MC3305
Algoritmos e Estruturas de Dados II

Aula 04 – Ordenação parcial

Prof. Jesús P. Mena-Chalco
jesus.mena@ufabc.edu.br

2Q-2015

Ordenação

- Limite assintótico para algoritmos de ordenação baseadas em comparações

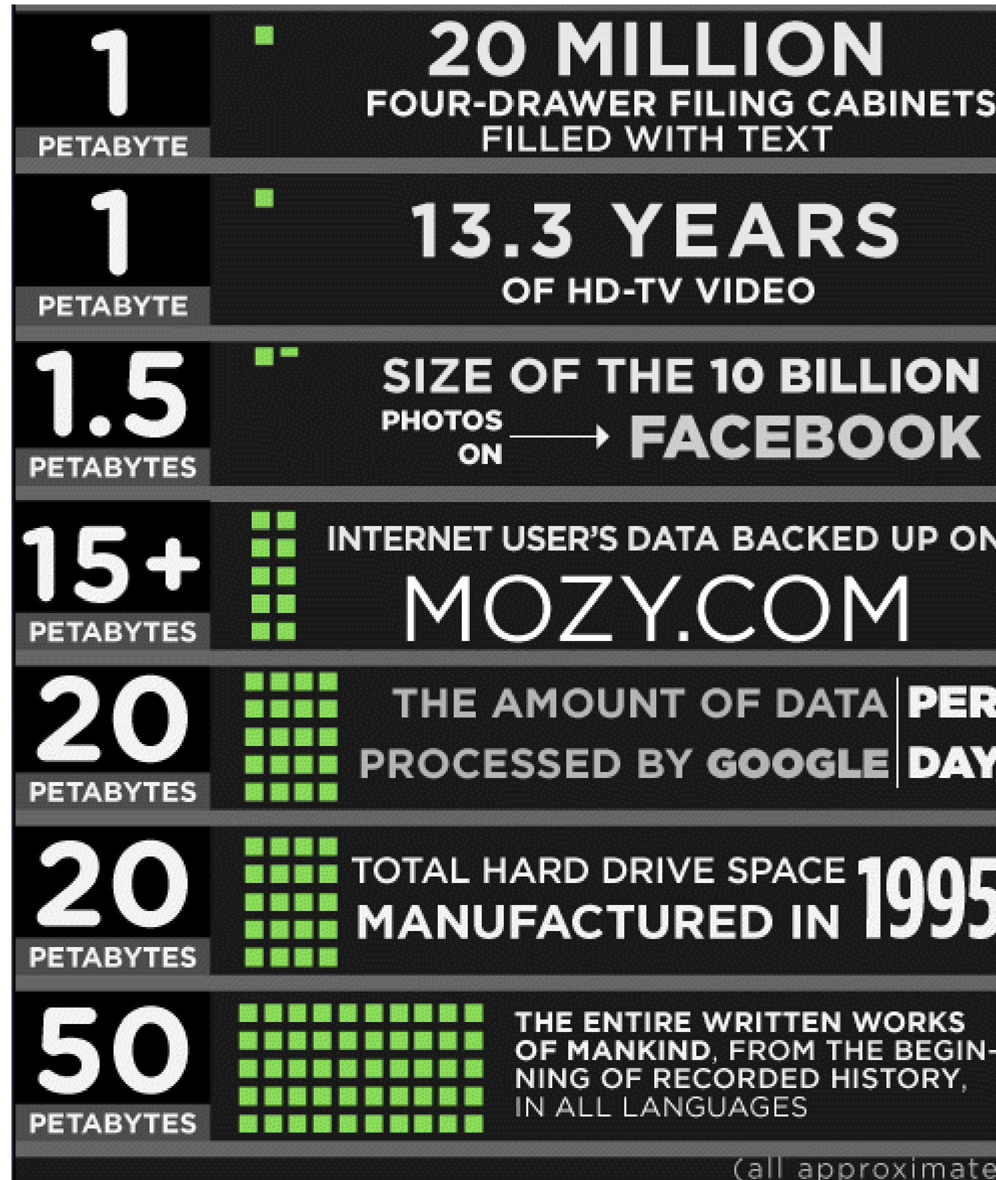
$$\Omega(n \log(n))$$

- A ordenação em tempo linear está associada a algoritmos que não consideram comparações entre seus elementos

$$O(n)$$

Grande escala?

