Análise de Algoritmos

Objetivos da aula:

- Identificar os problemas relacionados com a comparação computacional.
- Estimar os recursos (tempo ou memória) utilizados para a execução do algoritmo conforme o tamanho da entrada.
- Utilizar a notação $\mathcal{O}(n)$ para classificar um algoritmo conforme o seu comportamento à medida que o tamanho da entrada cresce.
- Comparar teoricamente de algoritmos

1. Compare os seguinte algoritmos:

Algorithm 1 linear_search(v, x)

Require: um vetor ordenado de tamanho n $arr[0..n-1]$ e um inteiro x
Ensure: devolve o menor inteiro inteiro k tal que $arr[k] == x$, caso contrário, devolve -1

```
prev ← 0
while arr[prev] < x do
    prev ← prev + 1
    if prev == n then
        return -1
    end if
end while
if arr[prev] == x then
    return prev
else
    return -1
end if
```

Algorithm 2 interpolation_search(v, x)

Require: um vetor ordenado de tamanho n $arr[0..n-1]$ e um inteiro x
Ensure: devolve o menor inteiro inteiro k tal que $arr[k] == x$, caso contrário, devolve -1

```
low ← 0
high ← n − 1
while arr[low]! = arr[high] and key ≥ arr[low] and key ≤ arr[high] do
    mid ← low + ((key − arr[low]) ∗ (high − low)/(arr[high] − arr[low]))
    if arr[mid] < key then
        low ← mid + 1
    else if key < arr[mid] then
        high ← mid − 1
    else
        return mid
    end if
end while
if key == arr[low] then
else
    return -1
end if
```

Algorithm 3 jump_search(v, x)

Require: um vetor ordenado de tamanho n $arr[0..n-1]$ e um inteiro x
Ensure: devolve o menor inteiro inteiro k tal que $arr[k] == x$, caso contrário, devolve -1

```
step ← √n
prev ← 0
while arr[min(step, n) − 1] < x do
    prev ← step
    step ← step + √n
    if prev ≥ n then
        return -1
    end if
end while
while arr[prev] < x do
    prev ← prev + 1
    if prev == min(step, n) then
        return -1
    end if
end while
if arr[prev] == x then
    return prev
else
    return -1
end if
```

Algorithm 4 binary_search(v, x)

Require: um vetor ordenado de tamanho n $arr[0..n-1]$ e um inteiro x
Ensure: devolve o menor inteiro inteiro k tal que $arr[k] == x$, caso contrário, devolve -1

```
res ← −1
while r ≥ l do
    mid ← l + (r − l)/2
    if arr[mid] ≥ x then
        r ← mid − 1
        if arr[mid] == x then
            res ← mid
        end if
    else
        l ← mid + 1
    end if
end while
return res
```

Primeiramente, podemos tentar fazer uma comparação computacional. Nessa comparação computacional, podemos analisar cada algoritmo e variando o tamanho da entrada:

Algoritmo × Entrada	1.000	10.000	100.000	1.000.000
linear_search	0.000009	0.000045	0.000131	0.001510
interpolation_search	0.000002	0.000004	0.000060	0.001255
jump_search	0.000002	0.000003	0.000009	0.000365
binary_search	0.000001	0.000004	0.000009	0.000295

Contudo, vários questionamentos podem ser levantados sobre a validade destes testes computacionais. Por exemplo,

- Como esses dados foram gerados?

- As entradas escolhidas podem ter favorecido algum algoritmo?
- Os dados possuem algum propriedade que pode ter sido explorada por algum algoritmo?

Precisamos de algum mecanismo para estimar os recursos utilizados pelo computador durante a execução de um algoritmo para uma entrada.

2. Estime o tempo de execução do algoritmo linear_search.

No modelo Máquina de acesso aleatório (Random Access Machine - RAM), consideramos as seguintes hipóteses:

- As instruções são executadas uma após a outra, sem operações concorrentes.
- Cada operação simples (+,-,*,/,if) demora um 1 passo.
- Cada acesso à memória custa também um passo.
- As operações realizadas com dados inteiros e ponto flutuantes tem o mesmo custo.

