

```

int main(int argc, char** argv)
{
    char c = 0;
    char* commands = "ads pq"; // key commands: "left,right,rotate,confirm,pause,quit"
    int speed = 2; // sets max moves per row
    int moves_to_go = 2;
    int full = 0; // whether board is full
    init(); // initialize board an tetrominoes

```

MAC122 - PRINCÍPIOS DE DESENVOLVIMENTO DE ALGORITMOS

Algoritmos de ordenação

```

// process user action
c = getchar(); // get new action
if (c == commands[0] && !intersect(cur, state[0]-1, state[1])) state[0]--; // move left
if (c == commands[1] && !intersect(cur, state[0]+1, state[1])) state[0]++; // move right
if (c == commands[2] && !intersect(cur->rotated, state[0], state[1])) cur = cur->rotated;
if (c == commands[3]) moves_to_go=0;

// scroll down
if (!moves_to_go--)
{
    if (intersect(cur,state[0],state[1]+1)) // if tetromino intersected with sth
    {
        cramp_tetromino();
        remove_complete_lines();
        cur = &tetrominoes[rand() % NUM_POSES];
        state[0] = (WIDTH - cur->width)/2;

```

Exercício: Prove que o seguinte algoritmo é correto

```
1  /* Rearranja um vetor v[0..n-1] de inteiros para que os
2  * pares ocorram antes dos ímpares */
3  void Rearranja (int n, int v[]) {
4      int x, j = 0;
5      for (int i = 0; i < n; i++) /* Invariante */
6          if (v[i] % 2 == 0) {
7              x = v[i]; v[i] = v[j]; v[j] = x;
8              j++;
9          }
10 }
```

Invariante de laço:

1. $v[0..n-1]$ é permutação do vetor original
2. $v[0..j-1]$ contém apenas números pares;
3. $v[j..i-1]$ contém apenas ímpares.

```
1 void Rearranja (int n, int v[]) {  
2     int x, j = 0;  
3     for (int i = 0; i < n; i++)  
4         if (v[i] % 2 == 0) {  
5             x = v[i]; v[i] = v[j]; v[j] = x;  
6             j++;  
7         }  
8 }
```

Prova de corretude (assumindo $n > 0$ pares e $n > 0$ ímpares em v):

1. **Base:** Para $i = j = 0$, resultado é trivialmente verdade .

```

1 void Rearranja (int n, int v[]) {
2     int x, j = 0;
3     for (int i = 0; i < n; i++)
4         if (v[i] % 2 == 0) {
5             x = v[i]; v[i] = v[j]; v[j] = x;
6             j++;
7         }
8 }

```

Prova de corretude (assumindo $n > 0$ pares e $n > 0$ ímpares em v):

1. **Base:** Para $i = j = 0$, resultado é trivialmente verdade .
2. **Indução:** Assuma que invariante é verdadeira. Se $v[i]$ for ímpar, vetor não é alterado e invariante permanece verdadeira. Se $v[i]$ for par, trocamos $v[j]$ (que é ímpar por hipótese) e $v[i]$ de lugar, fazendo que $v[0..j]$ seja par e $v[j+1..i-1]$, ímpar.

```

1 void Rearranja (int n, int v[]) {
2     int x, j = 0;
3     for (int i = 0; i < n; i++)
4         if (v[i] % 2 == 0) {
5             x = v[i]; v[i] = v[j]; v[j] = x;
6             j++;
7         }
8 }

```

Prova de corretude (assumindo $n > 0$ pares e $n > 0$ ímpares em v):

1. **Base:** Para $i = j = 0$, resultado é trivialmente verdade .
2. **Indução:** Assuma que invariante é verdadeira. Se $v[i]$ for ímpar, vetor não é alterado e invariante permanece verdadeira. Se $v[i]$ for par, trocamos $v[j]$ (que é ímpar por hipótese) e $v[i]$ de lugar, fazendo que $v[0..j]$ seja par e $v[j+1..i-1]$, ímpar.
3. **Término:** Invariante para $i=n$: $v[0..j-1]$ é par e $v[j..n-1]$ é ímpar, onde j é o número de pares em $v[0..n-1]$

Problema da ordenação

Rearranjar os elementos de uma lista x_1, \dots, x_n de tal modo que fiquem em ordem não decrescente: $x_1 \leq x_2 \leq \dots \leq x_n$

56	25	37	58	95	19	73	30
----	----	----	----	----	----	----	----



19	25	30	37	56	58	73	95
----	----	----	----	----	----	----	----

Problema da ordenação

Rearranjar os elementos de uma lista x_1, \dots, x_n de tal modo que fiquem em ordem não decrescente: $x_1 \leq x_2 \leq \dots \leq x_n$

56	25	37	58	95	19	73	30
----	----	----	----	----	----	----	----



19	25	30	37	56	58	73	95
----	----	----	----	----	----	----	----

- ▶ Elementos x_i são **genéricos** (inteiros, reais, strings, registros) porém comparáveis, isto é, $x_i \leq x_j$ é bem definido para quaisquer elementos
- ▶ Lista é representada como **vetor de tamanho fixo**

Terminologia

Vetor $v[0..n-1]$ é crescente se $v[0] \leq v[1] \leq \dots \leq v[n-1]$

19	25	25	30	30	56	56	95
----	----	----	----	----	----	----	----

- ▶ Por seleção
- ▶ Por inserção
- ▶ Bolha (bubblesort)
- ▶ ...

