Lista 4 - MC458 - Gabarito
Tema: Corretude e Teorema Mestre

Lembrete: Teorema Mestre (Tim Roughgarden). Se o tempo de execução satisfaz

$$T(n) \leq a \cdot T(n/b) + O(n^d),$$

em que

- $a \geq 1$ é o número de subproblemas (chamadas recursivas);
- $b > 1$ é o fator pelo qual o tamanho do subproblema encolhe;
- $d \geq 0$ é o expoente do trabalho feito fora das chamadas recursivas (dividir e combinar),

então

$$T(n) = \begin{cases} O(n^d \log n) & \text{se } a = b^d \quad \text{(Caso 1)} \\ O(n^d) & \text{se } a < b^d \quad \text{(Caso 2)} \\ O(n^{\log_b a}) & \text{se } a > b^d \quad \text{(Caso 3)} \end{cases}$$

Interpretação: a é a taxa em que o *número de subproblemas cresce* a cada nível da recursão, e b^d é a taxa em que o *trabalho por subproblema encolhe* a cada nível. Quem "ganha" essa disputa decide o caso:

- $a = b^d$: os níveis custam todos o mesmo, e há $\log n$ níveis;
- $a < b^d$: o trabalho encolhe mais rápido do que os subproblemas se multiplicam, então o primeiro nível (a raiz) domina;
- $a > b^d$: os subproblemas se multiplicam mais rápido do que o trabalho encolhe, então o último nível (as folhas) domina.

Roteiro para aplicar: i) identifique a, b e d; ii) calcule b^d; iii) compare a com b^d; iv) escolha o caso correspondente.

1. (Problema 4.1 — Tim Roughgarden) Qual é a melhor interpretação de b^d no Método Mestre?

Resolução:
Resposta: item d).

Para entender por que, vamos olhar para o nível j da árvore de recursão (a raiz é o nível 0):

- existem a^j subproblemas;
- cada um tem tamanho n/b^j;
- o trabalho feito em cada subproblema (fora das chamadas recursivas) é $c \cdot (n/b^j)^d = \frac{c\,n^d}{(b^d)^j}$.

Ou seja, **a cada nível, o trabalho de cada subproblema é dividido por** b^d. Esta é a taxa de encolhimento do trabalho por subproblema.

Por que os outros itens estão errados:

- **a)** O trabalho total no nível j é $a^j \cdot \frac{c\,n^d}{(b^d)^j} = c\,n^d \left(\frac{a}{b^d}\right)^j$. Portanto, a taxa de crescimento do trabalho total é a/b^d, e não b^d.
- **b)** O número de subproblemas cresce por um fator a.
- **c)** O tamanho do subproblema encolhe por um fator b.

Note ainda que a expressão acima mostra por que o Teorema Mestre compara a com b^d: o trabalho total por nível é proporcional a $(a/b^d)^j$, que é constante, decrescente ou crescente conforme $a = b^d$, $a < b^d$ ou $a > b^d$.

2. (Problema 4.2 — Tim Roughgarden) Para $T(n) \leq 7 \cdot T(n/3) + O(n^2)$, qual é o menor limite superior correto?

Resolução:
Aqui $a = 7$, $b = 3$ e $d = 2$, então $b^d = 3^2 = 9$. Como $a = 7 < 9 = b^d$, estamos no Caso 2 do Teorema Mestre, logo $T(n) = O(n^d) = O(n^2)$.

Resposta: item b) $O(n^2)$**.**

Por que não os outros? Os itens c) $O(n^2 \log n)$ e d) $O(n^{2.81})$ também são limites superiores válidos, pois são maiores que n^2, mas não são os *menores* entre os corretos. Já o item a) $O(n \log n)$ é menor que n^2 e não é garantido: só o trabalho feito na raiz já é da ordem de n^2.

3. (Problema 4.3 — Tim Roughgarden) Para $T(n) \leq 9 \cdot T(n/3) + O(n^2)$, qual é o menor limite superior correto?

Resolução:
Aqui $a = 9$, $b = 3$ e $d = 2$, então $b^d = 3^2 = 9$. Como $a = 9 = b^d$, estamos no Caso 1 do Teorema Mestre, logo $T(n) = O(n^d \log n) = O(n^2 \log n)$.

