Divisão e Conquista

Marcelo Keese Albertini
Faculdade de Computação
Universidade Federal de Uberlândia

16 de Abril de 2018

Aula de hoje

Nesta aula veremos

- Conceitos de Divisão e Conquista
- Análise de algoritmos de Divisão e Conquista
  - busca binária
  - mergesort
  - quicksort

Busca binária

```
int pos(int chave, int v[]) {
  int inf = 0;
  int sup = v.length - 1;
  while (inf <= sup) {
    // chave está em v[inf...sup] ou não existe
    int meio = inf + (sup-inf) /2;
    if (chave < v[meio]) sup = meio-1;
    else if (chave > v[meio]) inf = meio +1;
    else return meio;
  }
  return -1;
}
```

Número de comparações no pior caso

$B_N = B_{\lfloor N/2 \rfloor} + 1$ para $N > 1$ com $B_1 = 1$

Análise de busca binária (caso $N = 2^n$)

$B_N = B_{\lfloor N/2 \rfloor} + 1$ para $N > 1$ com $B_1 = 1$

Solução exata para $N = 2^n$

- $a_n \equiv B_{2^n}$
- $a_n = a_{n-1} + 1$, para $n > 0$ com $a_0 = 1$
- Expandir: $a_n = \sum_{1 \leq k \leq n} 1 = n$
- $B_N = \log N$, quando N é potência de 2

Análise de busca binária (caso geral)

Definir B_N como o número de bits na representação binária de N

- $B_1 = 1$
- Remover bit mais à direita de N resulta em $\lfloor N/2 \rfloor$
- Portanto, $B_N = B_{\lfloor N/2 \rfloor} + 1$
  - mesma recorrência que para busca binária

Exemplo

1101011	110101	1
107	53	
N	$N/2$	

Análise da busca binária

Teorema: número de bits B_N para N e o número de comparações na busca binária B_N é $\lfloor \lg N \rfloor + 1$

$$
\begin{aligned}
B_N &= \lfloor \lg N \rfloor + 1 \\
B_N &= n + 1 \qquad \text{para } 2^n \leq N < 2^{n+1} \text{ ou } n \leq \lg N < n + 1 \\
\Rightarrow n &= \lfloor \lg N \rfloor
\end{aligned}
$$

N	1	2	3	4	5	6	7	8	9
binário	1	10	11	100	101	110	111	1000	1001
$\lg N$	0	1.0	1.58...	2	2.32	2.58...	2.80...	3	3.16...
$\lfloor \lg N \rfloor$	0	1	1	2	2	2	2	3	3
$\lfloor \lg N \rfloor + 1$	1	2	2	3	3	3	3	4	4

Exercício: Busca ternária

- Vale a pena a dividir um array em três partes em vez de duas?

```java
int buscaTernaria(int[] a, int x) {
  int inf = 0, sup = a.length-1;
  while (inf <= sup) {
    int esq = inf +(sup-inf)/3, dir = sup -(sup-inf)/3;

    if      (a[esq] == x) return esq; // achou
    else if (a[dir] == x) return dir; // achou

    if      (a[esq] > x) sup = esq -1; //terço inferior
    else if (a[dir] < x) inf = dir +1; //terço superior
    else { inf = esq+1;  sup = dir -1;}//meio
    }
  }
  return -1; // nao achou
}
```

Mergesort

- Von Neumann: implementou para um EDVAC um dos primeiros computadores de propósito geral
- Ordenação estável em Java, C++, Python
- Comprova que ordenação por comparação é $O(n \log n)$

Mergesort

Ideia

1. dividir vetor em 2 metades
2. recursivamente ordenar cada metade
3. mesclar – merge – as duas metades ordenadas

entrada	5	1	3	6	4	2	9	0

Mergesort

Ideia

1. dividir vetor em 2 metades
2. recursivamente ordenar cada metade
3. mesclar – merge – as duas metades ordenadas

entrada	5	1	3	6	4	2	9	0
ordena esquerda	1	3	5	5	4	2	9	0

Mergesort

Ideia

1. dividir vetor em 2 metades
2. recursivamente ordenar cada metade
3. mesclar – merge – as duas metades ordenadas

entrada	5	1	3	6	4	2	9	0
ordena esquerda	1	3	5	5	4	2	9	0
ordena direita	1	3	5	6	0	2	4	9

Mergesort

Ideia

1. dividir vetor em 2 metades
2. recursivamente ordenar cada metade
3. mesclar – merge – as duas metades ordenadas

entrada	5	1	3	6	4	2	9	0
ordena esquerda	1	3	5	5	4	2	9	0
ordena direita	1	3	5	6	0	2	4	9
antes do merge	1	3	5	6	0	2	4	9

Mergesort

Ideia

1. dividir vetor em 2 metades
2. recursivamente ordenar cada metade
3. mesclar – merge – as duas metades ordenadas

entrada	5	1	3	6	4	2	9	0
ordena esquerda	1	3	5	5	4	2	9	0
ordena direita	1	3	5	6	0	2	4	9
antes do merge	1	3	5	6	0	2	4	9
ordenado	0	1	2	3	4	5	6	9

