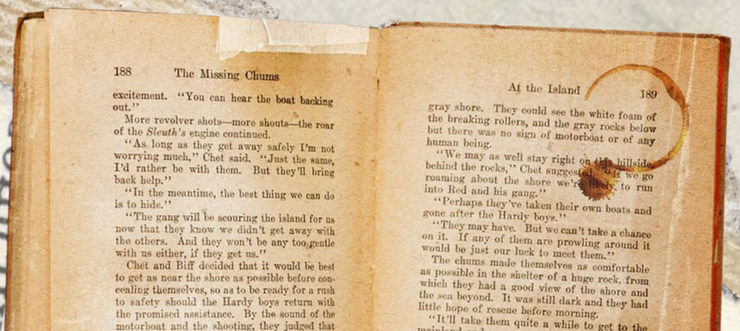
Teoria da Computação

Pérolas de Programação

Por Davi André,
Caio Alves e
Anderson Santos


188 The Missing Chums

excitement. "You can hear the boat backing out."

More revolver shots—more shouts—the roar of the *Sleuth's* engine continued.

"As long as they get away safely I'm not worrying much," Chet said. "Just the same, I'd rather be with them. But they'll bring back help."

"In the meantime, the best thing we can do is to hide."

"The gang will be scouring the island for us now that they know we didn't get away with the others. And they won't be any too gentle with us either, if they get us."

Chet and Biff decided that it would be best to get as near the shore as possible before concealing themselves, so as to be ready for a rush to safety should the Hardy boys return with the promised assistance. By the sound of the motorboat and the shooting, they judged that

At the Island 189

gray shore. They could see the white foam of the breaking rollers, and the gray rocks below but there was no sign of motorboat or of any human being.

"We may as well stay right on the hillside behind the rocks," Chet suggested. "If we go roaming about the shore we're likely to run into Red and his gang."

"Perhaps they've taken their own boats and gone after the Hardy boys."

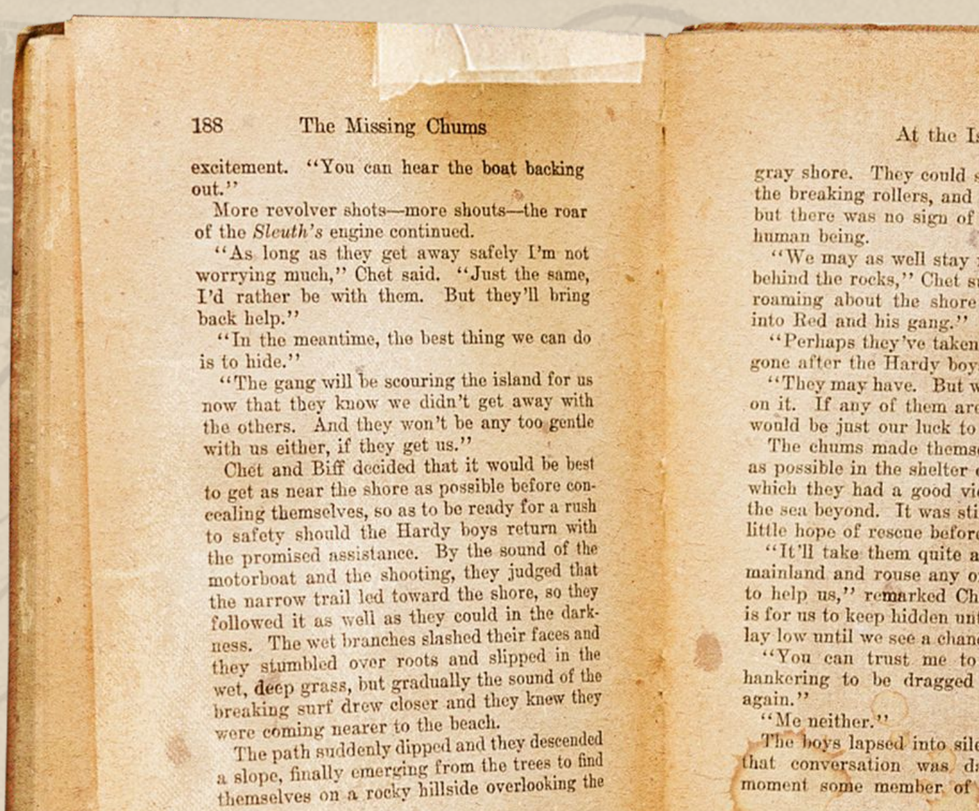
"They may have. But we can't take a chance on it. If any of them are prowling around it would be just our luck to meet them."

The chums made themselves as comfortable as possible in the shelter of a huge rock, from which they had a good view of the shore and the sea beyond. It was still dark and they had little hope of rescue before morning.

"It'll take them quite a while to get to the

Column 11: Sorting

- Ideia principal: pequenas melhorias podem transformar um algoritmo
- Bentley não ensina apenas a ordenar, mas como pensar como um engenheiro de software.
- A ordenação é uma das operações mais comuns em computação.



188 The Missing Chums

excitement. "You can hear the boat backing out."

More revolver shots—more shouts—the roar of the *Sleuth's* engine continued.