Resposta: item c) $O(n^2 \log n)$**.**

Intuição: em cada nível o trabalho total é o mesmo (da ordem de n^2), e existem cerca de $\log_3 n$ níveis. Os itens b) e a) são pequenos demais, e o item d) $O(n^{3.17})$ é válido, porém maior do que o necessário.

4. (Problema 4.4 — Tim Roughgarden) Para $T(n) \leq 5 \cdot T(n/3) + O(n)$, qual é o menor limite superior correto?

Resolução:
Aqui $a = 5$, $b = 3$ e $d = 1$, então $b^d = 3^1 = 3$. Como $a = 5 > 3 = b^d$, estamos no Caso 3 do Teorema Mestre, logo $T(n) = O(n^{\log_b a}) = O(n^{\log_3 5})$.

Resposta: item c) $O(n^{\log_3 5})$**.**

Para comparar com as demais opções, note que $\log_3 5 = \frac{\ln 5}{\ln 3} \approx 1{,}46$. Então:

- a) $O(n^{\log_5 3})$ tem expoente $\approx 0{,}68 < 1$, pequeno demais (cuidado para não inverter a base e o logaritmando!);

- b) $O(n \log n)$ também é menor que $n^{1,46}$, logo não é garantido;
- d) $n^{5/3} \approx n^{1,67}$, e) n^2 e f) $n^{2,59}$ são limites válidos, mas todos maiores que $n^{1,46}$.

5. Análise de Algoritmos Reais (Karatsuba e Strassen)

Resolução:

a) **Karatsuba:** $T(n) \leq 3\,T(n/2) + O(n)$.
Aqui $a = 3$, $b = 2$, $d = 1$, e portanto $b^d = 2$. Como $a = 3 > 2 = b^d$, estamos no Caso 3:

$$T(n) = O(n^{\log_2 3}) \approx O(n^{1,585}).$$

b) **Strassen:** $T(n) \leq 7\,T(n/2) + O(n^2)$.
Aqui $a = 7$, $b = 2$, $d = 2$, e portanto $b^d = 4$. Como $a = 7 > 4 = b^d$, estamos no Caso 3:

$$T(n) = O(n^{\log_2 7}) \approx O(n^{2,807}).$$

c) **Strassen × abordagem tradicional.**
Na abordagem tradicional, temos $a = 8$, $b = 2$, $d = 2$, então $b^d = 4$. Como $a = 8 > 4 = b^d$, estamos novamente no Caso 3:

$$T(n) = O(n^{\log_2 8}) = O(n^3).$$

Em ambos os algoritmos, b e d são iguais ($b^d = 4$). A única diferença é o valor de a: o Strassen mostra como calcular o produto usando **7 multiplicações** de submatrizes de tamanho $n/2$ em vez de 8. Como $a > b^d$ nos dois casos, quem domina o tempo são as folhas da recursão, e a quantidade de folhas é $a^{\log_b n} = n^{\log_b a}$. Logo, **reduzir a de 8 para 7 reduz o expoente** de $\log_2 8 = 3$ para $\log_2 7 \approx 2{,}81$.

O "preço" dessa troca são mais somas e subtrações de matrizes, mas elas custam $O(n^2)$ e já estão incluídas em $d = 2$. Como $7 > 4$, esse trabalho extra não muda o termo dominante.

6. Subvetor de Soma Máxima (Divisão e Conquista)

Resolução:
Convenção e notação. Escrevemos $\text{soma}(i, j) = A[i] + A[i+1] + \cdots + A[j]$. Repare que o caso base do algoritmo devolve $\max(0, A[\text{low}])$, ou seja, o algoritmo aceita o subvetor *vazio*, cuja soma vale 0. Assim, o valor que queremos provar que o algoritmo devolve é

$$S^*(\text{low}, \text{high}) = \max\Big\{\, 0,\ \max_{\text{low} \leq i \leq j \leq \text{high}} \text{soma}(i, j) \Big\}.$$

Quando o vetor tem algum elemento positivo, S^* é exatamente a maior soma contínua. Quando todos os elementos são negativos, $S^* = 0$.

a) **Corretude.**

i) **Corretude de `MaxCruzado`.**

Objetivo. Vamos provar que `MaxCruzado`$(A, \text{low}, mid, \text{high})$ devolve

$$X = \max_{\text{low} \le i \le mid < j \le \text{high}} \text{soma}(i, j),$$

isto é, a maior soma entre todos os subvetores $A[i \dots j]$ que cruzam mid. Este é o resultado que será usado no item (ii).

