Quick Sort

Considerações Sobre Algoritmos de Ordenação

Estagiário PAE: Jesimar da S. Arantes
Professor: Claudio F. M. Toledo

27 de Setembro de 2017

História

- **Quicksort** é um algoritmo recursivo desenvolvido por **Charles Antony Richard Hoare**.
- C.A.R. Hoare nasceu em Colombo, no Sri Lanka, filho de pais britânicos.
- Graduou-se na Universidade de Oxford em 1956.
- Estudou tradução computacional de linguagens humanas em visita à Universidade de Moscou, URSS.
  - Durante os estudos, foi preciso realizar a ordenação de palavras a serem traduzidas.
  - O quicksort foi o algoritmo desenvolvido por Hoare para ordenar as palavras, em 1960, aos 26 anos.

História: C.A.R. Hoare

- Recebeu o Prêmio Turing da ACM de 1980, por "suas contribuições fundamentais para a definição e projeto de linguagens de programação".
- Em 2009, desculpou-se por ter inventado a referência nula.
- É atualmente pesquisador sênior da Microsoft Research em Cambridge, Inglaterra e professor emérito da Universidade de Oxford.

Características

- **Quicksort** é um algoritmo recursivo que utiliza a estratégia da **divisão e conquista**.
- Considerada a mais **rápida** ordenação **baseada em comparações** sobre arranjos.
  - Na prática, se bem implementado, executa quase sempre em $\Theta(n \ lg \ n)$.
  - No pior caso pode executar em tempo $\Theta(n^2)$.
- **Não-estável**.

Método Estável e Não-Estável

Ordenação	
Nome	Sobrenome
Alex	Vieira
André	Tavares
Antônio	Vieira
Eduardo	Molina
Estevão	Lobo
Felipe	Matos
Felipe	Veloso
Glauco	Costa
Guilherme	Coelho
Júlio	Bueno
Leandro	Silva
Lívia	Fares
Lucas	Carlos
Lucas	Fukushima
Lucas	Tavares
Miguel	Filho
Rafael	Silva
Rodolfo	Cassimiro
Rodolfo	Santos
Vinícius	Ribeiro

	Ordenação Estável
Nome	Sobrenome
Júlio	Bueno
Lucas	Carlos
Rodolfo	Cassimiro
Guilherme	Coelho
Glauco	Costa
Lívia	Fares
Miguel	Filho
Lucas	Fukushima
Estevão	Lobo
Felipe	Matos
Eduardo	Molina
Vinícius	Ribeiro
Rodolfo	Santos
Leandro	Silva
Rafael	Silva
André	Tavares
Lucas	Tavares
Felipe	Veloso
Alex	Vieira
Antônio	Vieira

	Ordenação Não-Estável
Nome	Sobrenome
Júlio	Bueno
Lucas	Carlos
Rodolfo	Cassimiro
Guilherme	Coelho
Glauco	Costa
Lívia	Fares
Miguel	Filho
Lucas	Fukushima
Estevão	Lobo
Felipe	Matos
Eduardo	Molina
Vinícius	Ribeiro
Rodolfo	Santos
Rafael	Silva
Leandro	Silva
André	Tavares
Lucas	Tavares
Felipe	Veloso
Antônio	Vieira
Alex	Vieira

Figura: Exemplo de método estável e não-estável.

Estabilidade dos Métodos

- Métodos Estáveis: Merge sort, Insertion sort, Bubble sort, Bucket sort, Counting sort e Radix sort.
- Métodos Não-Estáveis: Quicksort, Heapsort, Selection sort e Shell sort.
- Curiosidade: cerca de metade os métodos existentes são estáveis e metade são não-estáveis. Na wikipédia tem-se descrito um conjunto de métodos de ordenação dos 41 métodos avaliados 21 são estáveis e 20 são não estáveis [1].

