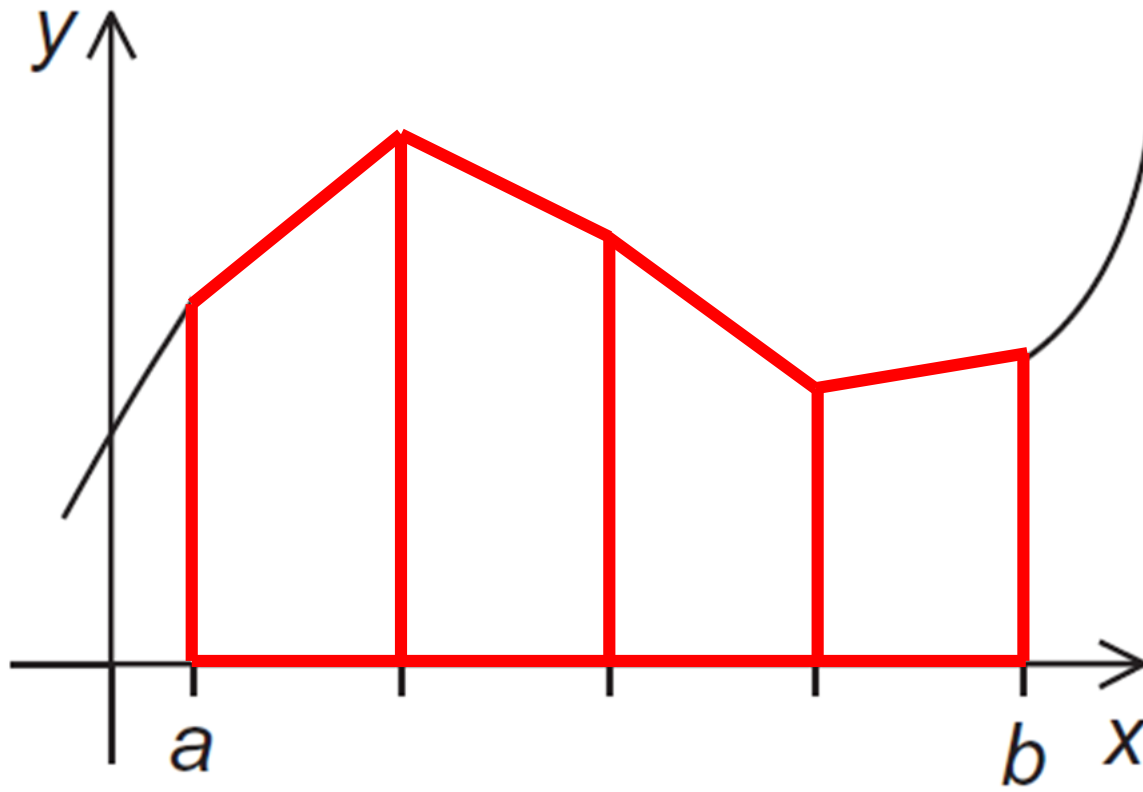
# Regra dos Trapézios



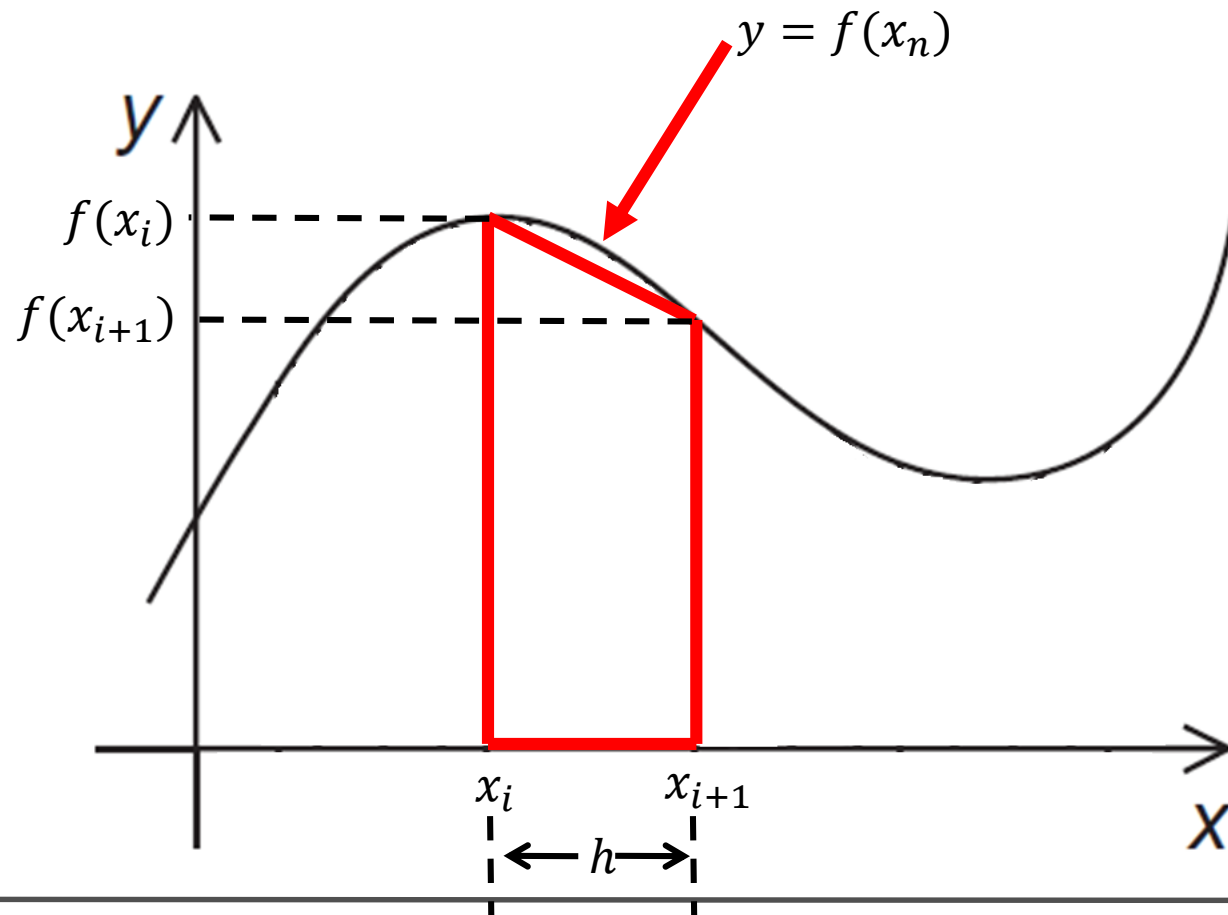
# Regra dos Trapézios



# Regra dos Trapézios



# Regra dos Trapézios



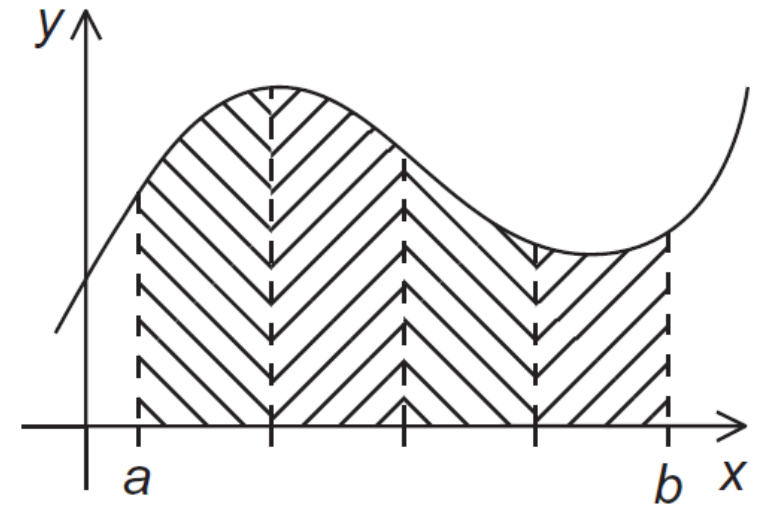
# Regra dos Trapézios

$$\text{Área de um trapézio} = \frac{h}{2} [f(x_i) + f(x_{i+1})]$$

$$h = \frac{b - a}{n} \quad \leftarrow \text{Nº de trapézios}$$

$$x_0 = a, \quad x_1 = a + h, \quad x_2 = a + 2h, \quad \dots \quad x_{n-1} = a + (n - 1)h, \quad x_n = b$$