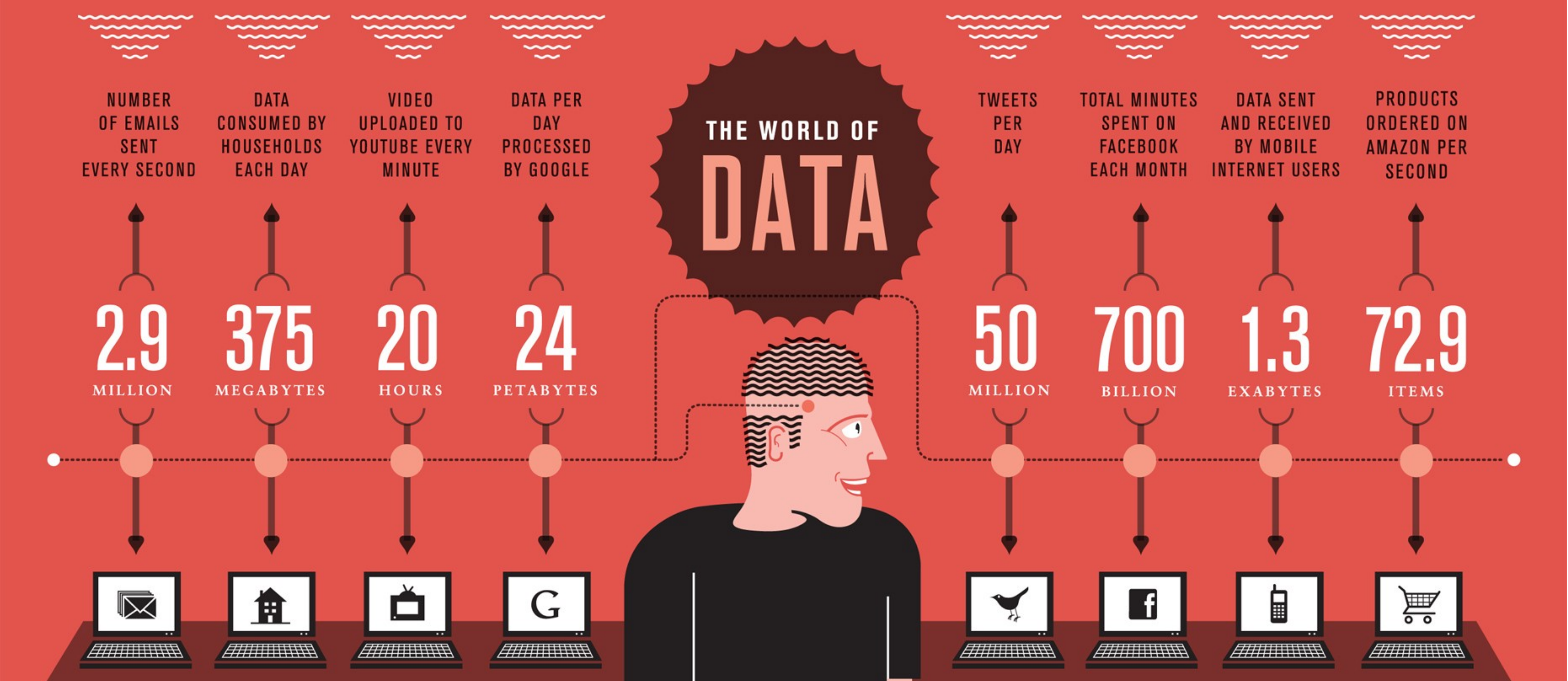
1000	kB	kilobyte
1000^2	MB	megabyte
1000^3	GB	gigabyte
1000^4	TB	terabyte
1000^5	PB	petabyte
1000^6	EB	exabyte
1000^7	ZB	zettabyte
1000^8	YB	yottabyte



http://mozy.com/blog/misc/how-much-is-a-petabyte/

Grande escala?

http://blog.bimeanalytics.com/english/world-of-data-infographic



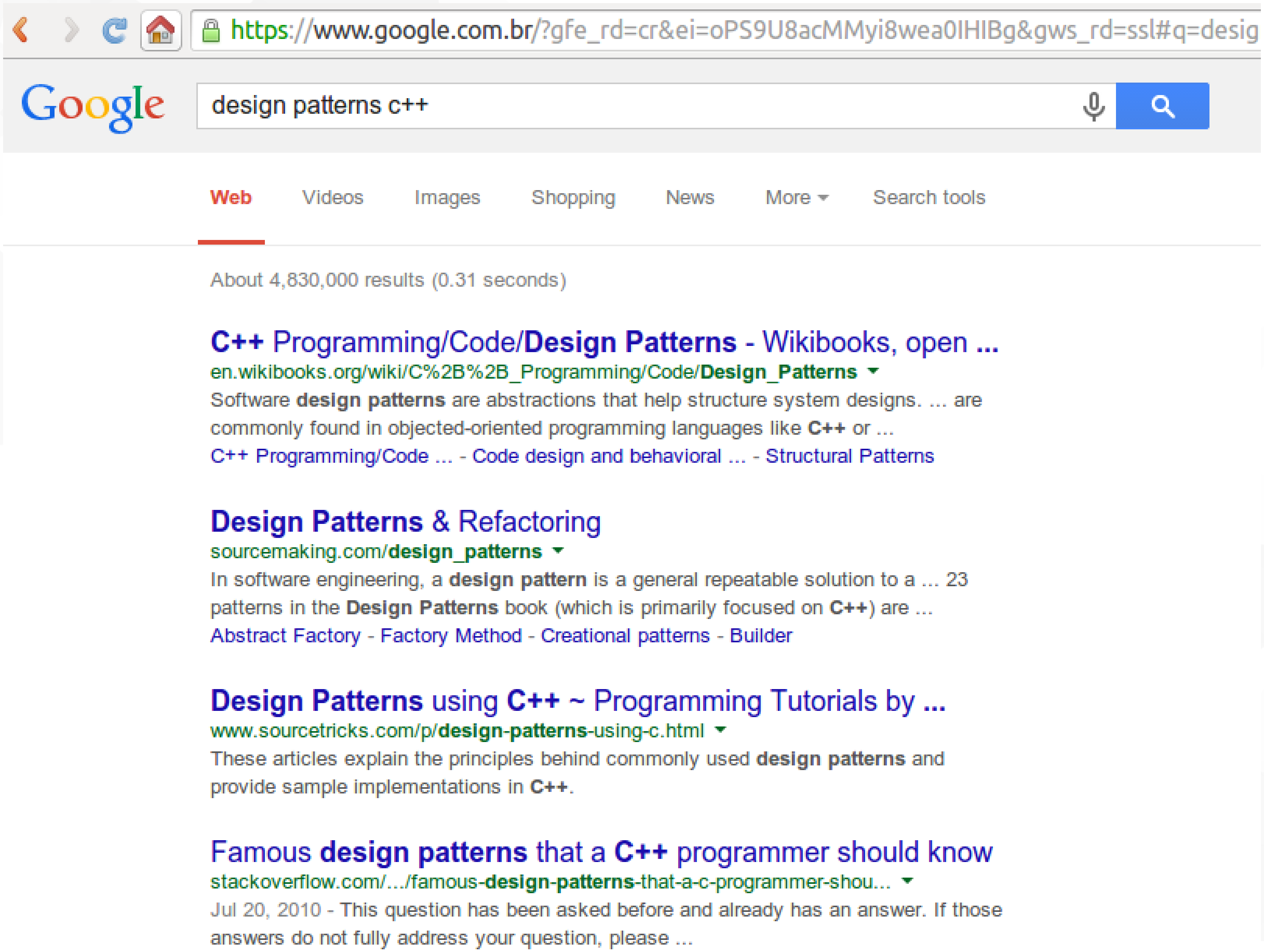
Em vez de a ciência não avançar devido à escassez de dados, hoje em dia ela frequentemente encontra **dificuldades em avançar por seu excesso**.