"As long as they get away safely I'm not worrying much," Chet said. "Just the same, I'd rather be with them. But they'll bring back help."

"In the meantime, the best thing we can do is to hide."

"The gang will be scouring the island for us now that they know we didn't get away with the others. And they won't be any too gentle with us either, if they get us."

Chet and Biff decided that it would be best to get as near the shore as possible before concealing themselves, so as to be ready for a rush to safety should the Hardy boys return with the promised assistance. By the sound of the motorboat and the shooting, they judged that the narrow trail led toward the shore, so they followed it as well as they could in the darkness. The wet branches slashed their faces and they stumbled over roots and slipped in the wet, deep grass, but gradually the sound of the breaking surf drew closer and they knew they were coming nearer to the beach.

The path suddenly dipped and they descended a slope, finally emerging from the trees to find themselves on a rocky hillside overlooking the

At the I

gray shore. They could s
the breaking rollers, and
but there was no sign of
human being.

"We may as well stay
behind the rocks," Chet s
roaming about the shore
into Red and his gang."

"Perhaps they've taken
gone after the Hardy boy

"They may have. But
on it. If any of them ar
would be just our luck to

The chums made thems
as possible in the shelter
which they had a good vi
the sea beyond. It was sti
little hope of rescue before

"It'll take them quite a
mainland and rouse any o
to help us," remarked Ch
is for us to keep hidden un
lay low until we see a chan

"You can trust me to
hankering to be dragged
again."

"Me neither."

The boys lapsed into sil
that conversation was d
moment some member of

Caso 11.1 - Ideia do Algoritmo

- O Insertion Sort é inspirado na forma como jogadores de cartas organizam suas cartas na mão..
- O algoritmo mantém uma parte do vetor sempre ordenada.
- A cada passo, um novo elemento é inserido na posição correta dentro da parte já ordenada.

Exemplo de execução:

3 | 1 4 2
1 3 | 4 2
1 3 4 | 2
1 2 3 4 |

A barra indica o elemento que está sendo inserido na parte ordenada.

Caso 11.1 - Primeira Implementação

Após explicar a ideia do algoritmo, Bentley apresenta
uma implementação simples do Insertion Sort.

- Percorre o vetor da esquerda para a direita
- Compara o elemento atual com os anteriores
- Realiza trocas até encontrar a posição correta

```
for i = [1, n)
 for (j = i; j > 0 && x[j-1] > x[j]; j--)
 swap(j-1, j)
```

Essa versão é simples, mas não é a mais eficiente.
Bentley usa esse código como ponto de partida para otimizações.

Caso 11.1 - Otimização do swap()

A primeira implementação utiliza a função swap() para
trocar elementos durante a ordenação.

```
swap(a, b)
 temp = a
 a = b
 b = temp
```

Bentley observa que chamadas de função podem ser custosas.

Melhoria proposta: Substituir a chamada de swap() por atribuições diretas.

Realiza trocas até encontrar a posição correta

```
t = x[j]
x[j] = x[j-1]
x[j-1] = t
```

Essa alteração reduz o custo da execução do algoritmo.
Bentley observa que essa mudança pode reduzir o tempo
para cerca de 1/3 da versão original.

Caso 11.1 Insertion Sort — Versão Otimizada

Bentley propõe uma versão mais eficiente do algoritmo.

Em vez de trocar elementos repetidamente, os valores maiores são deslocados para a direita até encontrar a posição correta para o elemento atual.

Realiza trocas até encontrar a posição correta

```
for i = [1, n)
    t = x[i]
    for (j = i; j > 0 && x[j-1] > t; j--)
        x[j] = x[j-1]
    x[j] = t
```

Essa versão reduz o número de operações e melhora o desempenho do algoritmo.

Caso 11.2 — Quicksort: Motivação

O Insertion Sort é simples e eficiente para listas pequenas, mas seu desempenho é limitado para grandes conjuntos de dados.

Complexidade do Insertion Sort:
• $O(n^2)$

Para grandes volumes de dados, algoritmos mais eficientes são necessários.

Uma solução amplamente utilizada é o Quicksort.

Quicksort utiliza a estratégia de "dividir e conquistar"
para ordenar dados de forma mais eficiente.

Caso 11.2 — Quicksort: Particionamento do Vetor

A ideia central do Quicksort é reorganizar o vetor em torno de um elemento chamado pivô.

Durante o particionamento:

- elementos menores que o pivô ficam à esquerda;
- elementos maiores que o pivô ficam à direita;

Realiza trocas até encontrar a posição correta

Exemplo:

Vetor original:
8-3-1-7-0-10-2

Escolhendo pivô = 7

Após o particionamento:
3-1-0-2-| 7 |-8-10

Após o particionamento, o pivô já está na posição correta.

Caso 11.2 — Quicksort: Comparação

Comparação com Insertion Sort:

O Insertion Sort coloca um elemento na posição correta por vez, já o Quicksort, ao fazer o particionamento, organiza muitos elementos de uma vez.

Resultado:

- Insertion Sort: $O(n^2)$
- Quicksort [médio]: $O(n \log n)$

Um único particionamento já deixa metade dos elementos de cada lado do pivô.

