Projeto e Análise de Algoritmos

Ordenação – Divisão e Conquista - QuickSort

Prof. Dr. Lidio Mauro Lima de Campos

limadecampos@gmail.com

Universidade Federal do Pará – UFPA
ICEN
FACOMP

Agenda

- Introdução
- Descrição do quicksort
- Desempenho do quicksort
  - Pior caso
  - Melhor caso
- Particionamento balanceado
- Versão aleatória do quicksort
- Análise do quicksort
- Pior caso
- Exercícios

Ordenação Rápida (*Quick Sort) - Introdução*

- O QuickSort (ordenação rápida) e provavelmente o algoritmo mais usado na pratica para ordenar vetores.
- Sua complexidade de pior caso e Ө(n^2), rotina de particionamento produz um subproblemas com n-1 elementos e outro com 0 elementos, mas a chance dela ocorrer fica menor a medida que n cresce.
- Sua complexidade no melhor caso e Ө(nlog_2^n), rotina Partition produz dois subproblemas, cada um de tamanho não maior que n/2.

- Funciona bem até mesmo em ambientes de memória virtual.

Ordenação Rápida (*Quick Sort) - Descrição*

- Baseado na técnica de Projeto de Algoritmos 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 $A[q+1..r]$.
    - $A[p..q-1]$ A[q] $A[q+1..r]$
  - **Conquistar**: Ordenar os dois subarrays $A[p..q-1]$ e $A[q+1..r]$. recursivamente
  - **Combinar**: Como os subarrays são ordenados localmente, não é necessário nenhum trabalho para combiná-los. A[p..r] estará ordenado.

- A chave do algoritmo é **o procedimento que faz o particionamento**, que devolve o índice q que separa os subarrays

Procedimento (*Quick Sort)*

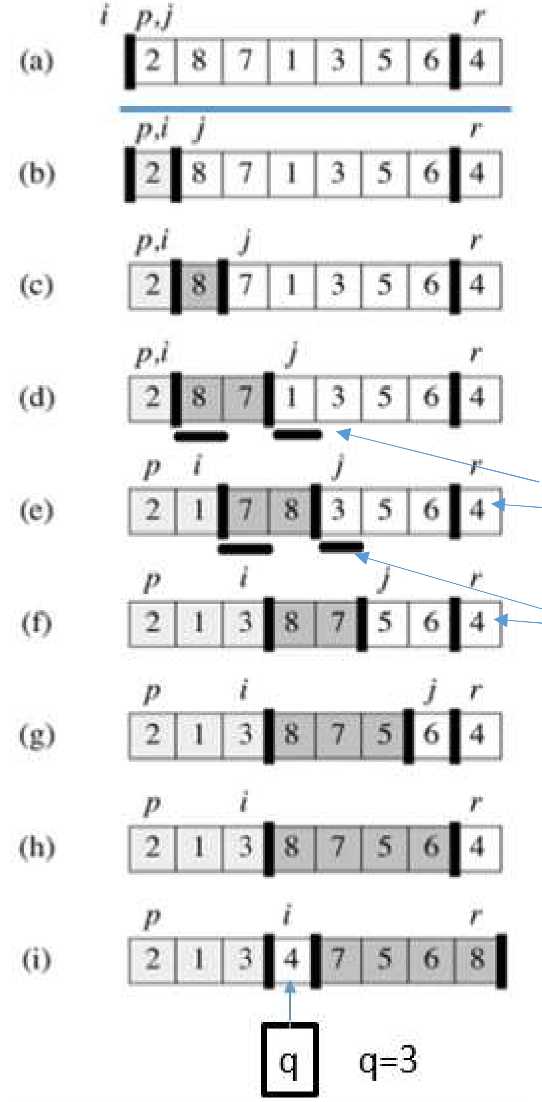
QUICKSORT(A, p, r)

```
if p < r
   q = PARTITION(A, p, r)
   QUICKSORT(A, p, q - 1)
   QUICKSORT(A, q + 1, r)
```

Para ordenar um array *A* inteiro, a chamada inicial é QUICKSORT (*A,0,7*)

