Algoritmos de ordenação
Quicksort

Sumário

Introdução

- Tempo de execução no pior caso: $\Theta(n^2)$
- Tempo de execução esperado: $\Theta(n \lg n)$
- Os fatores constantes escondidos em $\Theta(n \lg n)$ são pequenos
- Ordenação local
- Funciona bem em ambientes de memória virtual

Descrição do quicksort

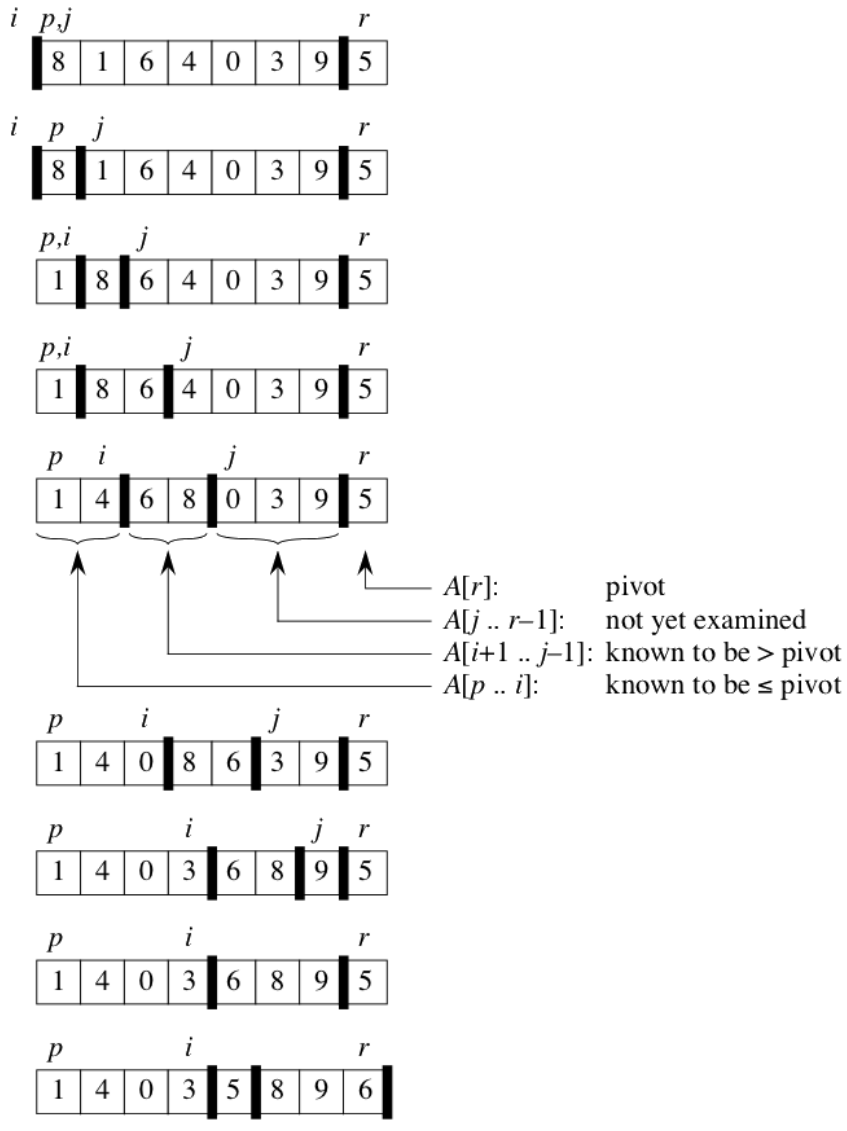
- Baseado no paradigma dividir para conquistar
- Para ordenar um array $A[p..r]$
  - **Dividir**: Particionar o array $A[p..r]$ em dois subarrays $A[p..q-1]$ e $A[q+1..r]$, tal que cada elemento de $A[p..q-1]$ seja menor ou igual que $A[q]$, e $A[q]$ seja menor ou igual a cada elemento de $A[q+1..r]$
  - **Conquistar**: Ordenar os dois subarrays recursivamente
  - **Combinar**: Como os subarrays são ordenados localmente, não é necessário nenhum trabalho para combiná-los
- A chave do algoritmo é o procedimento que faz o particionamento, que devolve o índice q que separa os subarrays