[1] https://en.wikipedia.org/wiki/Sorting_algorithm

Estratégia do Quicksort

- O ideia do método está na **partição** realizada em uma lista a ser ordenada.
  - Essa partição rearranja os elementos de uma lista $A[p..r]$ a partir de um índice i em $p..r$ tal que $A[p..i-1] < A[i] < A[i+1..r]$.
  - O elemento $v = A[i]$ é chamado de pivô.

Algoritmo

1. Iniciar com uma lista L de n itens
2. Escolher um item pivô v, de L
3. **Particionar** L em duas listas não ordenadas, $L1$ e $L2$
   1. $L1$: conterá todas as chaves menores que v
   2. $L2$: conterá todas as chaves maiores que v
   3. Itens com a mesma chave que v podem fazer parte de $L1$ ou $L2$
   4. O pivô v não faz parte de nenhuma das duas listas
4. **Ordenar**:
   1. $L1$ recursivamente, obtendo a lista ordenada $S1$
   2. $L2$ recursivamente, obtendo a lista ordenada $S2$
5. Concatenar $S1$, v, $S2$ — produzindo a lista ordenada S

Exemplo 1

- O pivô será sempre o primeiro elemento da lista.
- Na fase de partição, formaremos duas sub-listas, $L1$ e $L2$

```
    | 4 | 7 | 1 | 5 | 9 | 3 | 0 |
L1  | 1 | 3 | 0 |
L2  | 7 | 5 | 9 |
```

- $L1$ será ordenada recursivamente.
- como foi alcançado o caso base, as listas serão concatenadas

```
     | 1 | 3 | 0 |
L11  | 0 |
L12  | 3 |
 S1  | 0 | 1 | 3 |
```

Exemplo 1

	4	7	1	5	9	3	0
S1	0	1	3				
L2	7	5	9				

- *L2* será ordenada recursivamente.
- como foi alcançado o caso base, as listas serão concatenadas

	7	5	9
L21	5		
L22	9		
S2	5	7	9

Exemplo 1

	4	7	1	5	9	3	0
S1	0	1	3				
S2	5	7	9				

- após ordenar as sub-listas, é feita a concatenação final

0	1	3	4	5	7	9

Exemplo 2

- Um arranjo de números ordenados

	0	1	3	4	5	7	9
L1							
L2	1	3	4	5	7	9	

- Desenvolvimento na lousa...

- Quando a entrada já está ordenada o tempo de execução é $\Theta(n^2)$,
- escolher o primeiro item como pivô é uma má escolha para esse caso.

Escolha do Pivô

- É crucial para o bom desempenho do método, já que a fase de partição é a parte crítica do algoritmo.
- Há várias estratégias possíveis.

Elemento do meio

- Intuitivamente poderia ser uma boa escolha.
- No entanto, não funciona bem em alguns casos, levando o algoritmo à complexidade quadrática.

Exemplo 3

- Estratégia de escolha do pivô como elemento do meio.

```
    | 1 | 2 | 3 | 4 | 3 | 2 | 1 |
L1  | 1 | 2 | 3 | 3 | 2 | 1 |
L2  |

    L11 | 1 | 2 | 2 | 1 |
    L12 | 3 |

        L111 | 1 | 1 |
        L112 | 2 |

            L1111 | 1 |
            L1112 |
```