Ordenação por seleção

Percorra a lista da esquerda para a direita, e a cada passo faça:

1. Encontre o menor elemento à direita da posição atual
2. Troque o elemento do item anterior com o elemento da posição atual

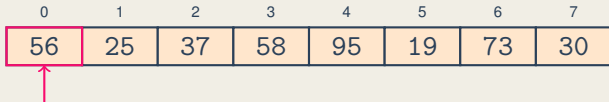
0	1	2	3	4	5	6	7
56	25	37	58	95	19	73	30

Ordenação por seleção

Percorra a lista da esquerda para a direita, e a cada passo faça:

1. Encontre o menor elemento à direita da posição atual
2. Troque o elemento do item anterior com o elemento da posição atual

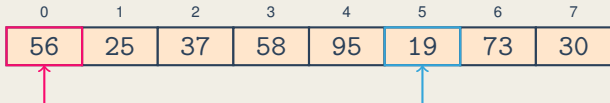
0	1	2	3	4	5	6	7
56	25	37	58	95	19	73	30



Ordenação por seleção

Percorra a lista da esquerda para a direita, e a cada passo faça:

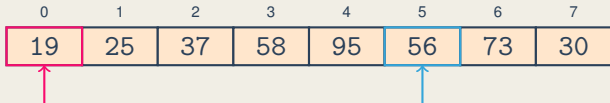
1. Encontre o menor elemento à direita da posição atual
2. Troque o elemento do item anterior com o elemento da posição atual



Ordenação por seleção

Percorra a lista da esquerda para a direita, e a cada passo faça:

1. Encontre o menor elemento à direita da posição atual
2. Troque o elemento do item anterior com o elemento da posição atual

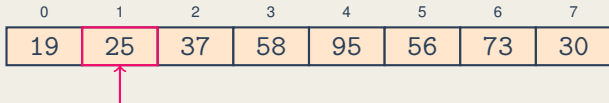


Ordenação por seleção

Percorra a lista da esquerda para a direita, e a cada passo faça:

1. Encontre o menor elemento à direita da posição atual
2. Troque o elemento do item anterior com o elemento da posição atual

0	1	2	3	4	5	6	7
19	25	37	58	95	56	73	30

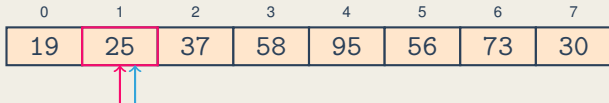


Ordenação por seleção

Percorra a lista da esquerda para a direita, e a cada passo faça:

1. Encontre o menor elemento à direita da posição atual
2. Troque o elemento do item anterior com o elemento da posição atual

0	1	2	3	4	5	6	7
19	25	37	58	95	56	73	30

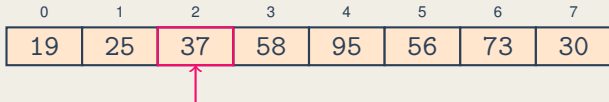


Ordenação por seleção

Percorra a lista da esquerda para a direita, e a cada passo faça:

1. Encontre o menor elemento à direita da posição atual
2. Troque o elemento do item anterior com o elemento da posição atual

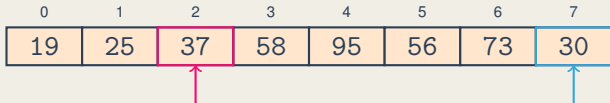
0	1	2	3	4	5	6	7
19	25	37	58	95	56	73	30



Ordenação por seleção

Percorra a lista da esquerda para a direita, e a cada passo faça:

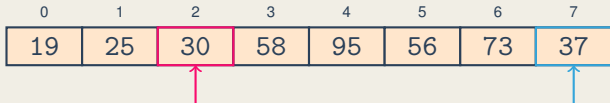
1. Encontre o menor elemento à direita da posição atual
2. Troque o elemento do item anterior com o elemento da posição atual



Ordenação por seleção

Percorra a lista da esquerda para a direita, e a cada passo faça:

1. Encontre o menor elemento à direita da posição atual
2. Troque o elemento do item anterior com o elemento da posição atual




Ordenação por seleção

Percorra a lista da esquerda para a direita, e a cada passo faça:

1. Encontre o menor elemento à direita da posição atual
2. Troque o elemento do item anterior com o elemento da posição atual

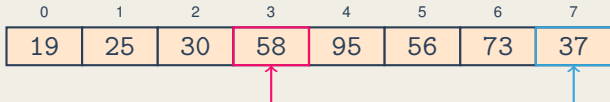
0	1	2	3	4	5	6	7
19	25	30	58	95	56	73	37



Ordenação por seleção

Percorra a lista da esquerda para a direita, e a cada passo faça:

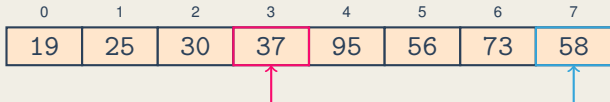
1. Encontre o menor elemento à direita da posição atual
2. Troque o elemento do item anterior com o elemento da posição atual