Procedimento quicksort

```
quicksort(A, p, r)
1 if p < r
2   q = partition(A, p, r)
3   quicksort(A, p, q - 1)
4   quicksort(A, q + 1, r)
```

Para ordenar um array A inteiro, a chamada inicial é $quicksort(A, 1, A.comprimento)$

Exemplo do funcionamento do procedimento `partition`



Procedimento `partition`

```
partition(A, p, r)
1 x = A[r]
2 i = p - 1
3 for j = p to r - 1
4   if A[j] <= x
5     i = i + 1
6     troca(A[i], A[j])
7 troca(A[i+1], A[r])
8 return i + 1
```

Análise: O tempo de execução de `partition` sobre um subarray $A[p..r]$ é $\Theta(n)$, onde $n = r - p + 1$.

Desempenho do quicksort

- O tempo de execução do `quicksort` depende do particionamento dos subarrays
  - Se o particionamento é balanceado, o `quicksort` é executado tão rápido quanto o `merge-sort`
  - Se o particionamento não é balanceado, o `quicksort` é executado tão lento quanto o `insertion-sort`

Pior caso

- Ocorre quando os subarrays estão completamente desbalanceados
- Um subarray tem 0 elementos e outro tem $n - 1$
- Obtemos a recorrência

$$
\begin{aligned}
T(n) &= T(n-1) + T(0) + \Theta(n) && (1)\\
&= T(n-1) + \Theta(n) && (2)\\
&= \Theta(n^2) && (3)
\end{aligned}
$$

- Mesmo tempo de execução do `insertion-sort`
- Ocorre quando o array já está ordenado

Melhor caso

- Ocorre quando os subarrays estão balanceados
- Um subarray tem tamanho $\lfloor n/2 \rfloor$ e o outro tem tamanho $\lceil n/2 \rceil - 1$
- Obtemos a recorrência

$$
\begin{aligned}
T(n) &\leq 2T(n/2) + \Theta(n) && (4)\\
&= \Theta(n \lg n) \text{ (caso 2 do teorema mestre)} && (5)
\end{aligned}
$$

Particionamento balanceado

- O tempo médio de execução do quicksort é muito mais próximo do melhor caso do que do pior caso
- Suponha que o algoritmo de particionamento sempre produza uma divisão na proporção 9 para 1
- Obtemos a recorrência

$$\begin{aligned} T(n) &\leq T(9n/10) + T(n/10) + \Theta(n) && (6) \\ &= O(n \lg n) && (7) \end{aligned}$$

- Porque o resultado é o mesmo que quando o particionamento é balanceado?
- Vejamos a árvore de recursão