Exemplo 3

```
          L1111 | 1 |
          L1112 |

        S111 | 1 | 1 |
        S112 | 2 |

    S11 | 1 | 1 | 2 | 2 |
    S12 | 3 |

S1 | 1 | 1 | 2 | 2 | 3 | 3 |
S2 |

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

Escolha do Pivô

- Há outras opções como a escolha do elemento correspondente à mediana da lista, ou ainda o mais próximo da média.
- Para o caso em que se *conhece* a **distribuição** dos dados, podemos utilizar a estratégia de escolha do pivô mais adequada àquela distribuição.
- Quando não há conhecimento...

Escolha aleatória

- Escolher aleatoriamente um item da lista L como pivô.
- Na média teremos uma partição da lista na proporção: $\frac{1}{4}$ e $\frac{3}{4}$.
- É possível provar que, se a partição da lista ocorrer pelo menos metade das vezes nessa proporção, o **tempo de execução esperado** é $O(n \cdot log\ n)$.

Escolha do Pivô

Mediana de três

- Escolher três elementos aleatoriamente.
- Utilizar como pivô o elemento correspondente à mediana dos três.
  - Essa estratégia aumenta ainda mais as chances de se obter caso esperado de $O(n \cdot log\ n)$.
  - Como há um maior custo em se obter três elementos aleatórios e obter a mediana, essa estratégia é utilizada apenas em listas grandes. Quando a lista a ser ordenada tem tamanhos menores, utiliza-se a escolha aleatória simples.

Exemplo 4

- Escolha do pivô de forma aleatória.
- Desenvolvimento na lousa...

| 0 | 1 | 3 | 4 | 5 | 7 | 9 |

| 1 | 0 | 4 | 3 | 5 | 9 | 7 |

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

Quicksort em Listas Ligadas

- Nesse caso é interessante tratar o problema da partição como sendo a partição em 3 Listas:
  - $L1$ contendo chaves menores que o pivô.
  - $L2$ contendo chaves maiores que o pivô.
  - L_v contendo chaves iguais ao pivô.
- A ordenação é realizada apenas em $L1$ e $L2$ e não em L_v.
- A concatenação é realizada na forma: $S1$, L_v , $L2$.

```
   | 5 | 7 | 5 | 0 | 6 | 5 | 5 |
L1 | 0 |
L2 | 7 | 6 |
Lv | 5 | 5 | 5 | 5 |
```

Quicksort em Arranjos

- Quicksort é considerado rápido para realizar ordenação *in-place*, ou seja, que utiliza apenas movimentações dentro do próprio arranjo, sem uso de memória auxiliar.
- É importante prestar atenção à implementação para evitar casos de execução quadrática.
- Mesmo alguns livros fornecem algoritmos que podem ser lentos em alguns casos.
- Um algoritmo possível será apresentado a seguir.

Quicksort em Arranjos: Algoritmo

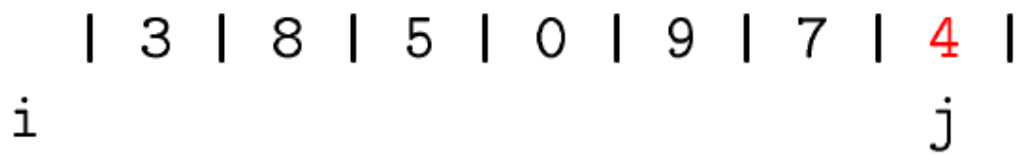
- Considere um arranjo A.
- Ordenar os itens de $A[p]$ até $A[r]$.
- Ao escolher um pivô v, substitua-o pelo último item ($A[r]$).
- Vamos utilizar duas variáveis de controle, $i = p - 1$ e $j = r$:

```
  | 3 | 8 | 4 | 0 | 9 | 7 | 5 |
    p       v               r

  | 3 | 8 | 5 | 0 | 9 | 7 | 4 |
i                             j
```

- O arranjo será ordenado então para as posições maiores que i e menores que j.

Quicksort em Arranjos: Algoritmo



Operações

- Incrementar i até que encontre uma chave maior ou igual ao pivô.
- Decrementar j até que encontre uma chave menor ou igual ao pivô.
- Trocar itens i, j.
- Parar quando $i \geq j$. e substituir o pivô de volta à posição inicial.

Quicksort: Código-fonte