Passo 1 — Separar a soma em duas partes independentes.
Um subvetor que cruza o meio é $A[i \dots j]$ com $\text{low} \le i \le mid < j \le \text{high}$. Sua soma é

$$\text{soma}(i, j) = \text{soma}(i, mid) + \text{soma}(mid + 1, j).$$

A primeira parcela depende só de i e a segunda só de j, e podemos escolher i e j de forma independente. Logo, o maior valor possível dessa soma é

$$X = \max_{i,j} \text{soma}(i, j) = \underbrace{\max_{\text{low} \le i \le mid} \text{soma}(i, mid)}_{\text{melhor lado esquerdo}} + \underbrace{\max_{mid < j \le \text{high}} \text{soma}(mid + 1, j)}_{\text{melhor lado direito}}.$$

Passo 2 — O primeiro laço calcula o melhor lado esquerdo.

(a) O que o laço faz, em palavras. O índice i começa em mid e vai andando para a esquerda até low. A variável `soma` acumula os elementos já visitados, ou seja, a cada momento ela é a soma do trecho que vai de i até mid. Já `soma_esq` guarda a *maior* dessas somas vistas até agora. Em outras palavras, o laço testa todos os subvetores que terminam em mid, $A[mid \dots mid]$, $A[mid - 1 \dots mid]$, $A[mid - 2 \dots mid]$ e assim por diante, e fica com o melhor.

(b) Exemplo. Considere $A = [-2,\ 3,\ -1,\ 4,\ -5,\ 2]$, com $\text{low} = 1$, $mid = 3$ e $\text{high} = 6$. O primeiro laço funciona assim:

i	$A[i]$	`soma`	`soma_esq`	trecho testado
(antes do laço)	—	0	$-\infty$	—
3	-1	-1	-1	$A[3 \dots 3]$
2	3	2	2	$A[2 \dots 3]$
1	-2	0	2	$A[1 \dots 3]$

(Os valores de `soma` e `soma_esq` na tabela são os de *depois* da iteração, já com $A[i]$ somado e o teste feito.) No final, `soma_esq` $= 2$, que é de fato a maior entre as somas -1, 2 e 0. Repare que, na última linha, $0 > 2$ é falso, então `soma_esq` não muda.

(c) Prova com invariante. Para provar que isso vale para qualquer vetor, usamos um *invariante de laço*: uma afirmação que é verdadeira antes de cada iteração. Para cada i de mid até low, o invariante $I(i)$ é:

Imediatamente antes da iteração com índice i, temos

$$\texttt{soma} = \text{soma}(i + 1, mid)$$
$$\texttt{soma_esq} = \max_{i < i' \le mid} \text{soma}(i', mid),$$

em que $\text{soma}(i + 1, mid)$ *vale* 0 *e o máximo vale* $-\infty$ *quando* $i = mid$ *(não há elementos, e o máximo de um conjunto vazio é* $-\infty$*).*

Em palavras: antes de olhar para $A[i]$, `soma` já guarda a soma do trecho $A[i+1 \ldots mid]$, e `soma_esq` já guarda a melhor soma entre os trechos que começam em $i+1$ ou mais à direita. Precisamos checar três coisas:

- **Inicialização (antes da primeira iteração, $i = mid$):** o código faz `soma` $= 0$ e `soma_esq` $= -\infty$. Como soma$(mid+1, mid)$ é uma soma de zero elementos, que vale 0, e o máximo sobre um conjunto vazio é $-\infty$, o invariante $I(mid)$ é verdadeiro.

- **Manutenção (se $I(i)$ vale, então $I(i-1)$ vale depois da iteração):** suponha $I(i)$ verdadeiro. A iteração executa duas linhas:

  A. `soma` $\leftarrow$ `soma` $+ A[i]$. Pelo invariante, o novo valor é soma$(i+1, mid) + A[i] =$ soma(i, mid).

  B. Se `soma` $>$ `soma_esq`, então `soma_esq` $\leftarrow$ `soma`. Isso é o mesmo que `soma_esq` $\leftarrow \max($`soma_esq`, soma$(i, mid))$. Pelo invariante, o valor antigo de `soma_esq` é $\max_{i<i' \le mid}$ soma(i', mid), então o novo valor é

  $$\max\Big(\max_{i<i' \le mid} \text{soma}(i', mid),\ \text{soma}(i, mid)\Big) = \max_{i \le i' \le mid} \text{soma}(i', mid).$$

  Depois da iteração, o próximo valor do índice é $i-1$, e temos `soma` $=$ soma$((i-1)+1, mid)$ e `soma_esq` $= \max_{i-1<i' \le mid}$ soma(i', mid). Isso é exatamente $I(i-1)$.