Operação Merge

copiar menor valor do auxiliar no vetor ordenado

1^i	3	5	6	0^j	2	4	9	10

0	#	#	#	#	#	#	#	#

Operação Merge

copiar menor valor do auxiliar no vetor ordenado

1^i	3	5	6	0^j	2	4	9	10

0	#	#	#	#	#	#	#	#

1^i	3	5	6	0	2^j	4	9	10

0	1	#	#	#	#	#	#	#

Operação Merge

copiar menor valor do auxiliar no vetor ordenado

1^i	3	5	6	0^j	2	4	9	10
0	#	#	#	#	#	#	#	#

1^i	3	5	6	0	2^j	4	9	10
0	1	#	#	#	#	#	#	#

1	3^i	5	6	0	2^j	4	9	10
0	1	2	#	#	#	#	#	#

Operação Merge

copiar menor valor do auxiliar no vetor ordenado

1^i	3	5	6	0^j	2	4	9	10
1^i	3	5	6	0	2^j	4	9	10
1	3^i	5	6	0	2^j	4	9	10
1	3^i	5	6	0	2	4^j	9	10

0	#	#	#	#	#	#	#	#
0	1	#	#	#	#	#	#	#
0	1	2	#	#	#	#	#	#
0	1	2	3	#	#	#	#	#

Operação Merge

copiar menor valor do auxiliar no vetor ordenado

1^i	3	5	6	0^j	2	4	9	10
0	#	#	#	#	#	#	#	#

1^i	3	5	6	0	2^j	4	9	10
0	1	#	#	#	#	#	#	#

1	3^i	5	6	0	2^j	4	9	10
0	1	2	#	#	#	#	#	#

1	3^i	5	6	0	2	4^j	9	10
0	1	2	3	#	#	#	#	#

1	3	5^i	6	0	2	4^j	9	10
0	1	2	3	4	#	#	#	#

Operação Merge

copiar menor valor do auxiliar	no vetor ordenado
1^i 3 5 6 0^j 2 4 9 10	0 # # # # # # # #
1^i 3 5 6 0 2^j 4 9 10	0 1 # # # # # # #
1 3^i 5 6 0 2^j 4 9 10	0 1 2 # # # # # #
1 3^i 5 6 0 2 4^j 9 10	0 1 2 3 # # # # #
1 3 5^i 6 0 2 4^j 9 10	0 1 2 3 4 # # # #
1 3 5^i 6 0 2 4 9^j 10	0 1 2 3 4 5 # # #

Operação Merge

copiar menor valor do auxiliar no vetor ordenado

1^i	3	5	6	0^j	2	4	9	10	0	#	#	#	#	#	#	#	#
1^i	3	5	6	0	2^j	4	9	10	0	1	#	#	#	#	#	#	#
1	3^i	5	6	0	2^j	4	9	10	0	1	2	#	#	#	#	#	#
1	3^i	5	6	0	2	4^j	9	10	0	1	2	3	#	#	#	#	#
1	3	5^i	6	0	2	4^j	9	10	0	1	2	3	4	#	#	#	#
1	3	5^i	6	0	2	4	9^j	10	0	1	2	3	4	5	#	#	#
1	3	5	6^i	0	2	4	9^j	10	0	1	2	3	4	5	6	#	#

Operação Merge

copiar menor valor do auxiliar no vetor ordenado

1^i 3 5 6 0^j 2 4 9 10	0 # # # # # # # #
1^i 3 5 6 0 2^j 4 9 10	0 1 # # # # # # #
1 3^i 5 6 0 2^j 4 9 10	0 1 2 # # # # # #
1 3^i 5 6 0 2 4^j 9 10	0 1 2 3 # # # # #
1 3 5^i 6 0 2 4^j 9 10	0 1 2 3 4 # # # #
1 3 5^i 6 0 2 4 9^j 10	0 1 2 3 4 5 # # #
1 3 5 6^i 0 2 4 9^j 10	0 1 2 3 4 5 6 # #

parte esquerda acabou, copiar os valores restantes

1 3 5 6^i 0 2 4 9^j 10	0 1 2 3 4 5 6 9 10

Operação Merge

copiar menor valor do auxiliar no vetor ordenado

auxiliar									vetor ordenado									
1^i	3	5	6	0^j	2	4	9	10		0	#	#	#	#	#	#	#	#
1^i	3	5	6	0	2^j	4	9	10		0	1	#	#	#	#	#	#	#
1	3^i	5	6	0	2^j	4	9	10		0	1	2	#	#	#	#	#	#
1	3^i	5	6	0	2	4^j	9	10		0	1	2	3	#	#	#	#	#
1	3	5^i	6	0	2	4^j	9	10		0	1	2	3	4	#	#	#	#
1	3	5^i	6	0	2	4	9^j	10		0	1	2	3	4	5	#	#	#
1	3	5	6^i	0	2	4	9^j	10		0	1	2	3	4	5	6	#	#

parte esquerda acabou , copiar os valores restantes

1	3	5	6^i	0	2	4	9^j	10		0	1	2	3	4	5	6	9	10

mergesort: implementação

```
1  void mergesort(int[] a, int lo, int hi) {
2    if (hi <= lo) return;
3    int mid = lo + (hi-lo)/2;
4    mergesort(a, lo, mid);   mergesort(a, mid+1, hi);
5
6    for (int k = lo;    k <= mid; k++)   b[k-lo] = a[k];
7    for (int k = mid+1; k <= hi; k++) c[k-mid-1] = a[k];
8
9    b[mid-lo+1]=c[hi-mid]=Integer.MAX_VALUE; //sentinelas
10
11   int i =0, j = 0;
12   for (int k = lo; k <= hi; k++) // merge
13     if (c[j] < b[i]) a[k] = c[j++];
14     else             a[k] = b[i++];
15 }
```

São necessárias N comparações (linha 13) para fazer um merge.
Vetores b e c são declarados externamente com N elementos.

Em cinza: posições desconsideradas do `merge` atual.
Em vermelho: subvetor com posições a partir de `inf` até `med`.
Em azul: subvetor com posições depois de `med` até `sup`.
Em verde: subvetor resultante do `merge`.