Caso 11.3 — Aprimorando o Quicksort

- A implementação básica do Quicksort funciona bem na maioria dos casos.

- Porém, em algumas entradas o desempenho pode cair para: $O(n^2)$

- Isso acontece quando:
  - as partições ficam muito desbalanceadas
  - o pivô escolhido é ruim

O objetivo desta seção é tornar o Quicksort mais robusto e eficiente na prática.

Caso 11.3 — Quicksort: Melhor Escolha do Pivô

Escolher sempre o primeiro elemento como pivô pode gerar partições ruins.

Exemplo:

- vetor já ordenado
- vetor quase ordenado

Isso pode fazer o Quicksort cair para $O(n^2)$.

Solução:
Escolher um pivô mais representativo
do conjunto de dados;
Estratégia comum:

- primeiro elemento
- elemento do meio
- último elemento

Caso 11.3 — Quicksort: A Mediana de Três

Como garantir um bom pivô e evitar o pior caso.

O Problema:
Escolher apenas o primeiro ou o último elemento como pivô ainda é arriscado. Se o vetor já estiver quase ordenado, o desempenho cai para $O(n^2)$.

A Solução de Bentley:
A técnica da Mediana de Três.

Como funciona:

- Analisamos três elementos: o primeiro, o do meio e o último.
- Ordenamos esses três e escolhemos o valor central (a mediana) como pivô.

Por que funciona?
Garante que o pivô nunca será o maior nem o menor elemento do trecho.

- Força partições mais balanceadas na grande maioria dos casos práticos.

Caso 11.3 — Quicksort: O Problema das Chaves Iguais

O que acontece quando o vetor tem muitos elementos idênticos?

A Armadilha:
Se todos os elementos do vetor forem iguais (ex: 2, 2, 2), o particionamento básico falha. Ele joga todos os elementos para um único lado, gerando o pior caso: $O(n^2)$.

A Solução Prática:
Alterar os ponteiros de varredura (esquerda e direita) para que eles parem ao encontrar um elemento igual ao pivô.

O Resultado:
Isso força uma troca desnecessária no momento, mas garante que os elementos iguais sejam divididos igualmente entre as duas metades (esquerda e direita).

Impacto:
Mantém a árvore de recursão balanceada e o tempo em $O(n \log n)$, sendo essencial para colocar o Quicksort em produção.

Caso 11.3 — Quicksort: Subarrays Pequenos

Quando os subarrays ficam muito pequenos, o Quicksort deixa de ser eficiente.

Isso acontece porque:

- a recursão gera overhead
- o custo das chamadas de função aumenta

Solução:

Usar Insertion Sort para ordenar subarrays pequenos.

Combinar algoritmos pode melhorar o desempenho.

Caso 11.3 — Quicksort: Reduzindo o Custo da Recursão

Para melhorar uso de memória e melhorar implementação prática; Contexto da época

O Quicksort usa recursão, o que pode gerar:

- muitas chamadas de função
- maior uso da pilha (stack)

Melhoria:

- Ordenar primeiro o menor subarray.
- Depois continuar com o maior subarray sem nova recursão.

Os Princípios

Use implementações prontas

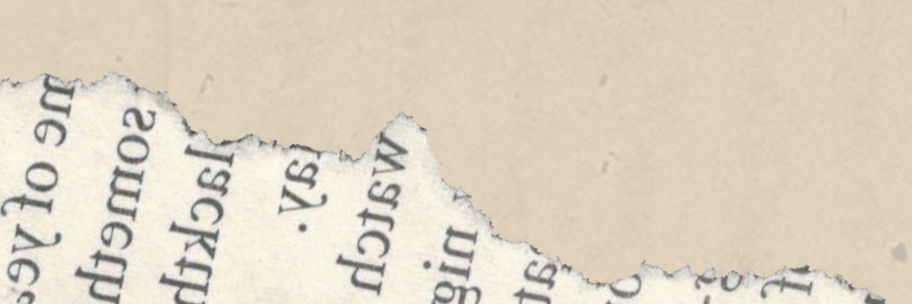
Sempre que possível, utilize funções de ordenação já disponíveis nas bibliotecas da linguagem. Elas são bem testadas, eficientes e normalmente possuem implementações altamente otimizadas.

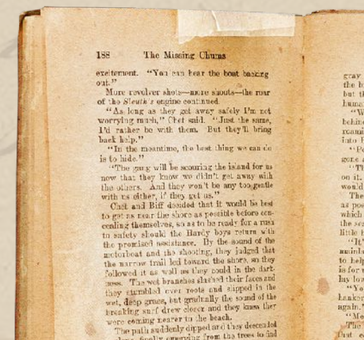
Algoritmos simples para problemas pequenos

Algoritmos como Insertion Sort são fáceis de implementar e podem apresentar bom desempenho quando aplicados a conjuntos pequenos de dados.

Algoritmo + otimização

Para grandes volumes de dados, algoritmos com complexidade O(n log n), como o Quicksort, são essenciais. Além disso, pequenas otimizações no código podem gerar melhorias significativas de desempenho.





Obrigado