Exemplo do funcionamento do procedimento Partition

1) QUICKSORT (*A,0,7*), **q=PARTITION(A,0,7)**



a)Config.Inicial, nenhum dos elementos foi inserido em nenhuma das duas partições.
p=0,i=-1, j=0 , r=7 , x=A[r]=4

b)For j=0...r-1 If(a[0]<=4)V, i=-1+1=0, A[0]=A[0], o valor 2 e permutado por ele mesmo.j=1

c)For j=1..r-1 If(a[1]=8<=4)F , j=2
d)For j=2..r-1 If(a[2]=7<=4)F , j=3
(c)-(d)-Os valores 8 e 7 são acrescentados à partição de valores maiores.

e)For j=3..r-1 If(a[3]=1<=4)V, i=i+1=1, A[1]=8 Troca com A[3]=1, A[3]=1 Troca com A[1]=8, os valores 1 e 8 são permutados, e a partição menor cresce. j=4

f) For j=4..r-1, If(a[4]=3<=4) V, i=i+1=2 A[2]=7 Troca com A[4]=3, A[4]=3 Troca com A[2]=7, a partição menor cresce.j=5

g)For j=5..r-1. If(a[5]=5<=4)F,
h)For j=6..r-1 If(a[6]=6<=6)F, j=6
a partição maior cresce para conter 5 e 6. e o Laco termina

i)Trocar A[i+1]=A[3] por A[r]=7, o elemento pivô é permutado de moso que se encontra entre as duas partições.

QUICKSORT(A, p, r)

if p < r

1) q = PARTITION(A, p, r)
2) QUICKSORT(A, p, q - 1)
3) QUICKSORT(A, q + 1, r)

PARTITION(A, p, r)

$x = A[r]$

$i = p - 1$

for $j = p$ **to** $r - 1$

  if $A[j] \leq x$

    $i = i + 1$

    trocar $A[i]$ por $A[j]$

trocar $A[i+1]$ por $A[r]$

return $i + 1$

https://www.youtube.com/watch?v=WprjBK0p6rw

Procedimento Partition

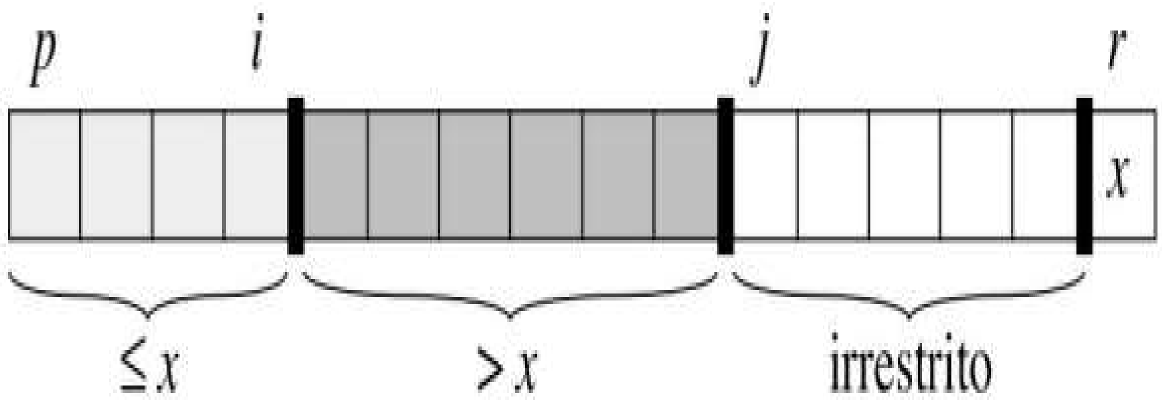


Figura 7.2 As quatro regiões mantidas pelo procedimento PARTITION em um subarranjo $A[p\,..\,r]$. Os valores em $A[p\,..\,i]$ são menores ou iguais a x, os valores em $A[i+1\,..\,j-1]$ são maiores que x e $A[r]=x$. O subarranjo $A[j\,..\,r-1]$ pode aceitar quaisquer valores.