6	8	2	9	0	7	4	1	3	5

mergesort

Em cinza: posições desconsideradas do `merge` atual.

Em vermelho: subvetor com posições a partir de `inf` até `med`.

Em azul: subvetor com posições depois de `med` até `sup`.

Em verde: subvetor resultante do `merge`.

6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5

Em cinza: posições desconsideradas do `merge` atual.
Em vermelho: subvetor com posições a partir de `inf` até `med`.
Em azul: subvetor com posições depois de `med` até `sup`.
Em verde: subvetor resultante do `merge`.

6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5

Em cinza: posições desconsideradas do `merge` atual.
Em vermelho: subvetor com posições a partir de `inf` até `med`.
Em azul: subvetor com posições depois de `med` até `sup`.
Em verde: subvetor resultante do `merge`.

6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5

mergesort

Em cinza: posições desconsideradas do `merge` atual.
Em vermelho: subvetor com posições a partir de `inf` até `med`.
Em azul: subvetor com posições depois de `med` até `sup`.
Em verde: subvetor resultante do `merge`.

6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5

mergesort

Em cinza: posições desconsideradas do `merge` atual.
Em vermelho: subvetor com posições a partir de `inf` até `med`.
Em azul: subvetor com posições depois de `med` até `sup`.
Em verde: subvetor resultante do `merge`.

6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5

Em cinza: posições desconsideradas do `merge` atual.

Em vermelho: subvetor com posições a partir de `inf` até `med`.

Em azul: subvetor com posições depois de `med` até `sup`.

Em verde: subvetor resultante do `merge`.

6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5

Em cinza: posições desconsideradas do `merge` atual.
Em vermelho: subvetor com posições a partir de `inf` até `med`.
Em azul: subvetor com posições depois de `med` até `sup`.
Em verde: subvetor resultante do `merge`.

6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
0	2	6	8	9	7	4	1	3	5

mergesort

Em cinza: posições desconsideradas do `merge` atual.
Em vermelho: subvetor com posições a partir de `inf` até `med`.
Em azul: subvetor com posições depois de `med` até `sup`.
Em verde: subvetor resultante do `merge`.

6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
0	2	6	8	9	7	4	1	3	5
0	2	6	8	9	7	4	1	3	5

Em cinza: posições desconsideradas do `merge` atual.
Em vermelho: subvetor com posições a partir de `inf` até `med`.
Em azul: subvetor com posições depois de `med` até `sup`.
Em verde: subvetor resultante do `merge`.

6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
0	2	6	8	9	7	4	1	3	5
0	2	6	8	9	7	4	1	3	5
0	2	6	8	9	4	7	1	3	5

mergesort

Em cinza: posições desconsideradas do `merge` atual.
Em vermelho: subvetor com posições a partir de `inf` até `med`.
Em azul: subvetor com posições depois de `med` até `sup`.
Em verde: subvetor resultante do `merge`.

6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
0	2	6	8	9	7	4	1	3	5
0	2	6	8	9	7	4	1	3	5
0	2	6	8	9	4	7	1	3	5

0	2	6	8	9	4	7	1	3	5

Em cinza: posições desconsideradas do `merge` atual.
Em vermelho: subvetor com posições a partir de `inf` até `med`.
Em azul: subvetor com posições depois de `med` até `sup`.
Em verde: subvetor resultante do `merge`.

6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
0	2	6	8	9	7	4	1	3	5
0	2	6	8	9	7	4	1	3	5
0	2	6	8	9	4	7	1	3	5

0	2	6	8	9	4	7	1	3	5
0	2	6	8	9	1	4	7	3	5

mergesort

Em cinza: posições desconsideradas do `merge` atual.
Em vermelho: subvetor com posições a partir de `inf` até `med`.
Em azul: subvetor com posições depois de `med` até `sup`.
Em verde: subvetor resultante do `merge`.

6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
0	2	6	8	9	7	4	1	3	5
0	2	6	8	9	7	4	1	3	5
0	2	6	8	9	4	7	1	3	5

0	2	6	8	9	4	7	1	3	5
0	2	6	8	9	1	4	7	3	5
0	2	6	8	9	1	4	7	3	5

mergesort

Em cinza: posições desconsideradas do `merge` atual.
Em vermelho: subvetor com posições a partir de `inf` até `med`.
Em azul: subvetor com posições depois de `med` até `sup`.
Em verde: subvetor resultante do `merge`.

6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
0	2	6	8	9	7	4	1	3	5
0	2	6	8	9	7	4	1	3	5
0	2	6	8	9	4	7	1	3	5

0	2	6	8	9	4	7	1	3	5
0	2	6	8	9	1	4	7	3	5
0	2	6	8	9	1	4	7	3	5
0	2	6	8	9	1	4	7	3	5

mergesort

Em cinza: posições desconsideradas do `merge` atual.
Em vermelho: subvetor com posições a partir de `inf` até `med`.
Em azul: subvetor com posições depois de `med` até `sup`.
Em verde: subvetor resultante do `merge`.

6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
0	2	6	8	9	7	4	1	3	5
0	2	6	8	9	7	4	1	3	5
0	2	6	8	9	4	7	1	3	5

0	2	6	8	9	4	7	1	3	5
0	2	6	8	9	1	4	7	3	5
0	2	6	8	9	1	4	7	3	5
0	2	6	8	9	1	4	7	3	5
0	2	6	8	9	1	4	7	3	5

mergesort

Em cinza: posições desconsideradas do `merge` atual.
Em vermelho: subvetor com posições a partir de `inf` até `med`.
Em azul: subvetor com posições depois de `med` até `sup`.
Em verde: subvetor resultante do `merge`.