Ordenação por seleção

Percorra a lista da esquerda para a direita, e a cada passo faça:

1. Encontre o menor elemento à direita da posição atual
2. Troque o elemento do item anterior com o elemento da posição atual

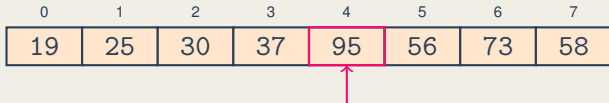


Ordenação por seleção

Percorra a lista da esquerda para a direita, e a cada passo faça:

1. Encontre o menor elemento à direita da posição atual
2. Troque o elemento do item anterior com o elemento da posição atual

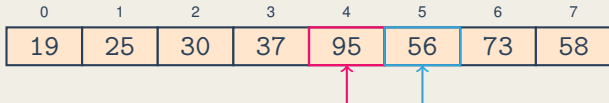
0	1	2	3	4	5	6	7
19	25	30	37	95	56	73	58



Ordenação por seleção

Percorra a lista da esquerda para a direita, e a cada passo faça:

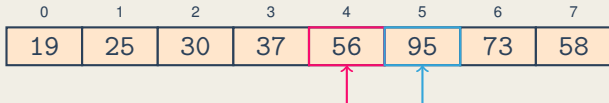
1. Encontre o menor elemento à direita da posição atual
2. Troque o elemento do item anterior com o elemento da posição atual



Ordenação por seleção

Percorra a lista da esquerda para a direita, e a cada passo faça:

1. Encontre o menor elemento à direita da posição atual
2. Troque o elemento do item anterior com o elemento da posição atual

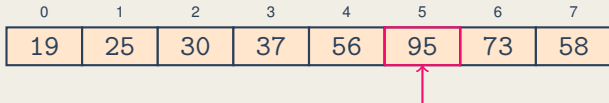


Ordenação por seleção

Percorra a lista da esquerda para a direita, e a cada passo faça:

1. Encontre o menor elemento à direita da posição atual
2. Troque o elemento do item anterior com o elemento da posição atual

0	1	2	3	4	5	6	7
19	25	30	37	56	95	73	58



Ordenação por seleção

Percorra a lista da esquerda para a direita, e a cada passo faça:

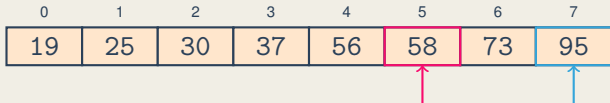
1. Encontre o menor elemento à direita da posição atual
2. Troque o elemento do item anterior com o elemento da posição atual

0	1	2	3	4	5	6	7
19	25	30	37	56	95	73	58

Ordenação por seleção

Percorra a lista da esquerda para a direita, e a cada passo faça:

1. Encontre o menor elemento à direita da posição atual
2. Troque o elemento do item anterior com o elemento da posição atual

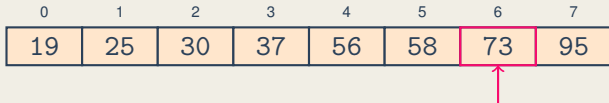


Ordenação por seleção

Percorra a lista da esquerda para a direita, e a cada passo faça:

1. Encontre o menor elemento à direita da posição atual
2. Troque o elemento do item anterior com o elemento da posição atual

0	1	2	3	4	5	6	7
19	25	30	37	56	58	73	95



Ordenação por seleção

Percorra a lista da esquerda para a direita, e a cada passo faça:

1. Encontre o menor elemento à direita da posição atual
2. Troque o elemento do item anterior com o elemento da posição atual

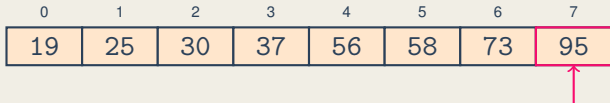
0	1	2	3	4	5	6	7
19	25	30	37	56	58	73	95

Ordenação por seleção

Percorra a lista da esquerda para a direita, e a cada passo faça:

1. Encontre o menor elemento à direita da posição atual
2. Troque o elemento do item anterior com o elemento da posição atual

0	1	2	3	4	5	6	7
19	25	30	37	56	58	73	95



Ordenação por seleção

Percorra a lista da esquerda para a direita, e a cada passo faça:

1. Encontre o menor elemento à direita da posição atual
2. Troque o elemento do item anterior com o elemento da posição atual

```
1 void Seleção (int v[], int n) {  
2  
3     /* ...? */  
4  
5 }
```

Ordenação por seleção

Percorra a lista da esquerda para a direita, e a cada passo faça:

1. Encontre o menor elemento à direita da posição atual
2. Troque o elemento do item anterior com o elemento da posição atual

```
1 void Seleção (int v[], int n) {  
2     int i, j, k, x;  
3     for (i = 0; i < n; i++) {  
4         /* Passo 1 */  
5         for (k = i, j = i + 1; j < n; j++)  
6             if (v[j] < v[k]) k = j;  
7         /* Passo 2 */  
8         x = v[i]; v[i] = v[k]; v[k] = x;  
9     }  
10 }
```


Ordenação por seleção

```
1 void Seleção (int v[], int n) {  
2     int i, j, k, x;  
3     for (i = 0; i < n; i++) { /* Invariante */  
4         for (k = i, j = i + 1; j < n; j++)  
5             if (v[j] < v[k]) k = j;  
6         x = v[i]; v[i] = v[k]; v[k] = x;  
7     }  
8 }
```