```
int quicksort(int a[], int p, int r) {
    int t;
    if (p < r) {
        int v = (rand()%(r-p))+p;
        int pivo = a[v];
        a[v] = a[r];
        a[r] = pivo;
        int i = p-1;
        int j = r;
        do {
            do {
                i++;
            } while (a[i] < pivo);
            do {
                j--;
            } while ((a[j] > pivo) && (j > p));
            if (i < j){
                t = a[i];
                a[i] = a[j];
                a[j] = t;
            }
        } while (i<j);
        a[r] = a[i];
        a[i] = pivo;
        quicksort(a, p, i-1);
        quicksort(a, i+1, r);
    }
    return 0;
}
```

Estratégias de Escolha do Pivô

- Primeiro elemento.
- Último elemento.
- Elemento do meio.
- Elemento aleatório.
- Mediana de 3 (primeiro, meio e último).
- Mediana de 3 (aleatório).
- Mediana de 3 de três mediana de 3.

Estratégias de Escolha do Pivô

- Primeiro elemento.
  - Pior caso: quando os elementos estão em ordem crescente ou decrescente.
  - Exemplo: | 0 | 1 | 3 | 4 | 5 | 7 | 9 |
- Último elemento.
  - Pior caso: quando os elementos estão em ordem crescente ou decrescente.
  - Exemplo: | 9 | 7 | 5 | 4 | 3 | 1 | 0 |
- Elemento do meio.
  - Pior caso: quando os elementos formam uma espécie de triângulo.
  - Exemplo: | 1 | 2 | 3 | 4 | 3 | 2 | 1 |
- Elemento aleatório.
  - Pior caso: depende da escolha dos índices (índices: 3, 0, 2, 6, 5, 1, 4).
  - Exemplo: | 3 | 8 | 4 | 0 | 9 | 7 | 5 |

Desempenho na Escolha do Pivô

- Pivô
  - Elemento aleatório: N° Comparações $= 1,386 \cdot n \cdot lg\ n$.