6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
0	2	6	8	9	7	4	1	3	5
0	2	6	8	9	7	4	1	3	5
0	2	6	8	9	4	7	1	3	5

0	2	6	8	9	4	7	1	3	5
0	2	6	8	9	1	4	7	3	5
0	2	6	8	9	1	4	7	3	5
0	2	6	8	9	1	4	7	3	5
0	2	6	8	9	1	4	7	3	5
0	2	6	8	9	1	3	4	5	7

Em cinza: posições desconsideradas do `merge` atual.
Em vermelho: subvetor com posições a partir de `inf` até `med`.
Em azul: subvetor com posições depois de `med` até `sup`.
Em verde: subvetor resultante do `merge`.

6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
0	2	6	8	9	7	4	1	3	5
0	2	6	8	9	7	4	1	3	5
0	2	6	8	9	4	7	1	3	5

0	2	6	8	9	4	7	1	3	5
0	2	6	8	9	1	4	7	3	5
0	2	6	8	9	1	4	7	3	5
0	2	6	8	9	1	4	7	3	5
0	2	6	8	9	1	4	7	3	5
0	2	6	8	9	1	3	4	5	7
0	2	6	8	9	1	3	4	5	7

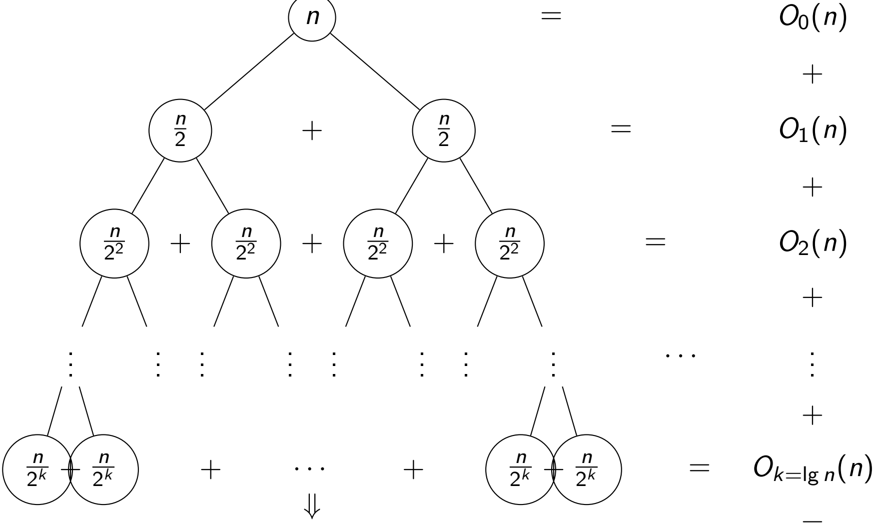
mergesort

Em cinza: posições desconsideradas do `merge` atual.
Em vermelho: subvetor com posições a partir de `inf` até `med`.
Em azul: subvetor com posições depois de `med` até `sup`.
Em verde: subvetor resultante do `merge`.

6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
0	2	6	8	9	7	4	1	3	5
0	2	6	8	9	7	4	1	3	5
0	2	6	8	9	4	7	1	3	5

0	2	6	8	9	4	7	1	3	5
0	2	6	8	9	1	4	7	3	5
0	2	6	8	9	1	4	7	3	5
0	2	6	8	9	1	4	7	3	5
0	2	6	8	9	1	4	7	3	5
0	2	6	8	9	1	3	4	5	7
0	2	6	8	9	1	3	4	5	7
0	1	2	3	4	5	6	7	8	9

Análise do mergesort: Ordem de Complexidade



$$O\left(\sum_{i=0}^{k} 2^i \cdot \frac{n}{2^i}\right) = O\left(\sum_{i=0}^{k} n\right) = O(k \cdot n) \Leftrightarrow O(n \cdot \lg n)$$

Complexidade de Tempo do Problema de Ordenação

Complexidade de tempo da ordenação por comparação é $\Omega(n \log n)$

(AOCP1, Knuth) Todo algoritmo de ordenação por comparação usa pelo menos

$$\lceil \lg N! \rceil > N \lg N - N / \ln 2$$

Ideia

- 1 comparação reduz tamanho de arranjo a ser ordenado por até um fator de 2
- Há $N!$ arranjos de N números
- Meta: obter apenas um arranjo de saída (o ordenado)
- Conclusão: número mínimo de comparações deve ser $\lceil \lg N! \rceil$
- Usando a fórmula de Stirling: $\lg(N!) > N \lg N - N / \ln 2$

Como o mergesort é $O(n \lg n)$, o que é igual ao melhor caso do problema, ele é ótimo.

Análise do mergesort: número de comparações - caso 2^n

- $C_N = C_{\lfloor N/2 \rfloor} + C_{\lceil N/2 \rceil} + N$ para $N > 1$ e $C_1 = 0$

Para $N = 2^n$ temos

$$C_{2^n} = 2C_{2^{n-1}} + 2^n \qquad \text{Dividir por } 2^n$$

$$\frac{C_{2^n}}{2^n} = \frac{C_{2^{n-1}}}{2^{n-1}} + 1 = \frac{C_{2^{n-2}}}{2^{n-2}} + 2 = \ldots \qquad \text{A cada iteração, soma}$$

$$\frac{C_{2^n}}{2^n} = \frac{C_{2^{n-k}}}{2^{n-k}} + k = \sum_{1 \le k \le n} 1 = n \qquad \text{Expande para } k = 1 \ldots n$$

$$C_{2^n} = n2^n \qquad \text{Voltar para } C_N \text{ usando } n = \lg N$$

$$C_{2^n} = C_N = \lg(N)2^{\lg N} = N \lg N \qquad \text{quando } N \text{ é potência de 2}$$