Roberto M. Cesar-Jr (IME/USP)

A COLLABORATION BETWEEN GOOD AND OLIVER MUNDAY

IN PARTNERSHIP WITH IBM

Uma aplicação



Uma aplicação

Facilitar a busca de informação na web com as máquinas de busca:

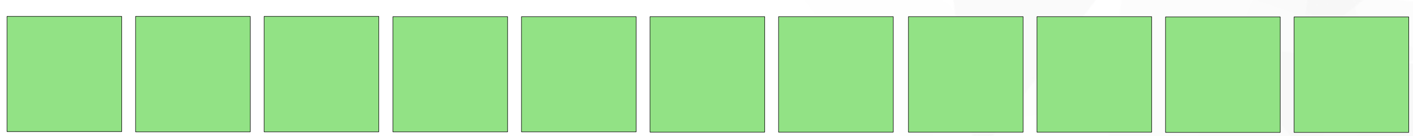
- É comum uma consulta na web retornar centenas de milhares de documentos relacionados com a consulta.
- O usuário está interessado em apenas os ***k*** mais relevantes.
- Em geral ***k<200*** documentos.
- Normalmente são consultados os 10 primeiros documentos.

Assim, são necessários algoritmos de ordenação parcial

Ordenação parcial

Ordenação parcial (seleção do k-éssimo maior)

- Consiste em obter os ***k*** primeiros elementos de um vetor ordenado com ***n*** elementos.
- Quando ***k=1*** o problema se reduz a encontrar o mínimo (ou o máximo) de um conjunto de elementos.
- Quando ***k=n*** caímos no problema clássico de ordenação.



Ordenação parcial

Os algoritmos de Ord. Parcial que estudaremos serão:

- Seleção parcial
- Inserção parcial
- Heapsort parcial
- Quicksort parcial

Seleção parcial

```
void SelectionSort (int v[], int n) {
    int i, j, iMin, aux;

    for (i=0; i<n-1; i++) {
        iMin = i;

        for (j=i+1; j<n; j++) {
            if (v[iMin]>v[j])
                iMin = j;
        }

        if (iMin!=i) {
            aux = v[iMin];
            v[iMin] = v[i];
            v[i] = aux;
        }
    }
}
```

Animação: https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html

Seleção parcial

- Um dos algoritmos mais simples.

- Principio de funcionamento:
  - Selecione o menor item do vetor.
  - Troque-o com o item que está na primeira posição do vetor.
  - Repita estas duas operações com os itens:

    n-1, n-2, n-3, …, n-(k-1), n-k

Seleção parcial

```
void SelectionSort (int v[], int n) {
    int i, j, iMin, aux;

    for (i=0; i<n-1; i++) {
        iMin = i;

        for (j=i+1; j<n; j++) {
            if (v[iMin]>v[j])
                iMin = j;
        }

        if (iMin!=i) {
            aux = v[iMin];
            v[iMin] = v[i];
            v[i] = aux;
        }
    }
}
```

```
void PartialSelectionSort (int v[], int n, int k) {
    int i, j, iMin, aux;

    for (i=0; i<k; i++) {
        iMin = i;    // indice do i-essimo menor

        for (j=i+1; j<n; j++) {
            if (v[iMin]>v[j])
                iMin = j;
        }

        if (iMin!=i) {
            aux = v[iMin];
            v[iMin] = v[i];
            v[i] = aux;
        }
    }
}
```

Seleção parcial

```
void ImprimeVetor(int v[], int n) {
    int i;
    printf("\n");
    for (i=0; i<n; i++)
        printf("%d ", v[i]);
}

int main()
{
    int v[] = {15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0,-1};
    int n=sizeof(v)/sizeof(v[0]);

    ImprimeVetor(v, n);
    PartialSelectionSort(v, n, 7);
    ImprimeVetor(v, n);
}
```

```
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 -1
-1 0 1 2 3 4 5 8 7 6 9 10 11 12 13 14 15
```

k=7

Seleção parcial

```
void PartialSelectionSort (int v[], int n, int k) {
    int i, j, iMin, aux;

    for (i=0; i<k; i++) {
        iMin = i;    // indice do i-essimo menor

        for (j=i+1; j<n; j++) {
            if (v[iMin]>v[j])
                iMin = j;
        }

        if (iMin!=i) {
            aux = v[iMin];
            v[iMin] = v[i];
            v[i] = aux;
        }
    }
}
```

Identifique o número de:

- Comparações entre elementos

- Movimentações entre registros

Seleção parcial

```c
void PartialSelectionSort (int v[], int n, int k) {
    int i, j, iMin, aux;

    for (i=0; i<k; i++) {
        iMin = i;    // indice do i-essimo menor

        for (j=i+1; j<n; j++) {
            if (v[iMin]>v[j])
                iMin = j;
        }

        if (iMin!=i) {
            aux = v[iMin];
            v[iMin] = v[i];
            v[i] = aux;
        }
    }
}
```

Identifique o número de:

- Comparações entre elementos

$$C(n) = nk - \frac{k^2}{2} - \frac{k}{2}$$

- Movimentações entre registros

$$M(n) = 3k$$

No pior caso

Espetacular: Comportamento linear no tamanho de k!