No início de cada iteração do laço das linhas 3−6, para qualquer índice k do arranjo,

1. Se $p \leq k \leq i$, então $A[k] \leq x$.
2. Se $i+1 \leq k \leq j-1$, então $A[k] > x$.
3. Se $k = r$, então $A[k] = x$.

https://www.youtube.com/watch?v=WprjBK0p6rw

Procedimento Partition

```
partition(A, p, r)
x = A[r]
i = p - 1
for j = p to r - 1   (condição de parada p=r)
  if A[j] <= x
    i = i + 1
    troca(A[i], A[j])
troca(A[i+1], A[r])
return i + 1
```

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

Ordenação Rápida (*Quick Sort) – Particionamento do vetor*

O algoritmo abaixo implementa o QuickSort, ate que todos os segmentos tenham tamanho 1.

#arr[] --> Array que será ordenado,
#p --> Índice de Inicio,
#r --> índice de fim*/

```
def quickSort(arr,p,r):
    if p < r:
        q = partition(arr,p,r)
        quickSort(arr, p, q-1)
        quickSort(arr, q+1, r)
```

Ordenação Rápida (*Quick Sort)*

```
def partition(arr,p,r):
        i =(p-1)          # índice do menor elemento
        x = arr[r]        # pivô

        for j in range(p , r):
            if arr[j] <= x: # Se o elemento corrente is menor ou igual ao pivô
                i = i+1
                arr[i],arr[j] = arr[j],arr[i] # incrementa o índice do menor elemento

        arr[i+1],arr[r] = arr[r],arr[i+1]
        return ( i+1 )
```

Desempenho Quick Sort – *Particionamento Não Balanceado – Pior Caso*

- Ocorre quando a **rotina de particionamento produz um subproblemas com n-1** elementos e **outro com 0 elementos**.
- Um **subarray** tem **0 elementos** e **outro** **tem *n* – 1 elementos** .
- O **particionamento (partition) custa ϴ(n)**
- A **chamada recursiva para uma arranjo de Tamanho 0**, apenas retorna **T(0)=ϴ(1)**
- Obtemos a recorrência:
  - **T(n)**=**T(n-1)**+**T(0)**+**ϴ(n)**
  - **T(n)**=**T(n-1)**+**ϴ(n)**
  - **T(n)=ϴ(n²)**
- A solução dessa recorrência é $T(n)=O(n^2)$, que é a mesma complexidade (no pior caso) do Bubble Sort, Insertion Sort e Selection Sort. Esse é o ponto fraco do Quicksort.
- Nas versões de Hoare e Cormen, esse caso patológico ocorre quando o **array está ordenado em ordem crescente ou descrente**.

Desempenho Ordenação Rápida (*Quick Sort) – Particionamento Não Balanceado – Pior Caso*

Passo base: $T(1) = 1$.

Expandir:

$k = 1$: $T(n) = \underline{T(n-1)} + n$, T(n-1)=T(n-2)+n-1

$k = 2$: $T(n) = [T(n-2) + n - 1] + n = T(n-2) + n - 1 + n$, T(n-2)=T(n-3)+n-2

$k = 3$: $T(n) = [\underline{T(n-3)} + n - 2] + n - 1 + n = T(n-3) + n - 2 + n - 1 + n$

Conjecturar: Após k expansões, temos

$$T(n) = T(n-k) + \sum_{i=0}^{k-1}(n-i).$$

Observando a expansão, ela irá parar quando $k = n - 1$, isso porque a base da recursividade é definida para 1 (um).

a1=n
an=2
nt=2

$$\text{Logo, } T(n) = T(1) + \sum_{i=0}^{n-2}(n-i) = 1 + (\frac{n(n+1)}{2} - 1) = \Theta(n^2).$$

S=(a1+an)n/2
S=(n+2)(n-1)/2

Desempenho QuickSort – *Particionamento Balanceado* – Melhor Caso

- Na divisão mais equitativa possível. **Partition produz dois subproblemas, cada um de tamanho não maior que n/2.**
- Um subarray tem tamanho $\left\lfloor \frac{n}{2} \right\rfloor$ e o outro tem tamanho $\left\lfloor \frac{n}{2} \right\rfloor - 1$
- Obtemos a recorrência:
  - $T(1)=\Theta(1)$ e
  - $T(n)=T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + \Theta(n)$, ou seja
  - $T(n)=2T\left(\frac{n}{2}\right)+ \Theta(n)$, para $n>1$
- Resolvendo-se a formulação acima: $\boldsymbol{\Theta(n\,log_2^n)}$
- **Balanceando igualmente os dois lados da partição em todo nível da recursão**, **obtemos uma algoritmo assintoticamente mais rápido.**