Análise do mergesort: número de comparações - caso geral

- $C_N = C_{\lfloor N/2 \rfloor} + C_{\lceil N/2 \rceil} + N$ para $N > 1$ e $C_1 = 0$

$$
\begin{aligned}
C_{N+1} &= C_{\lfloor (N+1)/2 \rfloor} + C_{\lceil (N+1)/2 \rceil} + (N+1) && \text{Para } N+1 \\
&= C_{\lceil N/2 \rceil} + C_{\lfloor N/2 \rfloor + 1} + N + 1 && \\
C_{N+1} - C_N &= C_{\lfloor N/2 \rfloor + 1} - C_{\lfloor N/2 \rfloor} + 1 && \text{(1) Subtrair} \\
D_N &= D_{\lfloor N/2 \rfloor} + 1 && D_N = C_{N+1} - C_N, D_1 = 2 \\
D_N &= \lfloor \lg N \rfloor + 2 && \text{Ver busca binária}
\end{aligned}
$$

Análise do mergesort: número de comparações - caso geral

- $C_N = C_{\lfloor N/2 \rfloor} + C_{\lceil N/2 \rceil} + N$ para $N > 1$ e $C_1 = 0$

$$
\begin{aligned}
D_N &= \lfloor \lg N \rfloor + 2 && \text{com } N > 1 \text{ e } D_1 = 2 \\
D_N &= C_{N+1} - C_N && \\
C_{N+1} &= D_N + C_N && \\
C_N &= D_{N-1} + C_{N-1} && \text{Expandir} \\
&= \lfloor \lg(N-1) \rfloor + 2 + C_{N-1} && \text{Iterar em } C_{N-1} \\
&= \sum_{1 \le k < N} [\lfloor \lg k \rfloor + 2] + C_1 && \text{mas } C_1 = 0 \\
C_N &= (N-1) + \sum_{1 \le k < N} (\lfloor \lg k \rfloor + 1) &&
\end{aligned}
$$

$C_N = N - 1 +$ número de bits dos números $< N$

Número de bits em $n < N$

- S_N = número de bits em inteiros positivos $< N$ em binário

$$S_N = S_{\lfloor N/2 \rfloor} + S_{\lceil N/2 \rceil} + N - 1, S_1 = 1$$

- Total de bits na matrix: $T_N = N(\lfloor \lg N \rfloor + 1)$
- Número de bits descontados: $V_N = \sum_{0 \le k \le \lfloor \lg N \rfloor} 2^k$

$$\begin{array}{c|l}
 & 8 \rightarrow 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 1 \\
\lfloor \lg N \rfloor + 1 & 4 \rightarrow 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 1\ 1\ 1 \\
 & 2 \rightarrow 0\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 0\ 0\ 1 \\
 & 1 \rightarrow 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0 \\
\hline
 & \qquad\qquad\qquad N
\end{array}$$

$$S_N = T_N - V_N \Rightarrow S_N = N\lfloor \lg N \rfloor + N - 2^{\lfloor \lg N \rfloor + 1} + 1$$

Análise do mergesort - caso geral

Número de bits de números $< N$

$S_N = N\lfloor \lg N \rfloor + N - 2^{\lfloor \lg N \rfloor + 1} + 1$

Comparações no mergesort para ordenar N números

$C_N = N - 1 +$ "número de bits de números $< N$" $= N - 1 + S_N$

$C_N = N\lfloor \lg N \rfloor + 2N - 2^{\lfloor \lg N \rfloor + 1}$

Uso prático do quicksort

quicksort

- Ordenação dicionário Inglês-Russo por Tony Hoare em 1959
- Algoritmo (e variantes) extensivamente analisado
- Mais rápido que mergesort, é in-place e não é estável
- Em C, `qsort()`
- Ordenação quicksort com 2 pivots em `java.util.Arrays` por Yaroslavskiy, Bentley e Bloch em 2009

quicksort: ideia

Ideia

1. **Desordenar** o vetor
2. **Particionar** tal que para algum elemento na posição `j` (pivot)
   - valor em `v[j]` está na posição correta
   - todos os valores à esquerda de *j* são menores que `v[j]`
   - todos os valores à direita de *j* são maiores que `v[j]`
3. **Ordenar** cada pedaço **recursivamente** sem copiar vetor

entrada	Q	U	I	C	K	S	O	R	T
desordenado	K	R	T	Q	S	O	I	U	C
partição	I	C	K	Q	U	R	T	S	O
ordena esq.	C	I	K	Q	U	R	T	S	O
ordena dir.	C	I	K	O	Q	R	S	T	U
resultado	C	I	K	O	Q	R	S	T	U

Partição

Objetivo

Dividir vetor em duas regiões separadas pelo `pivot`.

- A região anterior ao `pivot` consiste de elementos menores ou iguais a ele.
- A região posterior ao `pivot` consiste de elementos maiores a ele.

Guardamos duas variáveis de índice: da esquerda `i` e da direita `j`.

$v[p]$ é o elemento pivot.