  - Mediana de 3 elementos aleatórios: N° Comparações $= 1,188 \cdot n \cdot lg\ n$.
  - Melhoria: $\frac{(1,386-1,188)}{1,386} = 14,28\%$.

Análise de Complexidade

- O desempenho do quicksort depende do particionamento ser balanceado ou não balanceado.
- Um particionamento balanceado divide o conjunto a ser ordenado em duas Listas L1 e L2 de "mesmo tamanho".
- Pior caso ocorre quando o particionamento é totalmente não balanceado.
- Melhor caso ocorre quando o particionamento é totalmente balanceado.

Análise de Complexidade

Particionamento no Pior caso:

- Exemplo: A = [0 1 3 4 5 7 8].
- Critério de escolha do pivô: primeiro elemento.

Dessa forma, temos:

- $v = 0$
- $L1 = \varnothing$
- $L2 = \{1, 3, 4, 5, 7, 8\}$

Análise de Complexidade: Pior Caso

$T(n) = T(n-1) + T(0) + \Theta(n)$

Fazendo $T(0) = 1$ temos:

$T(n) = T(n-1) + 1 + \Theta(n)$

$T(n) = T(n-1) + 1 + cn$

$T(n) = T(n-2) + 1 + 1 + c(n-1) + cn$

$T(n) = T(n-3) + 1 + 1 + 1 + c(n-2) + c(n-1) + cn$

$\cdots$

$T(n) = T(n-k) + k + c(n-(k-1)) + \cdots + c(n-1) + cn$

Para $k = n$ temos:

$T(n) = T(n-n) + n + c(n-(n-1)) + \cdots + c(n-1) + cn$

$T(n) = T(0) + n + c(1 + 2 + \cdots + n)$

$T(n) = 1 + n + c \cdot \sum_{i=1}^{n} i$

$T(n) = 1 + n + c \cdot \sum_{i=1}^{n} i$

Resolvendo o somatório temos:

$T(n) = 1 + n + c \cdot n \cdot (n+1)/2$

$T(n) = cn^2/2 + cn/2 + n + 1 = \Theta(n^2)$

Análise de Complexidade

Particionamento de Melhor caso:

- Exemplo: A = [0 1 3 4 5 7 8]
- Critério de escolha do pivô: elemento do meio