Seleção parcial

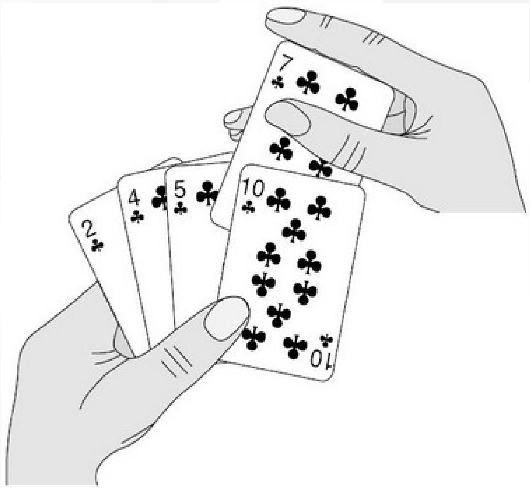
- Este algoritmo é “<u>muito</u>” simples de ser obtido a partir da implementação do **selectionSort**.

- Possui um comportamento espetacular quanto ao número de movimentos de registros:
    - Tempo de execução é linear no tamanho de ***k***.

Inserção parcial

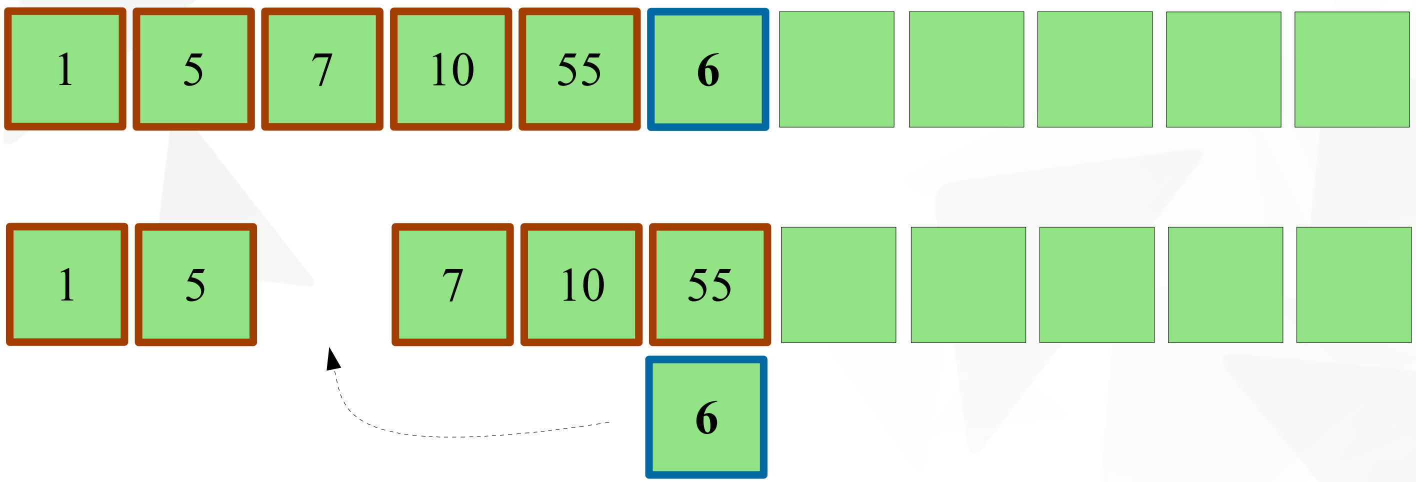
```
void InsertionSort (int v[], int n) {
    int i, j, aux;

    for (i=1; i<n; i++) {
        aux = v[i];

        for (j=i-1; j>=0 && v[j]>aux; j--)
            v[j+1] = v[j];

        v[j+1] = aux;
    }
}
```

InsertionSort



Método preferido dos jogadores de cartas

Em cada passo, a partir do i=1, o I-ésimo elemento da sequência fonte é apanhado e transferido para a sequência destino, sendo inserido no seu lugar apropriado.