Antes: $\boxed{v[p]}\boxed{v[\ldots]}$

Durante: $\boxed{v[p]}\boxed{v[\ldots] <= v[p]}\boxed{v[i] \ldots v[j]}\boxed{v[\ldots] > v[p]}$

Depois: $\boxed{v[\ldots] <= v[p]}\boxed{v[p]}\boxed{v[\ldots] > v[p]}$

```
quicksort(a,0,a.length-1);
```

```java
void quicksort(int[] a, int lo, int hi) {
  if (hi <= lo) return; //compara
  int i = lo-1, j = hi;
  int t, v = a[hi];

  while (true) { //estágio de particionamento
    while (a[++i] < v); //compara p/ achar maior que v

    while (v < a[--j]) //compara
      if (j == lo) break;

    if (i >= j) break; //terminou partições
    t = a[i]; a[i] = a[j]; a[j] = t;//troca
  }

  t = a[i]; a[i] = a[hi]; a[hi] = t;//troca
  quicksort(a, lo, i-1);
  quicksort(a, i+1, hi);
}
```

Análise do quicksort

- Número de comparações é $O(N^2)$

Quicksort usa, em média,

- $(N-1)/2$ estágios de particionamento
- $2(N+1)(H_{N+1} - 3/2) \approx 2N \ln N - 1.846N$ comparações
- $(N+1)(H_{N+1} - 3)/3 + 1 \approx 0.333N \ln N - 0.865$ trocas

Série harmônica H_N:

$$H_N = \sum_{1 \leq k \leq N} 1/k$$

Análise de pior caso

Pior caso ocorre quando o `pivot` for sempre o menor elemento do vetor. Ou seja, $k = 1$ em $T_N = T_K + T_{N-K} + \alpha N$

Relação de recorrência: pior caso k=1

Divide array com $k = 1$	$T_N = T_{N-1} + T_1 + \alpha n$
Obtém eq. para $n - 1$	$T_N = [T_{N-2} + T_1 + \alpha(N-1)] + \alpha N$
Reorganiza	$T_N = T_{N-2} + 2T_1 + \alpha(N - 1 + N)$

Análise de pior caso k=1: continuando

Para $k = 1$ $$T_N = T_{N-1} + T_1 + \alpha N$$

Análise de pior caso k=1: continuando

$$
\begin{aligned}
\text{Para } k = 1 \qquad & T_N = T_{N-1} + T_1 + \alpha N \\
\text{Eq. de } N - 1 \qquad & = [T_{N-2} + T_1 + \alpha(N-1)] + \alpha N
\end{aligned}
$$

Análise de pior caso k=1: continuando

Para $k = 1$	$T_N = T_{N-1} + T_1 + \alpha N$
Eq. de $N - 1$	$= [T_{N-2} + T_1 + \alpha(N-1)] + \alpha N$
Organiza	$= T_{N-2} + 2T_1 + \alpha(N - 1 + N)$

Análise de pior caso k=1: continuando

$$\begin{array}{rr}
\text{Para } k = 1 & T_N = T_{N-1} + T_1 + \alpha N \\
\text{Eq. de } N-1 & = [T_{N-2} + T_1 + \alpha(N-1)] + \alpha N \\
\text{Organiza} & = T_{N-2} + 2T_1 + \alpha(N - 1 + N) \\
\text{Eq. de } N-2 & = [T_{N-3} + T_1 + \alpha(N-2)] + 2T_1 + \alpha(N-1) + \alpha N
\end{array}$$

Análise de pior caso k=1: continuando

$$
\begin{aligned}
\text{Para } k = 1 \qquad & T_N = T_{N-1} + T_1 + \alpha N \\
\text{Eq. de } N-1 \qquad & = [T_{N-2} + T_1 + \alpha(N-1)] + \alpha N \\
\text{Organiza} \qquad & = T_{N-2} + 2T_1 + \alpha(N-1+N) \\
\text{Eq. de } N-2 \qquad & = [T_{N-3} + T_1 + \alpha(N-2)] + 2T_1 + \alpha(N-1) + \alpha N \\
\text{Organiza} \qquad & = T_{N-3} + 3T_1 + \alpha[(N-2) + (N-1) + N]
\end{aligned}
$$

Análise de pior caso k=1: continuando

$$
\begin{aligned}
\text{Para } k = 1 \quad & T_N = T_{N-1} + T_1 + \alpha N \\
\text{Eq. de } N-1 \quad & = [T_{N-2} + T_1 + \alpha(N-1)] + \alpha N \\
\text{Organiza} \quad & = T_{N-2} + 2T_1 + \alpha(N-1+N) \\
\text{Eq. de } N-2 \quad & = [T_{N-3} + T_1 + \alpha(N-2)] + 2T_1 + \alpha(N-1) + \alpha N \\
\text{Organiza} \quad & = T_{N-3} + 3T_1 + \alpha[(N-2) + (N-1) + N] \\
\text{Eq. de } N-i \quad & = T_{N-i} + iT_1 + \alpha[(N-i+1) + \ldots + (N-1) + N]
\end{aligned}
$$

Análise de pior caso k=1: continuando

$$
\begin{array}{rl}
\text{Para } k = 1 & T_N = T_{N-1} + T_1 + \alpha N \\
\text{Eq. de } N-1 & = [T_{N-2} + T_1 + \alpha(N-1)] + \alpha N \\
\text{Organiza} & = T_{N-2} + 2T_1 + \alpha(N-1+N) \\
\text{Eq. de } N-2 & = [T_{N-3} + T_1 + \alpha(N-2)] + 2T_1 + \alpha(N-1) + \alpha N \\
\text{Organiza} & = T_{N-3} + 3T_1 + \alpha[(N-2) + (N-1) + N] \\
\text{Eq. de } N-i & = T_{N-i} + iT_1 + \alpha[(N-i+1) + \ldots + (N-1) + N] \\
\text{Soma} & = T_{N-i} + iT_1 + \alpha\sum_{j=0}^{i-1}(N-j)
\end{array}
$$