- **Término:** o laço termina depois da iteração $i =$ low, e nesse momento vale $I(\text{low}-1)$, isto é,

  $$\texttt{soma_esq} = \max_{\text{low} \le i' \le mid} \text{soma}(i', mid).$$

  Esse é o "melhor lado esquerdo" do Passo 1.

(d) Por que $-\infty$ e não 0? Um subvetor que cruza o meio precisa conter pelo menos $A[mid]$ (do lado esquerdo) e pelo menos $A[mid+1]$ (do lado direito). Se `soma_esq` começasse em 0, o laço poderia "escolher" um lado esquerdo vazio quando todas as somas fossem negativas, e o resultado deixaria de ser um subvetor que cruza o meio. Começando em $-\infty$, o primeiro teste `soma` $> -\infty$ é sempre verdadeiro, então pelo menos o trecho $A[mid \ldots mid]$ sempre é considerado, mesmo que ele seja negativo.

Passo 3 — O segundo laço calcula o melhor lado direito.

O argumento é o mesmo, espelhado: agora j anda de $mid+1$ até high, e `soma` acumula o trecho que vai de $mid+1$ até j. O invariante $J(j)$ é:

Imediatamente antes da iteração com índice j, temos

$$\texttt{soma} = \text{soma}(mid+1, j-1)$$
$$\texttt{soma_dir} = \max_{mid<j'<j} \text{soma}(mid+1, j'),$$

com soma$(mid+1, mid) = 0$ *e máximo igual a* $-\infty$ *quando* $j = mid+1$.

- **Inicialização ($j = mid+1$):** `soma` $= 0$ e `soma_dir` $= -\infty$, e isso coincide com $J(mid+1)$.

- **Manutenção:** a iteração faz `soma` $\leftarrow$ soma$(mid+1, j-1) + A[j]$ = soma$(mid+1, j)$ e depois `soma_dir` $\leftarrow \max($`soma_dir`, soma$(mid+1, j))$, o que dá `soma_dir` $= \max_{mid<j'\le j}$ soma$(mid+1, j')$. Isso é $J(j+1)$.
- **Término:** depois da iteração $j =$ high vale $J(\text{high}+1)$, ou seja,

$$\texttt{soma_dir} = \max_{mid<j'\le \text{high}} \text{soma}(mid+1, j').$$

No exemplo anterior, o segundo laço dá:

j	$A[j]$	`soma`	`soma_dir`	trecho testado
(antes do laço)	—	0	$-\infty$	—
4	4	4	4	$A[4\ldots4]$
5	-5	-1	4	$A[4\ldots5]$
6	2	1	4	$A[4\ldots6]$

Então `soma_dir` $= 4$.

Conclusão do item (i). A função devolve `soma_esq`+`soma_dir`. Pelos Passos 2 e 3, isso é (melhor lado esquerdo) + (melhor lado direito) e, pelo Passo 1, essa soma é

$$X = \max_{\text{low}\le i\le mid<j\le \text{high}} \text{soma}(i, j).$$

No exemplo, $2+4=6$, que corresponde ao subvetor $A[2\ldots4] = [3, -1, 4]$. □

ii) **Corretude de `MaxSubarray` por indução em $n = \text{high} - \text{low} + 1$.**

Tipo de indução. Usamos *indução forte*: na hipótese de indução, assumimos que a afirmação vale para *todos* os tamanhos menores que n, e não só para $n-1$. Isso é necessário porque as chamadas recursivas são feitas sobre as duas metades do vetor, que não têm tamanho $n-1$.

Afirmação $P(n)$. Para todo vetor A e todos low, high com high $-$ low $+1 = n$, a chamada `MaxSubarray`$(A, \text{low}, \text{high})$ devolve $S^*(\text{low}, \text{high})$.