Dessa forma, temos:

- $v = 4$
- $L1 = \{0, 1, 3\}$
- $L2 = \{5, 7, 8\}$

Análise de Complexidade: Melhor Caso

$T(n) = 2T(n/2) + \Theta(n)$
$T(n) = 2T(n/2) + cn$
Resolvendo pelo método mestre temos:
$a = 2$
$b = 2$
$f(n) = cn$
Caso 1: $f(n) = O(n^{log_b a - \varepsilon})$
$cn = O(n^{log_2 2 - \varepsilon})$
$cn = O(n^{1-\varepsilon})$
Falso
Caso 2: $f(n) = \Theta(n^{log_b a})$
$cn = \Theta(n^{log_2 2})$
$cn = \Theta(n^1)$
Verdade, então:
$T(n) = \Theta(n^{log_b a} \cdot lg\ n)$
$T(n) = \Theta(n \cdot lg\ n)$

Análise de Complexidade: Caso Médio

$T(n) = T(9n/10) + T(n/10) + cn$

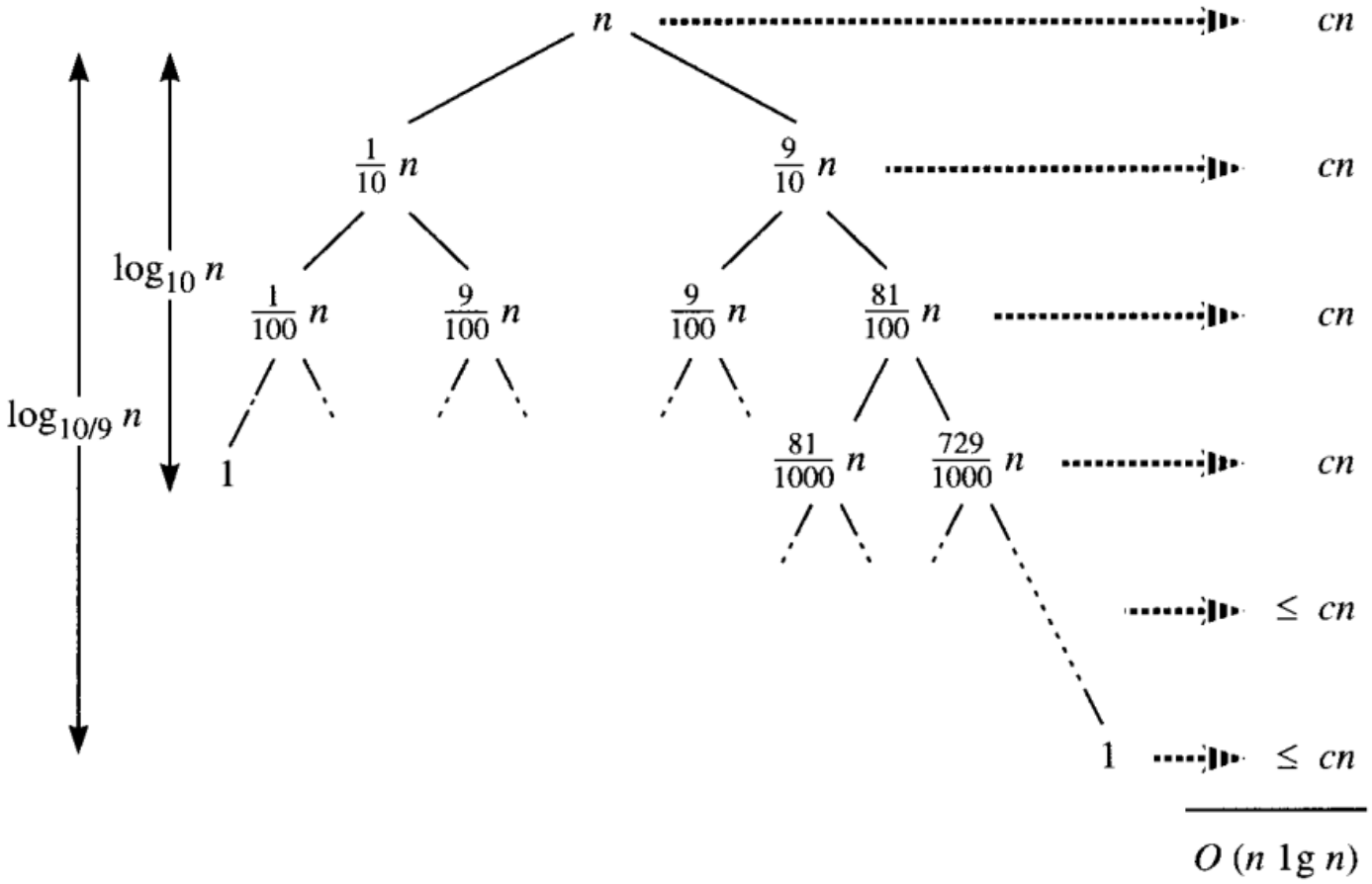


Figura: Particionamento proporcional a 9 para 1.

Intuição para Análise de Complexidade

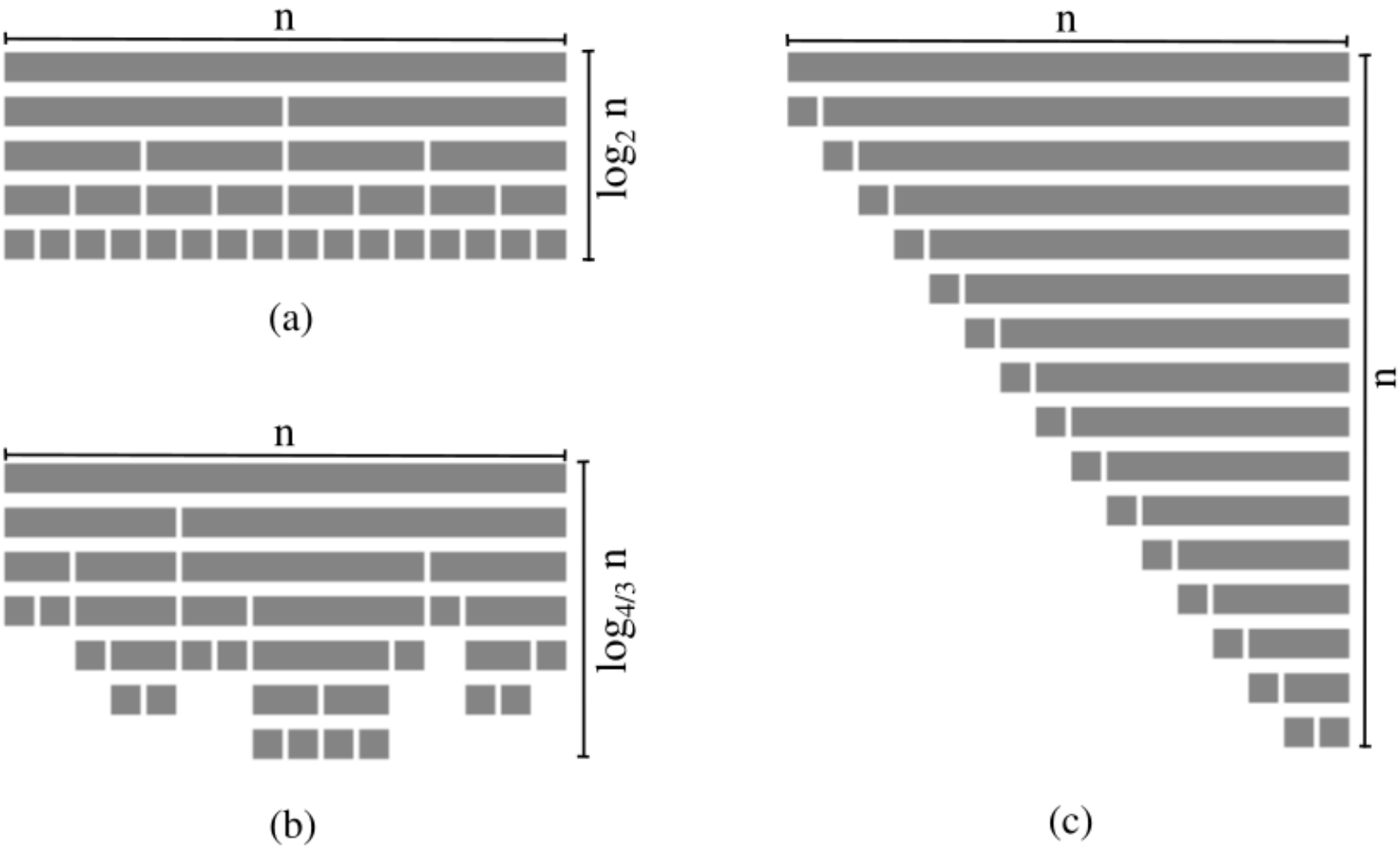


Figura: (a) Melhor caso. (b) Caso médio. (c) Pior caso.

Melhorias no Quicksort

- Quicksort com insertion sort:
  - Quicksort tem muita sobrecarga para vetores pequenos.
  - Estratégia: rodar o insertion sort quando o vetor é menor que k itens
    - Literatura: $k =\sim 10$
    - Meus Testes: $k =\sim 20$ obtive melhoria de $\sim 7\%$ sobre o quicksort
- Dual-pivot quicksort: Usa dois particionamento em três subarrays.
  - Chaves menores que p_1.
  - Chaves entre p_1 e p_2.
  - Chaves maiores que p_2.
  - Recursivamente ordena os três subarrays.
- Three-pivot quicksort: Usa três particionamento em quatro subarrays.
  - Chaves menores que p_1.
  - Chaves entre p_1 e p_2.
  - Chaves entre p_2 e p_3.
  - Chaves maiores que p_3.
  - Recursivamente ordena os três subarrays.