Algumas implicações da adoção desse modelo:

- A hierarquia de memória dos computadores é desprezada.
- A utilização de paralelismo também é desprezada.

Vamos construir uma tabela com a quantidade de operações realizada pelo algoritmo linear_search:

Operações	Número de operações
Atribuições	entre 1 e n+1
Comparação $\leq$	entre 1 e n
Comparação $==$	entre 1 e n
Acesso a um elemento do vetor	entre 1 e n
Incremento	entre 0 e n

No pior caso, o número de total de operações realizadas é $5n + 5$. No melhor caso, o número total de operações realizada é 4.

3. Estime um limite assintótico superior para o algoritmo algoritmo linear_search:

Novamente, faremos uma outra simplificação para estimativa dos recursos utilizados pelo algoritmo. Nessa simplificação, escolheremos uma função mais simples que represente um limite superior assintótico para a estimativa encontrada.

Dado duas funções positivas $f(n)$ e $g(n)$. Uma função $g(n)$ descreve um limite assintoticamente superior de uma função $f(n)$ se existem duas contantes c e n_0 tal que $cg(n) \geq f(n) \geq 0$ para todo $n \geq n_0$.

O número de operações realizadas no pior caso do algoritmo é $f(n) = 5n + 5$. Vamos mostrar que a função $g(n) = n$ é limite superior assintótico de $f(n)$. Precisamos encontrar as duas constantes c e n_0.

$$\begin{array}{lcl} 5n + 5 & \leq & 5n + n \ (n \geq 5) \\ 5n + 5 & \leq & 6n \ (n \geq 5) \end{array}$$

Logo, $c = 6$ e $n_0 = 5$

Dizemos que o algoritmo de linear_search está na classe dos algoritmos com crescimento assintótico linear.

4. Estime um limite superior assintótico os recursos utilizado pelo algoritmo binary_search:

Seja $T(n)$ o número de operações realizadas pelo algoritmo busca binária para uma entrada de tamanho n:

$$T(n) = \begin{cases} 11 & , n = 1 \\ T(n/2) + 11 & n > 1 \end{cases}$$

Vamos assumir que n é um potência de 2, ou seja, $n = 2^k$. Logo,

$$
\begin{array}{rcl}
T(2^k) & = & T(2^{k-1}) + 11 \\
 & = & T(2^{k-2}) + 11 + 11 \\
 & = & T(2^{k-3}) + 11 + 11 + 11 \\
 & \vdots & \vdots \\
 & \vdots & T(2^{k-k}) + 11k
\end{array}
$$

Logo, $T(n) = 11(k+1) = 11\log_2 n + 11$

Vamos mostrar que $g(n) = \log_2 n$ é um limite superior assintótico de $T(n)$

$$
\begin{array}{rcl}
11\log_2 n + 11 & \leq & 11\log_2 n + \log_2 n \; (n \geq 2^{11}) \\
11\log_2 n + 11 & \leq & 12\log_2 n \; (n \geq 2^{11})
\end{array}
$$

Logo, $c = 12$ e $n_0 = 2^{11}$

Dizemos que o algoritmo de busca binária está na classe dos algoritmos com crescimento assintótico logarítmico.

Como a função logarítmica tem um crescimento menor que a função linear, então o algoritmo busca_binaria é o mais indicado.

5. Estime um limite assintótico superior para o algoritmo jump_search:

   O número de operações realizadas pelo algoritmo no pior caso será $f(n) = 6\sqrt{n} + 2$.

   Vamos mostrar que $g(n) = sqrtn$ é um limite superior assintótico de $f(n)$

$$
\begin{array}{rcl}
6\sqrt{n} + 2 & \leq & 6\sqrt{n} + \sqrt{n} \; (n \geq 4) \\
6\sqrt{n} + 2 & \leq & 7\sqrt{n} \; (n \geq 4)
\end{array}
$$

   Logo, $c = 7$ e $n_0 = 4$