Base ($n = 1$).

Aqui low = high. Os únicos subvetores possíveis são o vazio (soma 0) e $A[\text{low}]$. Logo, $S^*(\text{low}, \text{low}) = \max(0, A[\text{low}])$, que é exatamente o que o algoritmo devolve na linha do caso base.

Passo indutivo ($n \ge 2$).

Seja $n \ge 2$ e fixe low e high com high $-$ low $+1 = n$.

Hipótese de indução (HI): $P(m)$ é verdadeira para todo m com $1 \le m < n$. Ou seja, qualquer chamada de `MaxSubarray` sobre um vetor de tamanho menor que n devolve o valor correto S^*.

Tamanho das metades.

Como low $<$ high, temos low $\le mid <$ high. Então as duas metades são não vazias, e seus tamanhos são

$$n_E = mid - \text{low} + 1 \ge 1 \qquad \text{e} \qquad n_D = \text{high} - mid \ge 1, \qquad \text{com } n_E + n_D = n.$$

Como os dois são pelo menos 1 e somam n, cada um é no máximo $n-1$. Logo, $n_E < n$ e $n_D < n$.

O que a HI nos dá.

Como $n_E < n$ e $n_D < n$, a HI se aplica às duas chamadas recursivas (uma sobre $A[\text{low} \ldots mid]$ e outra sobre $A[mid + 1 \ldots \text{high}]$). Ou seja, *assumimos* que essas instâncias menores estão corretas, e portanto:

$$\begin{aligned} \texttt{max_esq} &= S^*(\text{low}, mid) && \text{(HI com } m = n_E\text{)} \\ \texttt{max_dir} &= S^*(mid + 1, \text{high}) && \text{(HI com } m = n_D\text{)} \end{aligned}$$

Para escrever isso de forma mais conveniente, sejam B_E e B_D as maiores somas de subvetores *não vazios* de cada metade:

$$B_E = \max_{\text{low} \le i \le j \le mid} \text{soma}(i, j) \qquad \text{e} \qquad B_D = \max_{mid < i \le j \le \text{high}} \text{soma}(i, j).$$

Pela definição de S^*, temos $S^*(\text{low}, mid) = \max(0, B_E)$ e $S^*(mid + 1, \text{high}) = \max(0, B_D)$. Portanto,

$$\texttt{max_esq} = \max(0, B_E) \qquad \text{e} \qquad \texttt{max_dir} = \max(0, B_D).$$

O que o item (i) nos dá.

O item (i) provou que `MaxCruzado` devolve a maior soma entre os subvetores que cruzam mid. Ou seja,

$$\texttt{max_cruz} = X = \max_{\text{low} \le i \le mid < j \le \text{high}} \text{soma}(i, j).$$

Note que esta etapa não usa a HI: o item (i) vale para qualquer valores de low, mid e high.

Todo subvetor não vazio está em um (e só um) de três casos.

Dado $A[i \ldots j]$ com $\text{low} \le i \le j \le \text{high}$, vale exatamente uma das opções:

- $j \le mid$: o subvetor está inteiramente na metade esquerda (e sua soma é uma das consideradas em B_E);
- $i > mid$: o subvetor está inteiramente na metade direita (e sua soma é uma das consideradas em B_D);
- $i \le mid < j$: o subvetor cruza o meio (e sua soma é uma das consideradas em X).

Logo, a maior soma B entre todos os subvetores não vazios de $A[\text{low} \ldots \text{high}]$ é

$$B = \max(B_E,\ B_D,\ X).$$

Conclusão.

O algoritmo devolve $R = \max(\texttt{max_esq}, \texttt{max_dir}, \texttt{max_cruz})$. Então:

$$\begin{aligned} R &= \max\big(\max(0, B_E),\ \max(0, B_D),\ X\big) && \text{(HI e item (i))} \\ &= \max(0,\ B_E,\ B_D,\ X) && \text{(juntando os dois 0)} \\ &= \max(0,\ B) && \text{(três casos)} \\ &= S^*(\text{low}, \text{high}). \end{aligned}$$

Portanto, $P(n)$ vale. Pelo princípio da indução forte, $P(n)$ vale para todo $n \ge 1$. □

b) **Complexidade.**

Trabalho de uma chamada. A cada chamada com vetor de tamanho n, o algoritmo faz:

- duas chamadas recursivas sobre vetores de tamanho (aproximadamente) $n/2$;
- uma chamada a `MaxCruzado`, cujos dois laços juntos percorrem os n elementos de $A[\text{low} \dots \text{high}]$, gastando $O(n)$;
- algumas comparações e atribuições, em tempo $O(1)$.