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 $\Theta(n \lg n)$**

Versão Aleatória do QuickSort

Randomized-Partition(A, p, r)

```
i = random(p, r)
troca(A[r], A[i])
return Partition(A, p, r)
```

Randomized-Quicksort(A, p, r)

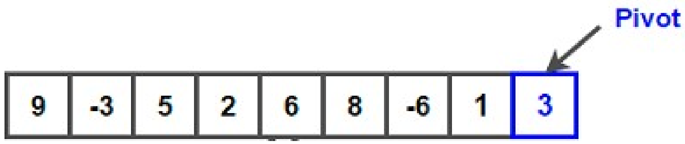
```
if p < r
q = Randomized-Partition(A, p, r)
Randomized-Quicksort(A, p, q - 1)
Randomized-Quicksort(A, q + 1, r)
```

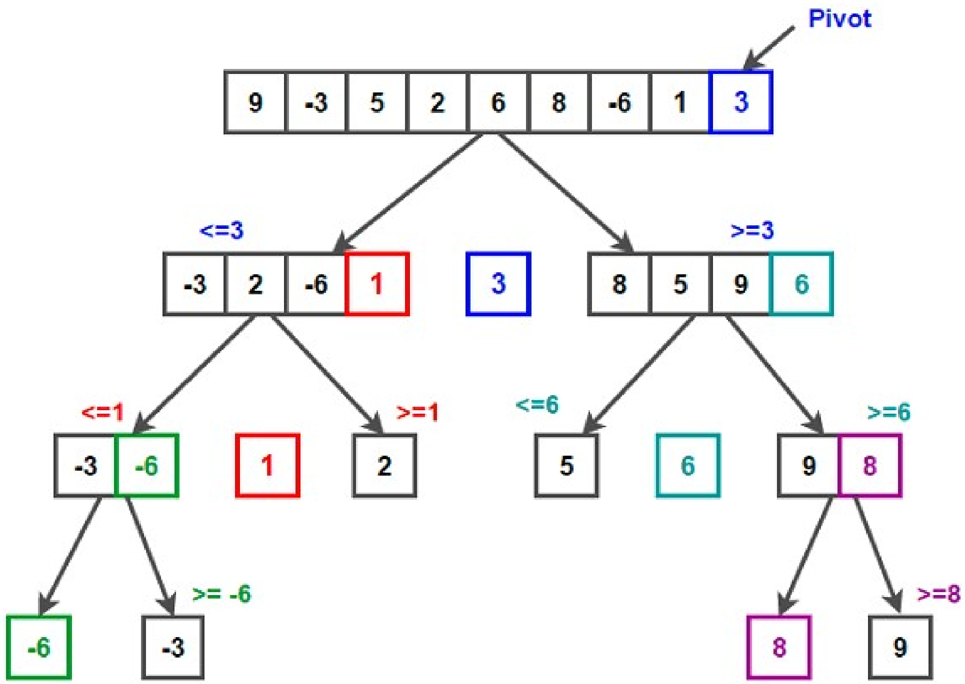
Versão Aleatória do QuickSort - Complexidade

- A aleatoriedade não evita o pior caso! **$T(n)=\Theta(n^2)$**
- Muitas pessoas consideram a versão aleatória do QuickSort o algoritmo preferido para entradas grandes o suficiente.

Versão Aleatória do QuickSort - Complexidade

- **Exercícios**
- 1)Mostrar intuitivamente os passos de ordenação do array abaixo, ilustrado as etapas do procedimento Partition.





Referências

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Algoritmos – Teoria e Prática, Tradução da Segunda Edição. Campus, 2016.
- Neto, Nelson Cruz Sampaio. Notas de Aula, P.A. Algoritmos, 2021.
- U. Manber, Algorithms: A Creative Approach, Addison-Wesley (1989).
- J. Kleinberg e E. Tardos, Algorithm Design, Addison Wesley, (2005).