Invariantes

Ordenação por seleção

```
1 void Seleção (int v[], int n) {  
2     int i, j, k, x;  
3     for (i = 0; i < n; i++) { /* Invariante */  
4         for (k = i, j = i + 1; j < n; j++)  
5             if (v[j] < v[k]) k = j;  
6         x = v[i]; v[i] = v[k]; v[k] = x;  
7     }  
8 }
```

Invariantes

- ▶ $v[0..n-1]$ é uma permutação do vetor original
- ▶ $v[0..i-1]$ é crescente
- ▶ $v[i-1] \leq v[j]$ para $j \geq i$

```
1 void Seleção (int v[], int n) {  
2     int i, j, k, x;  
3     for (i = 0; /* Invariante */ i < n; i++) {  
4         for (k = i, j = i + 1; j < n; j++)  
5             if (v[j] < v[k]) k = j;  
6         x = v[i]; v[i] = v[k]; v[k] = x;  
7     }  
8 }
```

Corretude

1. Base: Para $i = 0$, $v[0..n-1]$ é (permutação do) vetor original e $v[0]$ crescente.

```

1 void Seleção (int v[], int n) {
2     int i, j, k, x;
3     for (i = 0; /* Invariante */ i < n; i++) {
4         for (k = i, j = i + 1; j < n; j++)
5             if (v[j] < v[k]) k = j;
6         x = v[i]; v[i] = v[k]; v[k] = x;
7     }
8 }

```

Corretude

1. Base: Para $i = 0$, $v[0..n-1]$ é (permutação do) vetor original e $v[0]$ crescente.
2. Indução: Se $v[0..i-1]$ é crescente e $v[i-1] \leq v[i..n-1]$, iteração encontra $x = v[k] \leq v[i..n-1]$ e portanto $v[0..i-1] \leq x \leq v[i+1..n-1]$

```

1 void Seleção (int v[], int n) {
2     int i, j, k, x;
3     for (i = 0; /* Invariante */ i < n; i++) {
4         for (k = i, j = i + 1; j < n; j++)
5             if (v[j] < v[k]) k = j;
6         x = v[i]; v[i] = v[k]; v[k] = x;
7     }
8 }

```

Corretude

1. Base: Para $i = 0$, $v[0..n-1]$ é (permutação do) vetor original e $v[0]$ crescente.
2. Indução: Se $v[0..i-1]$ é crescente e $v[i-1] \leq v[i..n-1]$, iteração encontra $x = v[k] \leq v[i..n-1]$ e portanto $v[0..i-1] \leq x \leq v[i+1..n-1]$
3. Final: $v[0..n-1]$ é permutação crescente

Ordenação por seleção

```
1 void Seleção (int v[], int n) {  
2     int i, j, k, x;  
3     for (i = 0; i < n; i++) {  
4         for (k = i, j = i + 1; j < n; j++)  
5             if (v[j] < v[k]) k = j;  
6         x = v[i]; v[i] = v[k]; v[k] = x;  
7     }  
8 }
```

Complexidade de tempo:

Ordenação por seleção

```
1 void Seleção (int v[], int n) {  
2     int i, j, k, x;  
3     for (i = 0; i < n; i++) {  
4         for (k = i, j = i + 1; j < n; j++)  
5             if (v[j] < v[k]) k = j;  
6         x = v[i]; v[i] = v[k]; v[k] = x;  
7     }  
8 }
```

Complexidade de tempo:

- ▶ Modelo de execução: qtd. de comparações $v[i] < v[k]$
- ▶ Tempo: $(n - 1) + (n - 2) + \dots + 1 = n(n - 1)/2$ comparações

Algoritmo de ordenação é **estável** se não altera a posição relativa de elementos de mesmo valor

56	25	37	25	95	19	73	30
----	----	----	----	----	----	----	----



19	25	25	30	37	56	73	95
----	----	----	----	----	----	----	----



19	25	25	30	37	56	73	95
----	----	----	----	----	----	----	----

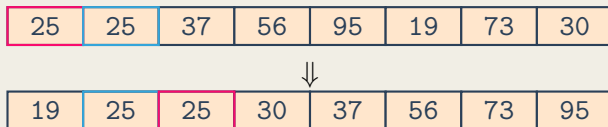
Ordenação por seleção é estável?

```
1 void Seleção (int v[], int n) {  
2     int i, j, k, x;  
3     for (i = 0; i < n; i++) {  
4         for (k = i, j = i + 1; j < n; j++)  
5             if (v[j] < v[k]) k = j;  
6         x = v[i]; v[i] = v[k]; v[k] = x;  
7     }  
8 }
```

Ordenação por seleção é estável?

```
1 void Seleção (int v[], int n) {  
2     int i, j, k, x;  
3     for (i = 0; i < n; i++) {  
4         for (k = i, j = i + 1; j < n; j++)  
5             if (v[j] < v[k]) k = j;  
6         x = v[i]; v[i] = v[k]; v[k] = x;  
7     }  
8 }
```

Não:



Ordenação por inserção

Para $j=1, \dots, n-1$ faça:

1. Seja $x = v[j]$ e $i = j - 1$
2. Enquanto $v[i] > x$ faça:
 - 2.1 Desloque $v[i]$ para a direita
 - 2.2 Defina $i = i - 1$
3. Defina $v[i+1] = x$

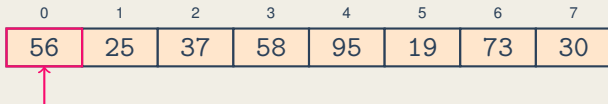
0	1	2	3	4	5	6	7
56	25	37	58	95	19	73	30

Ordenação por inserção

Para $j=1, \dots, n-1$ faça:

1. Seja $x = v[j]$ e $i = j - 1$
2. Enquanto $v[i] > x$ faça:
 - 2.1 Desloque $v[i]$ para a direita
 - 2.2 Defina $i = i - 1$
3. Defina $v[i+1] = x$

0	1	2	3	4	5	6	7
56	25	37	58	95	19	73	30

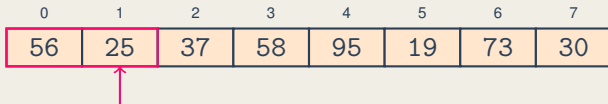


Ordenação por inserção

Para $j=1, \dots, n-1$ faça:

1. Seja $x = v[j]$ e $i = j - 1$
2. Enquanto $v[i] > x$ faça:
 - 2.1 Desloque $v[i]$ para a direita
 - 2.2 Defina $i = i - 1$
3. Defina $v[i+1] = x$

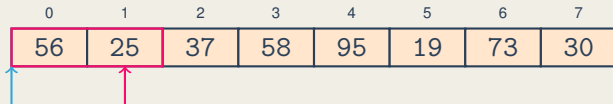
0	1	2	3	4	5	6	7
56	25	37	58	95	19	73	30



Ordenação por inserção

Para $j=1, \dots, n-1$ faça:

1. Seja $x = v[j]$ e $i = j - 1$
2. Enquanto $v[i] > x$ faça:
 - 2.1 Desloque $v[i]$ para a direita
 - 2.2 Defina $i = i - 1$
3. Defina $v[i+1] = x$

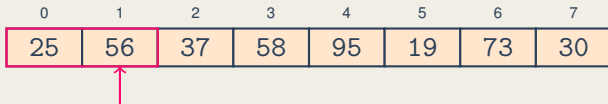


Ordenação por inserção

Para $j=1, \dots, n-1$ faça:

1. Seja $x = v[j]$ e $i = j - 1$
2. Enquanto $v[i] > x$ faça:
 - 2.1 Desloque $v[i]$ para a direita
 - 2.2 Defina $i = i - 1$
3. Defina $v[i+1] = x$

0	1	2	3	4	5	6	7
25	56	37	58	95	19	73	30

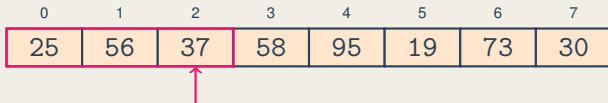


Ordenação por inserção

Para $j=1, \dots, n-1$ faça:

1. Seja $x = v[j]$ e $i = j - 1$
2. Enquanto $v[i] > x$ faça:
 - 2.1 Desloque $v[i]$ para a direita
 - 2.2 Defina $i = i - 1$
3. Defina $v[i+1] = x$

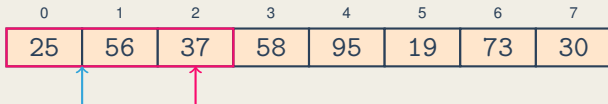
0	1	2	3	4	5	6	7
25	56	37	58	95	19	73	30



Ordenação por inserção

Para $j=1, \dots, n-1$ faça:

1. Seja $x = v[j]$ e $i = j - 1$
2. Enquanto $v[i] > x$ faça:
 - 2.1 Desloque $v[i]$ para a direita
 - 2.2 Defina $i = i - 1$
3. Defina $v[i+1] = x$

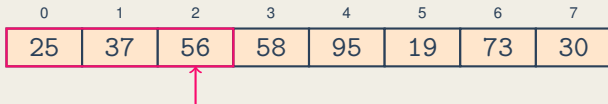


Ordenação por inserção

Para $j=1, \dots, n-1$ faça:

1. Seja $x = v[j]$ e $i = j - 1$
2. Enquanto $v[i] > x$ faça:
 - 2.1 Desloque $v[i]$ para a direita
 - 2.2 Defina $i = i - 1$
3. Defina $v[i+1] = x$

0	1	2	3	4	5	6	7
25	37	56	58	95	19	73	30



Ordenação por inserção

Para $j=1, \dots, n-1$ faça:

1. Seja $x = v[j]$ e $i = j - 1$
2. Enquanto $v[i] > x$ faça:
 - 2.1 Desloque $v[i]$ para a direita
 - 2.2 Defina $i = i - 1$
3. Defina $v[i+1] = x$

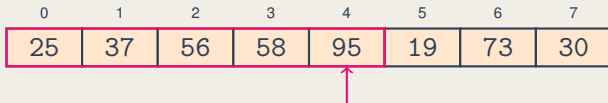
0	1	2	3	4	5	6	7
25	37	56	58	95	19	73	30

Ordenação por inserção

Para $j=1, \dots, n-1$ faça:

1. Seja $x = v[j]$ e $i = j - 1$
2. Enquanto $v[i] > x$ faça:
 - 2.1 Desloque $v[i]$ para a direita
 - 2.2 Defina $i = i - 1$
3. Defina $v[i+1] = x$

0	1	2	3	4	5	6	7
25	37	56	58	95	19	73	30

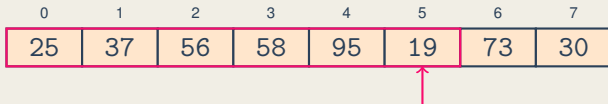


Ordenação por inserção

Para $j=1, \dots, n-1$ faça:

1. Seja $x = v[j]$ e $i = j - 1$
2. Enquanto $v[i] > x$ faça:
 - 2.1 Desloque $v[i]$ para a direita
 - 2.2 Defina $i = i - 1$
3. Defina $v[i+1] = x$

0	1	2	3	4	5	6	7
25	37	56	58	95	19	73	30

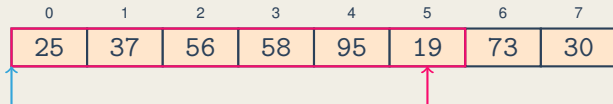


Ordenação por inserção

Para $j=1, \dots, n-1$ faça:

1. Seja $x = v[j]$ e $i = j - 1$
2. Enquanto $v[i] > x$ faça:
 - 2.1 Desloque $v[i]$ para a direita
 - 2.2 Defina $i = i - 1$
3. Defina $v[i+1] = x$

0	1	2	3	4	5	6	7
25	37	56	58	95	19	73	30

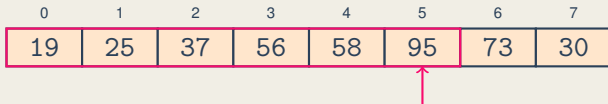


Ordenação por inserção

Para $j=1, \dots, n-1$ faça:

1. Seja $x = v[j]$ e $i = j - 1$
2. Enquanto $v[i] > x$ faça:
 - 2.1 Desloque $v[i]$ para a direita
 - 2.2 Defina $i = i - 1$
3. Defina $v[i+1] = x$

0	1	2	3	4	5	6	7
19	25	37	56	58	95	73	30

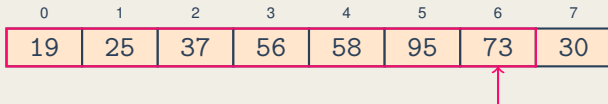


Ordenação por inserção

Para $j=1, \dots, n-1$ faça:

1. Seja $x = v[j]$ e $i = j - 1$
2. Enquanto $v[i] > x$ faça:
 - 2.1 Desloque $v[i]$ para a direita
 - 2.2 Defina $i = i - 1$
3. Defina $v[i+1] = x$

0	1	2	3	4	5	6	7
19	25	37	56	58	95	73	30



Ordenação por inserção

Para $j=1, \dots, n-1$ faça:

1. Seja $x = v[j]$ e $i = j - 1$
2. Enquanto $v[i] > x$ faça:
 - 2.1 Desloque $v[i]$ para a direita
 - 2.2 Defina $i = i - 1$
3. Defina $v[i+1] = x$

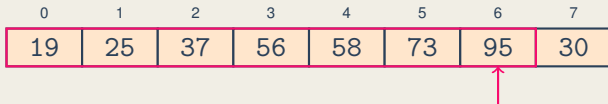
0	1	2	3	4	5	6	7
19	25	37	56	58	95	73	30

Ordenação por inserção

Para $j=1, \dots, n-1$ faça:

1. Seja $x = v[j]$ e $i = j - 1$
2. Enquanto $v[i] > x$ faça:
 - 2.1 Desloque $v[i]$ para a direita
 - 2.2 Defina $i = i - 1$
3. Defina $v[i+1] = x$

0	1	2	3	4	5	6	7
19	25	37	56	58	73	95	30

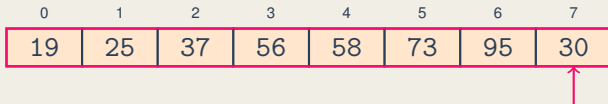


Ordenação por inserção

Para $j=1, \dots, n-1$ faça:

1. Seja $x = v[j]$ e $i = j - 1$
2. Enquanto $v[i] > x$ faça:
 - 2.1 Desloque $v[i]$ para a direita
 - 2.2 Defina $i = i - 1$
3. Defina $v[i+1] = x$

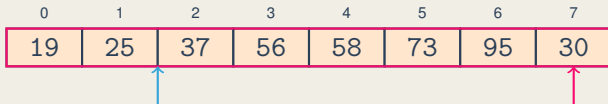
0	1	2	3	4	5	6	7
19	25	37	56	58	73	95	30



Ordenação por inserção

Para $j=1, \dots, n-1$ faça:

1. Seja $x = v[j]$ e $i = j - 1$
2. Enquanto $v[i] > x$ faça:
 - 2.1 Desloque $v[i]$ para a direita
 - 2.2 Defina $i = i - 1$
3. Defina $v[i+1] = x$