Análise de pior caso k=1: continuando

$$
\begin{array}{rr}
\text{Para } k=1 & T_N = T_{N-1} + T_1 + \alpha N \\
\text{Eq. de } N-1 & = [T_{N-2} + T_1 + \alpha(N-1)] + \alpha N \\
\text{Organiza} & = T_{N-2} + 2T_1 + \alpha(N-1+N) \\
\text{Eq. de } N-2 & = [T_{N-3} + T_1 + \alpha(N-2)] + 2T_1 + \alpha(N-1) + \alpha N \\
\text{Organiza} & = T_{N-3} + 3T_1 + \alpha[(N-2) + (N-1) + N] \\
\text{Eq. de } N-i & = T_{N-i} + iT_1 + \alpha[(N-i+1) + \ldots + (N-1) + N] \\
\text{Soma} & = T_{N-i} + iT_1 + \alpha \sum_{j=0}^{i-1}(N-j) \\
\text{Vai até } i=N-1 & = T_{N-N+1} + (N-1)T_1 + \alpha \sum_{j=0}^{N-1-1}(N-j)
\end{array}
$$

Análise de pior caso k=1: continuando

$$
\begin{array}{rl}
\text{Para } k = 1 & T_N = T_{N-1} + T_1 + \alpha N \\
\text{Eq. de } N-1 & = [T_{N-2} + T_1 + \alpha(N-1)] + \alpha N \\
\text{Organiza} & = T_{N-2} + 2T_1 + \alpha(N-1+N) \\
\text{Eq. de } N-2 & = [T_{N-3} + T_1 + \alpha(N-2)] + 2T_1 + \alpha(N-1) + \alpha N \\
\text{Organiza} & = T_{N-3} + 3T_1 + \alpha[(N-2)+(N-1)+N] \\
\text{Eq. de } N-i & = T_{N-i} + iT_1 + \alpha[(N-i+1) + \ldots + (N-1) + N] \\
\text{Soma} & = T_{N-i} + iT_1 + \alpha \sum_{j=0}^{i-1}(N-j) \\
\text{Vai até } i = N-1 & = T_{N-N+1} + (N-1)T_1 + \alpha \sum_{j=0}^{N-1-1}(N-j) \\
\text{Resultado} & = NT_1 + \alpha[(\sum_{j=1}^{N} j) - 1]
\end{array}
$$

Análise de pior caso k=1: continuando

Para $k = 1$	$T_N = T_{N-1} + T_1 + \alpha N$
Eq. de $N - 1$	$= [T_{N-2} + T_1 + \alpha(N-1)] + \alpha N$
Organiza	$= T_{N-2} + 2T_1 + \alpha(N - 1 + N)$
Eq. de $N - 2$	$= [T_{N-3} + T_1 + \alpha(N-2)] + 2T_1 + \alpha(N-1) + \alpha N$
Organiza	$= T_{N-3} + 3T_1 + \alpha[(N-2) + (N-1) + N]$
Eq. de $N - i$	$= T_{N-i} + iT_1 + \alpha[(N - i + 1) + \ldots + (N-1) + N]$
Soma	$= T_{N-i} + iT_1 + \alpha \sum_{j=0}^{i-1}(N - j)$
Vai até $i = N - 1$	$= T_{N-N+1} + (N-1)T_1 + \alpha \sum_{j=0}^{N-1-1}(N - j)$
Resultado	$= NT_1 + \alpha[(\sum_{j=1}^{N} j) - 1]$
Só o somatório	$\sum_{j=1}^{N} j = (N+1)N/2$

Análise de melhor caso

Melhor caso ocorre quando o `pivot` for sempre o elemento que divide o vetor na metade. Ou seja, $k = N/2$ em $T_N = 2T_{N/2} + \alpha N$

Relação de recorrência: melhor caso k=N/2

$$T_N = 2T_{N/2} + \alpha N$$

Análise de melhor caso

Melhor caso ocorre quando o `pivot` for sempre o elemento que divide o vetor na metade. Ou seja, $k = N/2$ em $T_N = 2T_{N/2} + \alpha N$

Relação de recorrência: melhor caso k=N/2

$$
\begin{aligned}
& T_N = 2T_{N/2} + \alpha N \\
\text{Para } N/4 \quad & = 2(2T_{N/4} + \alpha N/2) + \alpha N
\end{aligned}
$$

Análise de melhor caso

Melhor caso ocorre quando o `pivot` for sempre o elemento que divide o vetor na metade. Ou seja, $k = N/2$ em $T_N = 2T_{N/2} + \alpha N$

Relação de recorrência: melhor caso k=N/2

$$
\begin{aligned}
& & T_N &= 2T_{N/2} + \alpha N \\
&\text{Para } N/4 & &= 2(2T_{N/4} + \alpha N/2) + \alpha N \\
&\text{Organizar} & &= 4T_{n/4} + 2\alpha N/2 + \alpha N
\end{aligned}
$$

Análise de melhor caso

Melhor caso ocorre quando o `pivot` for sempre o elemento que divide o vetor na metade. Ou seja, $k = N/2$ em $T_N = 2T_{N/2} + \alpha N$

Relação de recorrência: melhor caso k=N/2

$$\begin{aligned} & T_N = 2T_{N/2} + \alpha N \\ \text{Para } N/4 \quad & = 2(2T_{N/4} + \alpha N/2) + \alpha N \\ \text{Organizar} \quad & = 4T_{n/4} + 2\alpha N/2 + \alpha N \\ & = 2^2 T_{N/2^2} + 2\alpha N \end{aligned}$$

Análise de melhor caso

Melhor caso ocorre quando o `pivot` for sempre o elemento que divide o vetor na metade. Ou seja, $k = N/2$ em $T_N = 2T_{N/2} + \alpha N$