$$\text{Area de trapézio} = \frac{h}{2} (B + b)$$





# Regra dos Trapézios

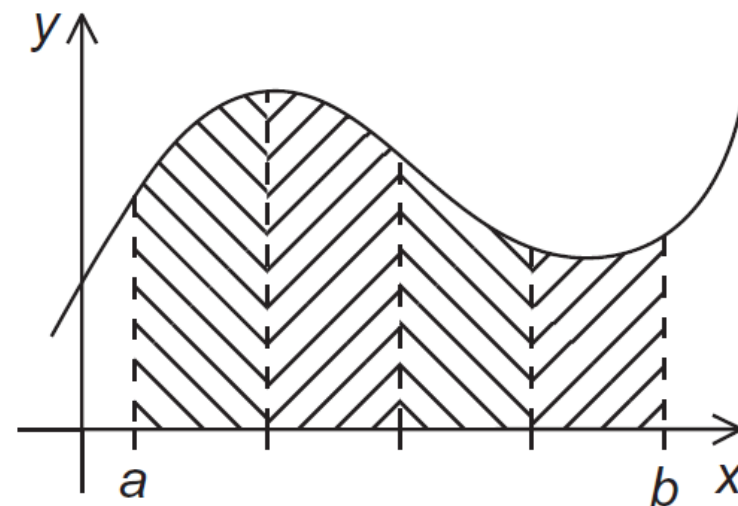
## Solução 1

Considere que  $f(x_n) = y_n$

$$\text{Área total} = \text{Trapézio}_1 + \text{Trapézio}_2 + \cdots + \text{Trapézio}_{n-1} + \text{Trapézio}_n$$

$$\text{Área total} = \frac{h}{2}[y_0 + y_1] + \frac{h}{2}[y_1 + y_2] + \frac{h}{2}[y_2 + y_3] + \cdots + \frac{h}{2}[y_{n-2} + y_{n-1}] + \frac{h}{2}[y_{n-1} + y_n]$$

$$\text{Area de trapézio} = \frac{h}{2}(B + b)$$



# Regra dos Trapézios

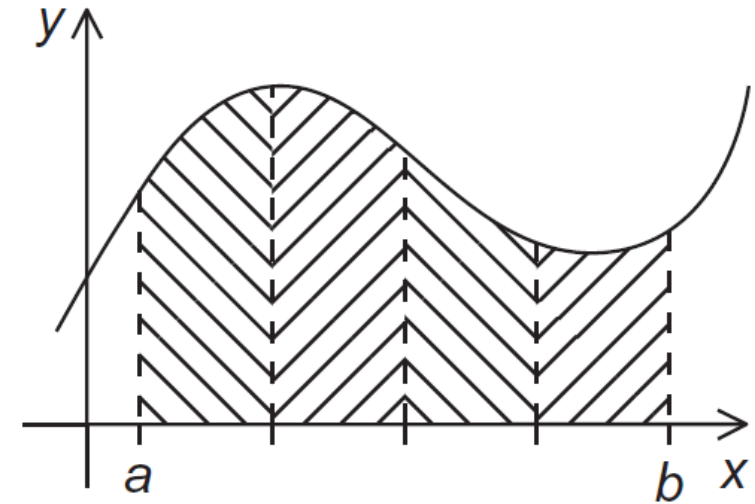
## Solução 1

Considere que  $f(x_n) = y_n$

$$\text{Área total} = \text{Trapézio}_1 + \text{Trapézio}_2 + \cdots + \text{Trapézio}_{n-1} + \text{Trapézio}_n$$

$$\text{Área total} = \frac{h}{2}[y_0 + y_1] + \frac{h}{2}[y_1 + y_2] + \frac{h}{2}[y_2 + y_3] + \cdots + \frac{h}{2}[y_{n-2} + y_{n-1}] + \frac{h}{2}[y_{n-1} + y_n]$$

$$\text{Area de trapézio} = \frac{h}{2}(B + b)$$



## Tem como simplificar a solução?