Ordenação por inserção

Para $j=1, \dots, n-1$ faça:

1. Seja $x = v[j]$ e $i = j - 1$
2. Enquanto $v[i] > x$ faça:
 - 2.1 Desloque $v[i]$ para a direita
 - 2.2 Defina $i = i - 1$
3. Defina $v[i+1] = x$

0	1	2	3	4	5	6	7
19	25	30	37	56	58	73	95



Ordenação por inserção

Para $j=1, \dots, n-1$ faça:

1. Seja $x = v[j]$ e $i = j - 1$
2. Enquanto $v[i] > x$ faça:
 - 2.1 Desloque $v[i]$ para a direita
 - 2.2 Defina $i = i - 1$
3. Defina $v[i+1] = x$

0	1	2	3	4	5	6	7
19	25	30	37	56	58	73	95

Ordenação por inserção

Para $j=1, \dots, n-1$ faça:

1. Seja $x = v[j]$ e $i = j - 1$
2. Enquanto $v[i] > x$ faça:
 - 2.1 Desloque $v[i]$ para a direita
 - 2.2 Defina $i = i - 1$
3. Defina $v[i+1] = x$

```
1 void Inserção (int v[], n) {  
2  
3     /* ...? */  
4  
5 }
```

Ordenação por inserção

Para $j=1, \dots, n-1$ faça:

1. Seja $x = v[j]$ e $i = j$
2. Enquanto $v[i] > x$ faça:
 - 2.1 Desloque $v[i]$ para a direita
 - 2.2 Defina $i = i - 1$
3. Defina $v[i] = x$

```
1 void Inserção (int v[], n) {  
2     int i, j, x;  
3     for (j = 1; j < n; j++) {  
4         x = v[j];  
5         for (i = j-1; i >= 0 && v[i] > x; i--)  
6             v[i+1] = v[i];  
7         v[i+1] = x;  
8     }  
9 }
```


Ordenação por inserção

```
1 void Inserção (int n, int v[]) {  
2     int i, j, x;  
3     for (j = 1; j < n; j++) { /* Invariante */  
4         x = v[j];  
5         for (i = j-1; i >= 0 && v[i] > x; i--)  
6             v[i+1] = v[i];  
7         v[i+1] = x;  
8     }  
9 }
```

Invariantes

Ordenação por inserção

```
1 void Inserção (int n, int v[]) {  
2     int i, j, x;  
3     for (j = 1; j < n; j++) { /* Invariante */  
4         x = v[j];  
5         for (i = j-1; i >= 0 && v[i] > x; i--)  
6             v[i+1] = v[i];  
7         v[i+1] = x;  
8     }  
9 }
```

Invariantes

- ▶ $v[0..n-1]$ é uma permutação do vetor original
- ▶ $v[0..j-1]$ é crescente

Ordenação por inserção

```
1 void Insercao (int n, int v[]) {  
2     int i, j, x;  
3     for (j = 1; /* Invariante */ j < n; j++) {  
4         x = v[j];  
5         for (i = j-1; i >= 0 && v[i] > x; i--)  
6             v[i+1] = v[i];  
7         v[i+1] = x;  
8     }  
9 }
```

Complexidade de tempo:

Ordenação por inserção

```
1 void Insercao (int n, int v[]) {  
2     int i, j, x;  
3     for (j = 1; /* Invariante */ j < n; j++) {  
4         x = v[j];  
5         for (i = j-1; i >= 0 && v[i] > x; i--)  
6             v[i+1] = v[i];  
7         v[i+1] = x;  
8     }  
9 }
```

Complexidade de tempo:

- ▶ Modelo de execução: qtd. de comparações `v[i] > x`
- ▶ Pior caso:

Ordenação por inserção

```
1 void Insercao (int n, int v[]) {  
2     int i, j, x;  
3     for (j = 1; /* Invariante */ j < n; j++) {  
4         x = v[j];  
5         for (i = j-1; i >= 0 && v[i] > x; i--)  
6             v[i+1] = v[i];  
7         v[i+1] = x;  
8     }  
9 }
```

Complexidade de tempo:

- ▶ Modelo de execução: qtd. de comparações $v[i] > x$
- ▶ Pior caso: $1 + 2 + \dots + n - 1 = n(n - 1)/2$ comparações
- ▶ Melhor caso:

Ordenação por inserção

```
1 void Insercao (int n, int v[]) {  
2     int i, j, x;  
3     for (j = 1; /* Invariante */ j < n; j++) {  
4         x = v[j];  
5         for (i = j-1; i >= 0 && v[i] > x; i--)  
6             v[i+1] = v[i];  
7         v[i+1] = x;  
8     }  
9 }
```

Complexidade de tempo:

- ▶ Modelo de execução: qtd. de comparações $v[i] > x$
- ▶ Pior caso: $1 + 2 + \dots + n - 1 = n(n - 1)/2$ comparações
- ▶ Melhor caso: n comparações

Animações com vetores ordenados de forma distintas:

`https://www.toptal.com/developers/sorting-algorithms/
insertion-sort`

Ordenação por inserção em vetor é estável?

```
1 void Inserção (int n, int v[]) {  
2     int i, j, x;  
3     for (j = 1; j < n; j++) {  
4         x = v[j];  
5         for (i = j-1; i >= 0 && v[i] > x; i--)  
6             v[i+1] = v[i];  
7         v[i+1] = x;  
8     }  
9 }
```