Árvore de recursão

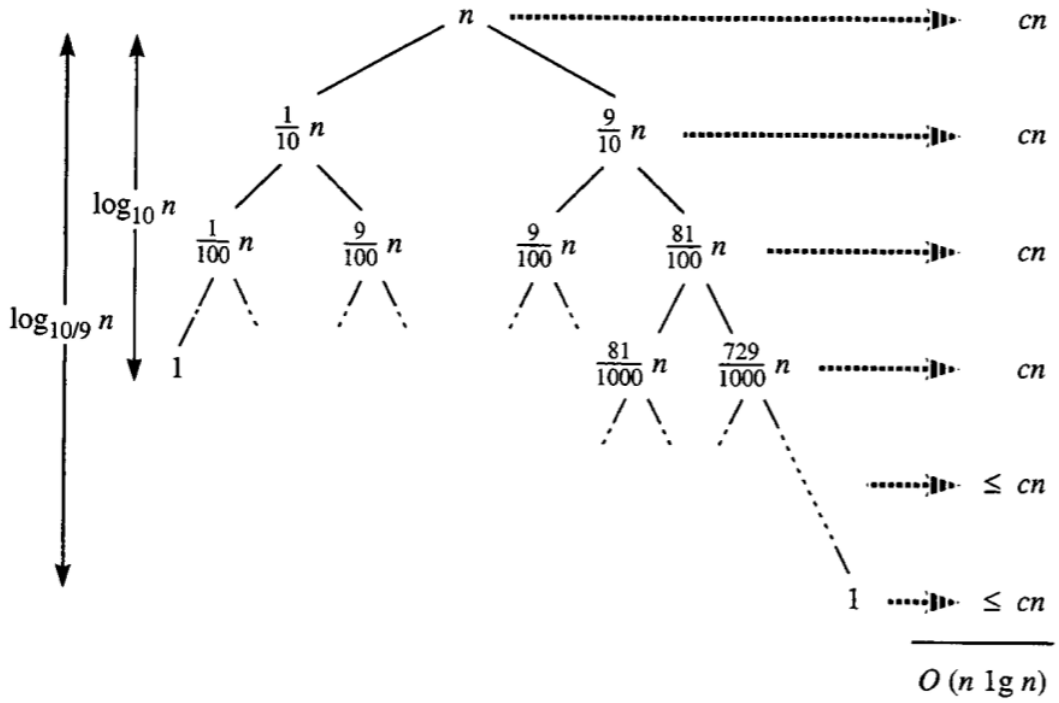


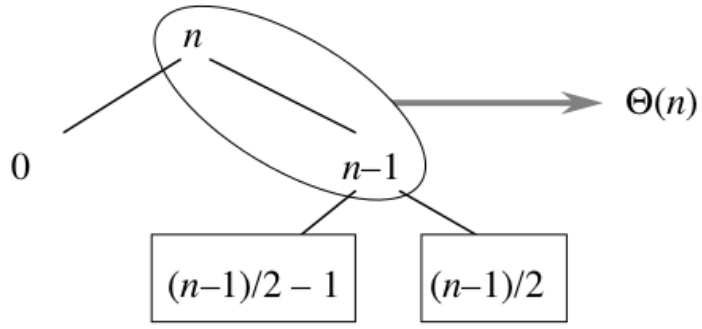
FIGURA 7.4 Uma árvore de recursão para QUICKSORT, na qual PARTITION sempre produz uma divisão de 9 para 1, resultando no tempo de execução $O(n \lg n)$. Os nós mostram tamanhos de subproblemas, com custos por nível à direita. Os custos por nível incluem a constante c implícita no termo $\Theta(n)$

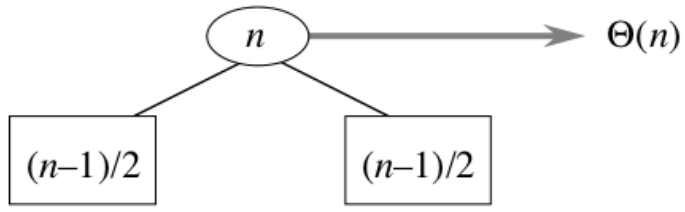
Particionamento balanceado

- Temos $\log_{10} n$ níveis completos e $\log_{10/9} n$ níveis não vazios
- Desde que seja constante, a base do logaritmo não importa para a notação assintótica
- Qualquer divisão de proporção constante gerará uma árvore de recursão de profundidade $\Theta(\lg n)$

Intuição para o caso médio

- A proporção das divisões na árvore de recursão não será sempre constante
- No caso médio, haverá uma mistura de divisões boas e ruins
- Para facilitar o entendimento, suponha que que as divisões boas e ruins alternem seus níveis na árvore, obtemos