Animação: https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html

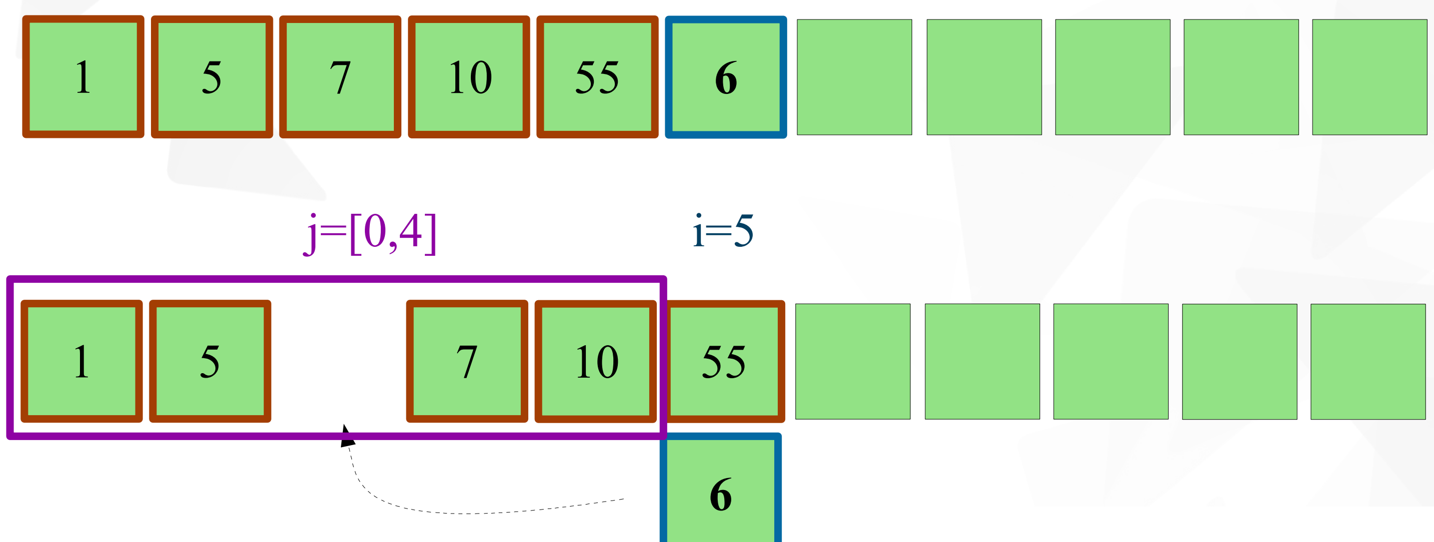
Inserção parcial

Pode ser obtido a partir do algoritmo de ordenação por Inserção por meio de uma modificação:

- Tendo sido ordenado os primeiros k itens, **o item da k-essima posição funciona como um pivô**.
- Quando o item entre os restantes é menor do que o pivô, ele é inserido na posição correta entre os k itens de acordo com o algoritmo original.

Inserção

```
void InsertionSort (int v[], int n) {
    int i, j, aux;

    for (i=1; i<n; i++) {
        aux = v[i];

        for (j=i-1; j>=0 && v[j]>aux; j--)
            v[j+1] = v[j];

        v[j+1] = aux;
    }
}
```



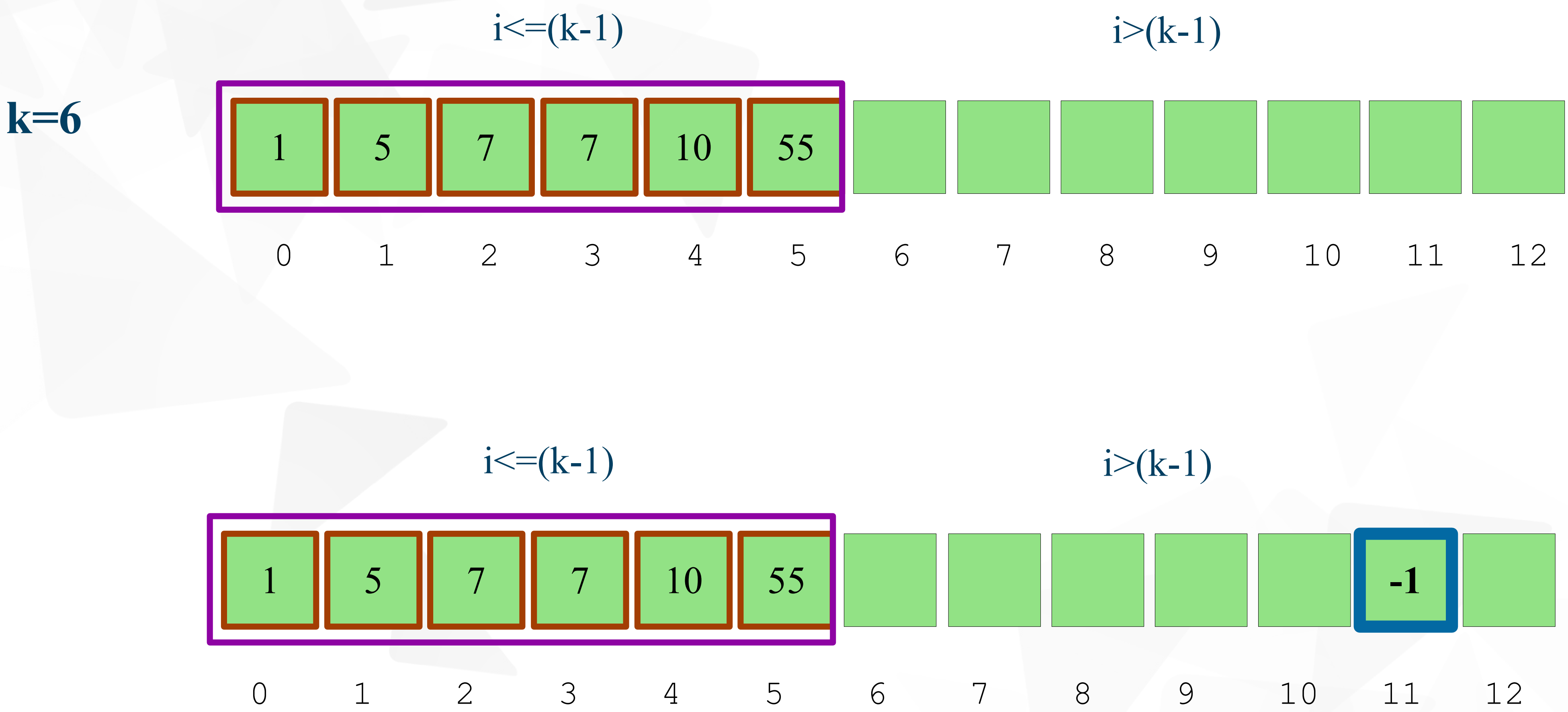
Inserção parcial

i<=(k-1) i>(k-1)