Recorrência.

$$T(n) \leq 2\,T(n/2) + O(n).$$

Parâmetros e conclusão. Temos $a = 2$, $b = 2$ e $d = 1$, então $b^d = 2$. Como $a = b^d$, estamos no Caso 1:

$$T(n) = O(n^d \log n) = O(n \log n).$$

7. Busca por Elemento Majoritário

Resolução:

a) **Propriedade essencial.**

Afirmação. Se x é majoritário em $A[1 \dots n]$, então x é majoritário na metade esquerda ou na metade direita.

Ideia. Se x "perdesse" (ou empatasse) nas duas metades, ele não poderia ganhar no vetor inteiro. Vamos formalizar isso por contradição.

Notação. Sejam n_E e n_D os tamanhos das metades esquerda e direita (assim $n_E + n_D = n$), e sejam f_E e f_D as frequências de x em cada metade. A frequência de x no vetor inteiro é $f = f_E + f_D$.

Prova. Suponha, por absurdo, que x é majoritário em $A[1 \dots n]$, mas *não* é majoritário em nenhuma das duas metades. Como "majoritário" significa frequência estritamente maior que metade do tamanho, não ser majoritário significa frequência menor ou igual à metade do tamanho:

$$f_E \leq \frac{n_E}{2} \qquad \text{e} \qquad f_D \leq \frac{n_D}{2}.$$

Somando as duas desigualdades,

$$f = f_E + f_D \leq \frac{n_E + n_D}{2} = \frac{n}{2}.$$

Isso contradiz o fato de x ser majoritário em $A[1 \dots n]$, que exige $f > n/2$. Portanto, x é majoritário em pelo menos uma das metades. □

Por que isso garante a corretude do algoritmo? Pela propriedade, se existe um elemento majoritário no vetor inteiro, ele só pode ser o candidato devolvido pela metade esquerda ou pela metade direita. Basta, então, testar esses dois candidatos contando suas ocorrências (é o que fazem as duas chamadas a `Contar`). Se os dois candidatos forem iguais, esse valor é majoritário nas duas metades e, somando as frequências, também no vetor inteiro.

b) **Complexidade (pior caso).**

Pior caso. O pior caso acontece quando os candidatos são diferentes, de modo que as duas chamadas a `Contar` são executadas.

Trabalho de uma chamada. Cada chamada à função faz:

- duas chamadas recursivas sobre metades do vetor;
- duas chamadas a `Contar`, cada uma em $O(n)$, o que dá $O(n)$ no total;
- comparações em tempo $O(1)$.

Recorrência.

$$T(n) \leq 2\,T(n/2) + O(n).$$

Parâmetros e conclusão. Temos $a = 2$, $b = 2$ e $d = 1$, então $b^d = 2 = a$. Estamos no Caso 1:

$$T(n) = O(n \log n).$$

Interpretação da relação entre a e b^d. A cada nível, o número de subproblemas dobra ($a = 2$), mas o trabalho de cada um cai pela metade ($b^d = 2$). Esses dois efeitos se compensam exatamente, e todos os níveis custam cerca de n no total. Como há $\log_2 n$ níveis, o custo final é $O(n \log n)$.

8. Localização de Pico em Vetor (Peak Finding)

Resolução:

a) **Corretude.**

Lembre que um *pico* é um índice i com $A[i] \geq A[i-1]$ e $A[i] \geq A[i+1]$, com $A[0] = A[n+1] = -\infty$. Note que os vizinhos de i são sempre os do vetor A *inteiro*, mesmo que i esteja numa extremidade do subvetor $A[\text{low} \ldots \text{high}]$.

Ideia: o algoritmo sempre caminha para o lado em que o vetor "sobe". Quando ele entra num subvetor, o vetor está subindo na entrada (ou estamos no início) e descendo na saída (ou estamos no fim); então, em algum ponto do meio, tem que existir um topo. Para formalizar, usamos o seguinte invariante.