- Em ambas as figuras, o tempo de execução é $O(n \lg n)$

Versão aleatória do quicksort

- Para explorar o caso médio, assumimos que todas as permutações de entrada são igualmente possíveis
- O que nem sempre é verdade
- Para corrigir esta situação, adicionamos aleatoriedade ao quicksort
- A ideia é não usar sempre $A[r]$ como pivô. Ao invés, escolhemos um elemento do array aleatoriamente
- Como o pivô é escolhido aleatoriamente, esperamos que a divisão do array de entrada seja equilibrada na média

Versão aleatória do quicksort

```
randomized-partition(A, p, r)
1 i = random(p, r)
2 troca(A[r], A[i])
3 return partition(A, p, r)

randomized-quicksort(A, p, r)
1 if p < r
2   q = randomized-partition(A, p, r)
3   randomized-quicksort(A, p, q - 1)
3   randomized-quicksort(A, q + 1, r)
```

Análise do pior caso

- Vamos mostrar que uma divisão do pior caso em todos os níveis produz um tempo de execução $\Theta(n^2)$
- Podemos escrever a recorrência como

$$T(n) = \max_{0 \leq q \leq n-1} (T(q^2) + T(n-q-1)^2) + \Theta(n) \quad (8)$$

- onde o parâmetro q varia de 0 a $n-1$
- Vamos supor que $T(n) \leq cn^2$
- Fazendo a substituição, obtemos

$$T(n) = \max_{0 \leq q \leq n-1} (cq^2 + c(n-q-1)^2) + \Theta(n) \quad (9)$$

$$= c \max_{0 \leq q \leq n-1} (q^2 + (n-q-1)^2) + \Theta(n) \quad (10)$$

Análise do pior caso

- O valor máximo de $(q^2 + (n - q - 1)^2)$ ocorre quando q é 0 ou $n - 1$, o que significa que

$$\begin{aligned} \max_{0 \leq q \leq n-1} (q^2 + (n - q - 1)^2) &\leq (n-1)^2 && (11) \\ &= n^2 - 2n + 1 && (12) \end{aligned}$$

- Obtemos

$$\begin{aligned} T(n) &\leq cn^2 - c(2n-1) + \Theta(n) && (13) \\ &\leq cn^2 \qquad \text{se } c(2n-1) \geq \Theta(n) && (14) \end{aligned}$$

- Portanto, o tempo de execução do quicksort no pior caso é $O(n^2)$
- Podemos mostrar que $T(n) = \Omega(n^2)$, e portanto $T(n) = \Theta(n^2)$

Exercícios

7.1-1 Usando a figura 7.1 como modelo, ilustre a operação de `partition` sobre o array $A = \{13, 19, 9, 5, 12, 8, 7, 4, 11, 2, 6, 21\}$.

7.1-2 Que valor de q `partition` retorna quando todos os elementos no arranjo $A[p..r]$ têm o mesmo valor? Modifique `partition` de forma que $q = (p + r)/2$ quando todos os elementos no array $A[p..r]$ têm o mesmo valor.

7.1-3 Forneça um breve argumento mostrando que o tempo de execução de `partition` sobre um subarray de tamanho n é $\Theta(n)$.

7.1-4 De que maneira você modificaria `quicksort` para fazer a ordenação em ordem não crescente?

7.2-1 Use o método de substituição para provar que a recorrência $T(n) = T(n-1) + \Theta(n)$ tem a solução $T(n) = \Theta(n^2)$.

7.2-2 Qual o tempo de execução de `quicksort` quando todos os elementos do array A têm o mesmo valor.

7.2-3 Mostre que o tempo de execução de `quicksort` é $\Theta(n^2)$ quando o array A contém elementos distintos e está ordenado em ordem decrescente.

Referências

- Algoritmos: Teoria e Prática. Thomas H. Cormen – Charles E. Leiserson – Ronald L. Rivest. Tradução da 2ª edição americana.