Pergunta

Quando cada algoritmo de ordenação deve ser usado?

- **Quick sort:** Quando não se precisa de ordenação estável. E o desempenho no caso médio é mais importante que no pior caso.
- **Merge sort:** Quando se precisa de ordenação estável. E um método que garanta um bom desempenho no pior caso. Esse método possui uma constante maior que o quick sort e gasta mais memória.
- **Insertion sort:** Quando se precisa de ordenação estável. E quando a entrada n é garantidamente pequena. Quando a entrada estiver quase-ordenada. Seu fator constante é pequeno.
- **Bubble sort:** Quando se precisa de ordenação estável. E quando a entrada n é garantidamente pequena. Sua vantagem sobre o insertion sort é a facilidade de implementação.
- **Selection sort:** Quando não se precisa de ordenação estável. E quando a entrada n é garantidamente pequena. Sua vantagem sobre o insertion sort é a facilidade de implementação.

Análise Empírica dos Algoritmos

Tempo estimado de execução:

- Home PC: Executa 10^8 comparações/segundo.
- Supercomputador: Executa 10^{12} comparações/segundo.

	insertion sort (N^2)			mergesort (N log N)			quicksort (N log N)		
computer	thousand	million	billion	thousand	million	billion	thousand	million	billion
home	instant	2.8 hours	317 years	instant	1 second	18 min	instant	0.6 sec	12 min
super	instant	1 second	1 week	instant	instant	instant	instant	instant	instant

Figura: Comparação entre Insertion Sort, Merge Sort e Quick Sort.

- Lição: Bons algoritmos são melhores que supercomputadores.