k=6

0	1	2	3	4	5	6	7	8	9	10	11	12
1	5	7	7	10	55							

Inserção parcial



Inserção parcial

```c
void InsertionSort (int v[], int n) {
    int i, j, aux;

    for (i=1; i<n; i++) {
        aux = v[i];

        for (j=i-1; j>=0 && v[j]>aux; j--)
            v[j+1] = v[j];

        v[j+1] = aux;
    }
}
```

```c
void PartialInsertionSort (int v[], int n, int k) {
    int i, j, aux, inicio;

    for (i=1; i<n; i++) {
        aux = v[i];

        if (i>k-1)
            inicio = k-1;
        else
            inicio = i-1;

        for (j=inicio; j>=0 && v[j]>aux; j--)
            v[j+1] = v[j];

        v[j+1] = aux;
    }
}
```

Inserção parcial

```
void PartialInsertionSort (int v[], int n, int k) {
    int i, j, aux, inicio;

    for (i=1; i<n; i++) {
        aux = v[i];

        if (i>k-1)
            inicio = k-1;
        else
            inicio = i-1;

        for (j=inicio; j>=0 && v[j]>aux; j--)
            v[j+1] = v[j];

        v[j+1] = aux;
    }
}
```

Inserção parcial

15	14	13	12	11	10	9	8	7	6	5	4	3	2	10	-1

-1	0	1	2	3	4	5	6	7	6	5	4	3	2	10	-1

k=7

Inserção parcial

```
void PartialInsertionSort (int v[], int n, int k) {
    int i, j, aux, inicio;

    for (i=1; i<n; i++) {
        aux = v[i];

        if (i>k-1)
            inicio = k-1;
        else
            inicio = i-1;

        for (j=inicio; j>=0 && v[j]>aux; j--)
            v[j+1] = v[j];

        v[j+1] = aux;
    }
}
```

- Comparações entre elementos: **(melhor caso e pior caso) ?**

- Movimentações entre registros: **(melhor caso e pior caso) ?**

Inserção parcial

```
void PartialInsertionSort (int v[], int n, int k) {
    int i, j, aux, inicio;

    for (i=1; i<n; i++) {              ← n-1 iterações
        aux = v[i];

        if (i>k-1)
            inicio = k-1;
        else
            inicio = i-1;

        for (j=inicio; j>=0 && v[j]>aux; j--)
            v[j+1] = v[j];

        v[j+1] = aux;
    }
}
```

Comparações entre elementos:

- Melhor caso $C(n) = n - 1$

- Pior caso $C(n) = kn - \frac{k}{2} - \frac{k^2}{2}$

Inserção parcial

```
void PartialInsertionSort (int v[], int n, int k) {
    int i, j, aux, inicio;

    for (i=1; i<n; i++) {
        aux = v[i];

        if (i>k-1)
            inicio = k-1;
        else
            inicio = i-1;

        for (j=inicio; j>=0 && v[j]>aux; j--)
            v[j+1] = v[j];

        v[j+1] = aux;
    }
}
```

← *1 movimentação*

← *1 movimentação*

← *1 movimentação*

Movimentações entre elementos:

- **Melhor caso** $M(n) = 2(n-1)$
- **Pior caso** $M(n) = 2(n-1) + C(n) = n(k+2) - \frac{k^2}{2} - \frac{k}{2} - 2$

Inserção parcial

15	14	13	12	11	10	9	8	7	6	5	4	3	2	10	-1
14	15	13	12	11	10	9	8	7	6	5	4	3	2	10	-1
13	14	15	12	11	10	9	8	7	6	5	4	3	2	10	-1
12	13	14	15	11	10	9	8	7	6	5	4	3	2	10	-1
11	12	13	14	15	10	9	8	7	6	5	4	3	2	10	-1
10	11	12	13	14	15	9	8	7	6	5	4	3	2	10	-1
9	10	11	12	13	14	15	8	7	6	5	4	3	2	10	-1
8	9	10	11	12	13	14	15	7	6	5	4	3	2	10	-1
7	8	9	10	11	12	13	14	7	6	5	4	3	2	10	-1
6	7	8	9	10	11	12	13	7	6	5	4	3	2	10	-1
5	6	7	8	9	10	11	12	7	6	5	4	3	2	10	-1
4	5	6	7	8	9	10	11	7	6	5	4	3	2	10	-1
3	4	5	6	7	8	9	10	7	6	5	4	3	2	10	-1
2	3	4	5	6	7	8	9	7	6	5	4	3	2	10	-1
1	2	3	4	5	6	7	8	7	6	5	4	3	2	10	-1
0	1	2	3	4	5	6	7	7	6	5	4	3	2	10	-1
-1	0	1	2	3	4	5	6	7	6	5	4	3	2	10	-1