Relação de recorrência: melhor caso k=N/2

$$
\begin{aligned}
& & T_N &= 2T_{N/2} + \alpha N \\
&\text{Para } N/4 & &= 2(2T_{N/4} + \alpha N/2) + \alpha N \\
&\text{Organizar} & &= 4T_{n/4} + 2\alpha N/2 + \alpha N \\
& & &= 2^2 T_{N/2^2} + 2\alpha N \\
&\text{Para } N/8 & &= 2^2[2(T_{N/8} + \alpha N/8)] + 2\alpha N
\end{aligned}
$$

Análise de melhor caso

Melhor caso ocorre quando o `pivot` for sempre o elemento que divide o vetor na metade. Ou seja, $k = N/2$ em $T_N = 2T_{N/2} + \alpha N$

Relação de recorrência: melhor caso k=N/2

$$
\begin{aligned}
& T_N = 2T_{N/2} + \alpha N \\
\text{Para } N/4 \quad & = 2(2T_{N/4} + \alpha N/2) + \alpha N \\
\text{Organizar} \quad & = 4T_{n/4} + 2\alpha N/2 + \alpha N \\
& = 2^2 T_{N/2^2} + 2\alpha N \\
\text{Para } N/8 \quad & = 2^2[2(T_{N/8} + \alpha N/8)] + 2\alpha N \\
\text{Organizar} \quad & = 2^3 T_{N/2^3} + 3\alpha N
\end{aligned}
$$

Análise de melhor caso

Melhor caso ocorre quando o `pivot` for sempre o elemento que divide o vetor na metade. Ou seja, $k = N/2$ em $T_N = 2T_{N/2} + \alpha N$

Relação de recorrência: melhor caso k=N/2

$$
\begin{aligned}
& & T_N &= 2T_{N/2} + \alpha N \\
&\text{Para } N/4 & &= 2(2T_{N/4} + \alpha N/2) + \alpha N \\
&\text{Organizar} & &= 4T_{n/4} + 2\alpha N/2 + \alpha N \\
& & &= 2^2 T_{N/2^2} + 2\alpha N \\
&\text{Para } N/8 & &= 2^2[2(T_{N/8} + \alpha N/8)] + 2\alpha N \\
&\text{Organizar} & &= 2^3 T_{N/2^3} + 3\alpha N \\
&\text{Para } N/2^k & &= 2^k T_{N/2^k} + k\alpha N
\end{aligned}
$$

Análise de melhor caso

Melhor caso ocorre quando o `pivot` for sempre o elemento que divide o vetor na metade. Ou seja, $k = N/2$ em $T_N = 2T_{N/2} + \alpha N$

Relação de recorrência: melhor caso k=N/2

$$
\begin{aligned}
& & T_N &= 2T_{N/2} + \alpha N \\
& \text{Para } N/4 & &= 2(2T_{N/4} + \alpha N/2) + \alpha N \\
& \text{Organizar} & &= 4T_{n/4} + 2\alpha N/2 + \alpha N \\
& & &= 2^2 T_{N/2^2} + 2\alpha N \\
& \text{Para } N/8 & &= 2^2[2(T_{N/8} + \alpha N/8)] + 2\alpha N \\
& \text{Organizar} & &= 2^3 T_{N/2^3} + 3\alpha N \\
& \text{Para } N/2^k & &= 2^k T_{N/2^k} + k\alpha N \\
& \text{Até } N = 2^k\text{, com } k = \lg N & T_N &= nT_1 + \alpha N \lg N
\end{aligned}
$$

Análise caso médio do quicksort

- Comparações para particionar N elementos: $(N+1)$
- Fazer média para cada par de partições
  - C_{j-1} e C_{N-j} definidos pelo pivot em $j-1$
- Para $N > 1$, $C_1 = C_0 = 0$,

$$C_N = N + 1 + \frac{1}{N} \sum_{1 \leq j \leq N} (C_{j-1} + C_{N-j})$$

$$C_N = N + 1 + \frac{1}{N} \sum_{1 \le j \le N} (C_{j-1} + C_{N-j}) \qquad N > 1,\, C_1 = C_0 = 0$$

$$C_N = N + 1 + \frac{2}{N} \sum_{1 \le j \le N} C_{j-1} \qquad \text{Faz } j = N - j + 1 \text{ em } C_{N-j}$$

$$NC_N = N(N+1) + 2 \sum_{1 \le j \le N} C_{j-1} \qquad \text{Multiplica } N$$

$$NC_N - (N-1)C_{N-1} = 2N + 2C_{N-1} \qquad \text{Subtrai } NC_N \text{ em } N \text{ e } N-1$$

$$NC_N = 2N + (N+1)C_{N-1} \qquad \text{Divide por } N(N+1)$$

$$\frac{C_N}{N+1} = \frac{C_{N-1}}{N} + \frac{2}{N+1} \qquad \text{Itera em } \frac{C_N}{N+1}$$

$$C_N/(N+1) = C_1/2 + 2 \sum_{3 \le k \le N+1} 1/k \qquad \text{Finaliza}$$

$$C_N = (N+1)2(-3/2 + \sum_{1 \le k \le N+1} 1/k) \qquad = 2(N+1)(H_N - 3/2)$$

Exercícios

1. Escreva a recorrência para o total de comparações do quicksort para todas as possíveis $N!$ permutações
2. Resolva a seguinte recorrência:

$$A_N = 1 + \frac{2}{N} \sum_{1 \leq j \leq N} A_{j-1}$$

   com $N > 0$ e $A_0 = 0$
3. Escreva e resolva a recorrência do número médio de estágios de particionamento do quicksort

Vimos

- Primeiros algoritmos de divisão e conquista
- Análise de busca binária, mergesort, quicksort