Ordenação por inserção em vetor é estável?

```
1 void Inserção (int n, int v[]) {  
2     int i, j, x;  
3     for (j = 1; j < n; j++) {  
4         x = v[j];  
5         for (i = j-1; i >= 0 && v[i] > x; i--)  
6             v[i+1] = v[i];  
7         v[i+1] = x;  
8     }  
9 }
```

Sim, posição relativa entre elementos nunca é alterada

Algoritmo

Para $j=1, \dots, n-1$ faça:

1. Seja $x = v[j]$ e $i = j$
2. Enquanto $v[i] > x$ faça:
 - 2.1 **Desloque** $v[i]$ **para a direita**
 - 2.2 **Defina** $i = i - 1$
3. Defina $v[i] = x$

```
1 /* Rearranja lista com cabeça para que seja crescente */
2 void Inserção (celula *lista) {
3
4     /* ... */
5
6 }
```

Ordenação por inserção em lista encadeada

Algoritmo

Para cada elemento p na lista:

1. Seja $q = p \rightarrow \text{prox}$, $r = \text{cabeça}$ e $s = r \rightarrow \text{prox}$
2. Enquanto $s \rightarrow \text{valor} < q \rightarrow \text{valor}$ faça
 $r = s$, $s = s \rightarrow \text{prox}$
3. Insira q entre r e s

```
1 /* Rearranja lista com cabeça para que seja crescente */
2 void Inserção (celula *lista) {
3
4     /* ... */
5
6 }
```

Ordenação por inserção em lista encadeada

```
1  /* Rearranja lista com cabeça para que seja crescente */
2  void Inserção (celula *lista) {
3      celula *p, *q, *r, *s;
4      for (p=lista, q=lista->prox; q!=NULL; p=q, q=q->prox) {
5          for (r=lista, s=lista->prox;
6              s->valor < q->valor; r=s, s=s->prox);
7          if (q != s) { /* sem isso cria laço q->prox=q */
8              p->prox = q->prox; /* remove q */
9              r->prox = q; q->prox=s; /* insere q entre r e s */
10             q = p; /* elemento seguinte */
11         }
12     }
13 }
```

Ordenação por inserção em lista encadeada

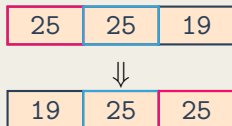
```
1  /* Rearranja lista com cabeca para que seja crescente */
2  void Inserção (celula *lista) {
3      celula *p, *q, *r, *s;
4      for (p=lista , q=lista ->prox; q!=NULL; p=q, q=q->prox) {
5          for (r=lista , s=lista ->prox;
6              s->valor < q->valor; r=s, s=s->prox);
7          if (q != s) { /* se nao cria laco q->prox=q */
8              p->prox = q->prox;      /* remove q */
9              r->prox = q; q->prox=s; /* insere q entre r e s */
10             q = p;                  /* elemento seguinte */
11         }
12     }
13 }
```

Estável?

Ordenação por inserção em lista encadeada

```
1  /* Rearranja lista com cabeca para que seja crescente */
2  void Inserção (celula *lista) {
3      celula *p, *q, *r, *s;
4      for (p=lista, q=lista->prox; q!=NULL; p=q, q=q->prox) {
5          for (r=lista, s=lista->prox;
6              s->valor < q->valor; r=s, s=s->prox);
7          if (q != s) { /* se nao cria laco q->prox=q */
8              p->prox = q->prox;      /* remove q */
9              r->prox = q; q->prox=s; /* insere q entre r e s */
10             q = p;                  /* elemento seguinte */
11         }
12     }
13 }
```

Estável? Não



Algoritmo

Para cada elemento p na lista:

1. Seja $q = p \rightarrow \text{prox}$, $r = \text{cabeça}$ e $s = r \rightarrow \text{prox}$
2. **Enquanto** $s \rightarrow \text{valor} \leq q \rightarrow \text{valor}$ **faça**
 $r = s$, $s = s \rightarrow \text{prox}$
3. Insira q entre r e s

Ordenação por inserção em lista encadeada estável

```
1  /* Rearranja lista com cabeca para que seja crescente
2     Mantem ordem relativa */
3  void Insercao (celula *lista) {
4     celula *p, *q, *r, *s;
5     for (p=lista, q=lista->prox; q!=NULL; p=q, q=q->prox) {
6         for (r=lista, s=lista->prox;
7             s->valor<=q->valor && s != q; r=s, s=s->prox)
8             ;
9         if (q != s) {          /* para nao criar laco q->prox=q */
10            p->prox = q->prox;    /* remove q */
11            r->prox = q; q->prox=s; /* insere q entre r e s */
12            q = p;                /* elemento seguinte */
13        }
14    }
```


Aplicação

```
1  /* Determina se string s é anagrama de string t */
2  int Anagrama (char *s, char *t) {
3      int i, n = strlen(s), m = strlen(t);
4      if (n != m) return 0;
5      Ordena (s, n);
6      Ordena (t, m);
7      for (i = 0; i < n; i++) if (s[i] != t[i]) return 0;
8      return 1;
9  }
```

Exercício 10