-1	0	1	2	3	4	5	6	7	6	5	4	3	2	10	-1

k=7

Inserção parcial

15	14	13	12	11	10	9	8	7	6	5	4	3	2	10	-1
14	15	13	12	11	10	9	8	7	6	5	4	3	2	10	-1
13	14	15	12	11	10	9	8	7	6	5	4	3	2	10	-1
12	13	14	15	11	10	9	8	7	6	5	4	3	2	10	-1
11	12	13	14	15	10	9	8	7	6	5	4	3	2	10	-1
10	11	12	13	14	15	9	8	7	6	5	4	3	2	10	-1
9	10	11	12	13	14	15	8	7	6	5	4	3	2	10	-1
8	9	10	11	12	13	14	15	7	6	5	4	3	2	10	-1
7	8	9	10	11	12	13	14	7	6	5	4	3	2	10	-1
6	7	8	9	10	11	12	13	7	6	5	4	3	2	10	-1
5	6	7	8	9	10	11	12	7	6	5	4	3	2	10	-1
4	5	6	7	8	9	10	11	7	6	5	4	3	2	10	-1
3	4	5	6	7	8	9	10	7	6	5	4	3	2	10	-1
2	3	4	5	6	7	8	9	7	6	5	4	3	2	10	-1
1	2	3	4	5	6	7	8	7	6	5	4	3	2	10	-1
0	1	2	3	4	5	6	7	7	6	5	4	3	2	10	-1
-1	0	1	2	3	4	5	6	7	6	5	4	3	2	10	-1

-1	0	1	2	3	4	5	6	7	6	5	4	3	2	10	-1

k=7

O algoritmo não preserva o restante do vetor

Inserção parcial

```
void PartialInsertionSort (int v[], int n, int k) {
    int i, j, aux, inicio;

    for (i=1; i<n; i++) {
        aux = v[i];

        if (i>k-1)
            inicio = k-1;
        else
            inicio = i-1;

        for (j=inicio; j>=0 && v[j]>aux; j--)
            v[j+1] = v[j];

        v[j+1] = aux;
    }
}
```

Modifique o algoritmo para preservar todos os elementos do vetor

Inserção parcial 2 (preserva o restante do vetor)

```
void PartialInsertionSort2 (int v[], int n, int k) {
    int i, j, aux, inicio;

    for (i=1; i<n; i++) {
        aux = v[i];

        if (i>k-1) {
            inicio = k-1;
            if (v[i]<v[k])
                v[i] = v[k];
        }
        else
            inicio = i-1;

        for (j=inicio; j>=0 && v[j]>aux; j--)
            v[j+1] = v[j];

        v[j+1] = aux;
    }
}
```

Inserção parcial 2 (preserva o restante do vetor)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-1
14	15	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-1
13	14	15	12	11	10	9	8	7	6	5	4	3	2	1	0	-1
12	13	14	15	11	10	9	8	7	6	5	4	3	2	1	0	-1
11	12	13	14	15	10	9	8	7	6	5	4	3	2	1	0	-1
10	11	12	13	14	15	9	8	7	6	5	4	3	2	1	0	-1
9	10	11	12	13	14	15	8	7	6	5	4	3	2	1	0	-1
8	9	10	11	12	13	14	15	7	6	5	4	3	2	1	0	-1
7	8	9	10	11	12	13	14	15	6	5	4	3	2	1	0	-1
6	7	8	9	10	11	12	13	15	14	5	4	3	2	1	0	-1
5	6	7	8	9	10	11	12	15	14	13	4	3	2	1	0	-1
4	5	6	7	8	9	10	11	15	14	13	12	3	2	1	0	-1
3	4	5	6	7	8	9	10	15	14	13	12	11	2	1	0	-1
2	3	4	5	6	7	8	9	15	14	13	12	11	10	1	0	-1
1	2	3	4	5	6	7	8	15	14	13	12	11	10	9	0	-1
0	1	2	3	4	5	6	7	15	14	13	12	11	10	9	8	-1
-1	0	1	2	3	4	5	6	15	14	13	12	11	10	9	8	7

-1	0	1	2	3	4	5	6	15	14	13	12	11	10	9	8	7

$k=7$

Inserção parcial 2 (preserva o restante do vetor)

Versão 1

```
-1   0   1   2   3   4   5   6   7   6   5   4   3   2  10   -1
```

Versão 2

```
-1   0   1   2   3   4   5   6  15  14  13  12  11  10   9   8   7
```

Heapsort parcial

Inserção parcial

Utiliza um tipo abstrato de dados min-heap para informar o menor item do conjunto.

Usando um MIN-HEAP

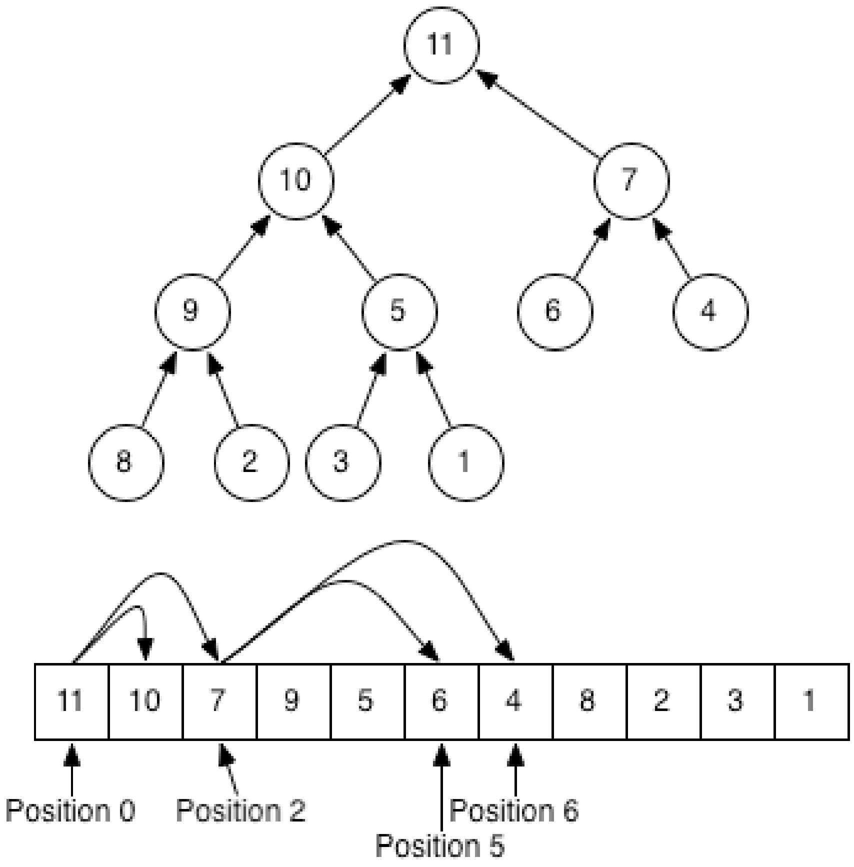
*Na primeira iteração, o **menor** item que está em A[0] (raiz do heap) é trocado com o item que está em A[n-1].*

Em seguida o heap é refeito.