Invariante (I) sobre uma chamada $(\textbf{low}, \textbf{high})$:

$$A[\text{low} - 1] < A[\text{low}] \quad \text{e} \quad A[\text{high}] > A[\text{high} + 1].$$

(Se low $= 1$ ou high $= n$, a desigualdade correspondente vale automaticamente por causa de $A[0] = A[n+1] = -\infty$.) A chamada inicial $(1, n)$ satisfaz (I).

Afirmação $P(n)$: toda chamada `EncontraPico`$(A, \text{low}, \text{high})$ que satisfaz (I) e tem high $-$ low $+ 1 = n$ termina e devolve um índice em $[\text{low}, \text{high}]$ que é pico de A. Em particular, $A[\text{low} \ldots \text{high}]$ contém um pico.

Provamos por indução forte em $n \geq 1$.

Observação preliminar (as chamadas recursivas são válidas). Suponha que (I) vale.

- Se $mid =$ high, então $A[mid] > A[mid+1]$ por (I), e o primeiro teste ($A[mid] < A[mid+1]$) falha. Logo, se ele for verdadeiro, então $mid <$ high e a chamada $(mid + 1, \text{high})$ é um intervalo não vazio.

- Se $mid = \text{low}$, então $A[mid] > A[mid-1]$ por (I), e o segundo teste ($A[mid] < A[mid-1]$) falha. Logo, se ele for verdadeiro, então $mid > \text{low}$ e a chamada $(\text{low}, mid-1)$ é um intervalo não vazio.
- **Base ($n = 1$):** temos $\text{low} = \text{high} = mid$. Por (I), $A[mid-1] < A[mid]$ e $A[mid] > A[mid+1]$. Os dois testes falham, e o algoritmo devolve mid, que é pico pois $A[mid] \geq A[mid-1]$ e $A[mid] \geq A[mid+1]$.
- **Passo indutivo:** seja $n \geq 2$ e assuma como hipótese de indução (HI) que $P(m)$ vale para todo $1 \leq m < n$. Há três casos:
  Caso 1: $A[mid] < A[mid+1]$. O algoritmo chama $(mid+1, \text{high})$.
  - O tamanho é $\text{high} - mid \leq n-1$, pois $mid \geq \text{low}$.
  - O invariante vale na nova chamada: $A[(mid+1)-1] = A[mid] < A[mid+1]$, e o lado direito (com high) não mudou.
  - Pela HI, a chamada devolve um pico de A.

  Caso 2: $A[mid] \geq A[mid+1]$ e $A[mid] < A[mid-1]$. O algoritmo chama $(\text{low}, mid-1)$.
  - O tamanho é $mid - \text{low} \leq n-1$, pois $mid \leq \text{high}$.
  - O invariante vale na nova chamada: o lado esquerdo (com low) não mudou, e $A[mid-1] > A[mid] = A[(mid-1)+1]$.
  - Pela HI, a chamada devolve um pico de A.

  Caso 3: $A[mid] \geq A[mid+1]$ e $A[mid] \geq A[mid-1]$. O algoritmo devolve mid, que é pico pela própria definição.

Nos três casos o algoritmo devolve um pico de A. Logo, $P(n)$ vale para todo $n \geq 1$. Como a chamada inicial $(1, n)$ satisfaz (I), o algoritmo sempre termina devolvendo um pico válido. □

b) **Complexidade.**

A cada chamada, o algoritmo faz um número constante de comparações, $O(1)$, e no máximo *uma* chamada recursiva. O subvetor da chamada recursiva tem tamanho $\lfloor n/2 \rfloor$ (lado direito) ou $\lceil n/2 \rceil - 1$ (lado esquerdo), ou seja, no máximo $n/2$. Logo,

$$T(n) \leq T(n/2) + O(1).$$

Os parâmetros são $a = 1$, $b = 2$ e $d = 0$ (pois $O(1) = O(n^0)$), então $b^d = 2^0 = 1 = a$. Estamos no Caso 1:

$$T(n) = O(n^0 \log n) = O(\log n).$$

Interpretação: há um único subproblema por nível ($a = 1$) e o trabalho por subproblema não encolhe ($b^d = 1$). Como cada nível custa $O(1)$ e o vetor é dividido pela metade a cada nível, há $\log_2 n$ níveis. É o mesmo comportamento da busca binária.