*Novamente o **menor** está em A[0], troque-o com A[n-1].*

Repita as duas últimas operações até que o k-ésimo menor esteja seja trocado com A[n-k].

Ao final, os k menores estão nas k últimas posições do vetor A.



Animação: https://www.cs.usfca.edu/~galles/visualization/HeapSort.html

Inserção parcial

- O heapsort-Parcial deve construir o heap a um custo O(n).
- O prodecimento Refaz (arruma o heap) tem um custo de O(lg(n)).
- O procedimento heapsort parcial chama o procedimento anterior k vezes.
- Complexidade:

$$O(n + k \log n)$$

Inserção parcial

- O heapsort-Parcial deve construir o heap a um custo O(n).
- O prodecimento Refaz (arruma o heap) tem um custo de O(lg(n)).
- O procedimento heapsort parcial chama o procedimento anterior k vezes.
- Complexidade:

$$O(n + k \log n) = \begin{cases} O(n) & \text{se } k \leq \frac{n}{\log n} \\ O(k \log n) & \text{se } k > \frac{n}{\log n} \end{cases}$$

Quicksort parcial

Lista 01
Envio até 16/06 (23h50-Tidia)

- Implemente o algoritmo Quicksort parcial:
  http://www2.dcc.ufmg.br/livros/algoritmos-edicao2/cap4/transp/completo1/cap4.pdf

- Seu programa não deve impor limitações sobre o número de elementos (n), nem o valor de k. Para simplificar, os números do vetor estão na base 10.
- Apresentação livre de exemplos (quanto mais completo melhor).
- Pode ser elaborado por até 2 alunos.

- Apenas 2 arquivos que deverá submeter pelo Tidia:
  - Código fonte em C/C++ (PartialQuickSort.c/cpp)
  - Um PDF contendo uma simples descrição do programa (não maior a 4 páginas). O formato desse relatório é livre.

Comparação empírica dos algoritmos

n, k	Seleção	Quicksort	Inserção	Inserção2	Heapsort
$n: 10^1 \quad k: 10^0$	1	2,5	1	1,2	1,7
$n: 10^1 \quad k: 10^1$	1,2	2,8	1	1,1	2,8
$n: 10^2 \quad k: 10^0$	1	3	1,1	1,4	4,5
$n: 10^2 \quad k: 10^1$	1,9	2,4	1	1,2	3
$n: 10^2 \quad k: 10^2$	3	1,7	1	1,1	2,3
$n: 10^3 \quad k: 10^0$	1	3,7	1,4	1,6	9,1
$n: 10^3 \quad k: 10^1$	4,6	2,9	1	1,2	6,4
$n: 10^3 \quad k: 10^2$	11,2	1,3	1	1,4	1,9
$n: 10^3 \quad k: 10^3$	15,1	1	3,9	4,2	1,6
$n: 10^5 \quad k: 10^0$	1	2,4	1,1	1,1	5,3
$n: 10^5 \quad k: 10^1$	5,9	2,2	1	1	4,9
$n: 10^5 \quad k: 10^2$	67	2,1	1	1,1	4,8
$n: 10^5 \quad k: 10^3$	304	1	1,1	1,3	2,3
$n: 10^5 \quad k: 10^4$	1445	1	33,1	43,3	1,7
$n: 10^5 \quad k: 10^5$	∞	1	∞	∞	1,9
$n: 10^6 \quad k: 10^0$	1	3,9	1,2	1,3	8,1
$n: 10^6 \quad k: 10^1$	6,6	2,7	1	1	7,3
$n: 10^6 \quad k: 10^2$	83,1	3,2	1	1,1	6,6
$n: 10^6 \quad k: 10^3$	690	2,2	1	1,1	5,7
$n: 10^6 \quad k: 10^4$	∞	1	5	6,4	1,9
$n: 10^6 \quad k: 10^5$	∞	1	∞	∞	1,7
$n: 10^6 \quad k: 10^6$	∞	1	∞	∞	1,8
$n: 10^7 \quad k: 10^0$	1	3,4	1,1	1,1	7,4
$n: 10^7 \quad k: 10^1$	8,6	2,6	1	1,1	6,7
$n: 10^7 \quad k: 10^2$	82,1	2,6	1	1,1	6,8
$n: 10^7 \quad k: 10^3$	∞	3,1	1	1,1	6,6
$n: 10^7 \quad k: 10^4$	∞	1,1	1	1,2	2,6
$n: 10^7 \quad k: 10^5$	∞	1	∞	∞	2,2
$n: 10^7 \quad k: 10^6$	∞	1	∞	∞	1,2
$n: 10^7 \quad k: 10^7$	∞	1	∞	∞	1,7
	Seleção	Quicksort	Inserção	Inserção2	Heapsort

Fonte:http://www2.dcc.ufmg.br/livros/algoritmos-edicao2/cap4/transp/completo1/cap4.pdf