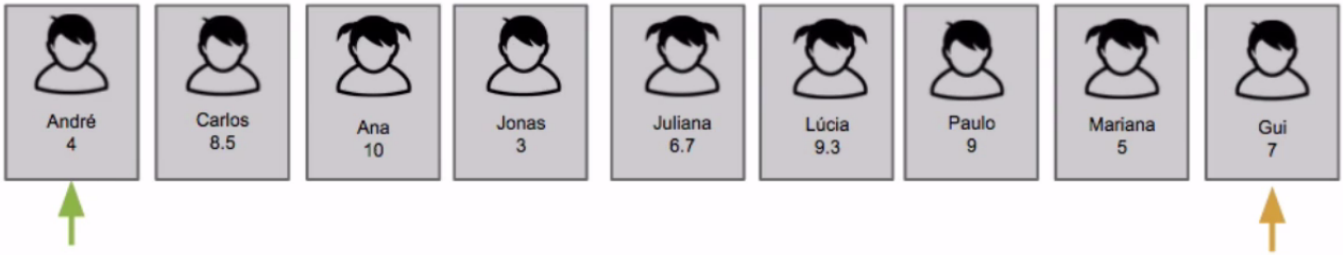
02

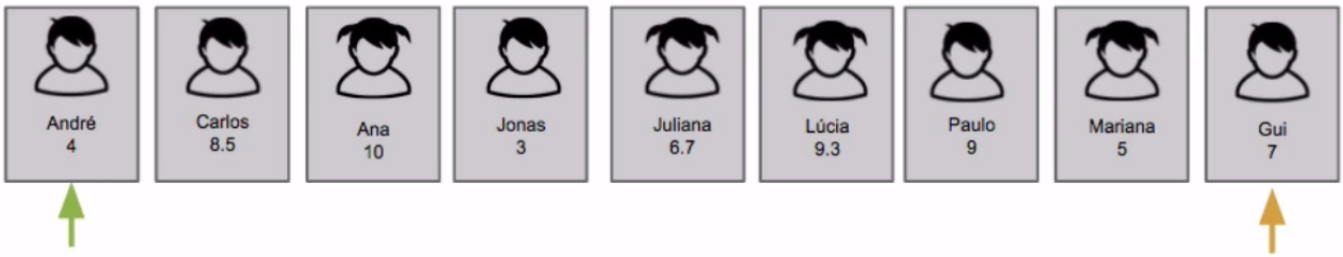
Colocando um elemento na posição adequada

Transcrição

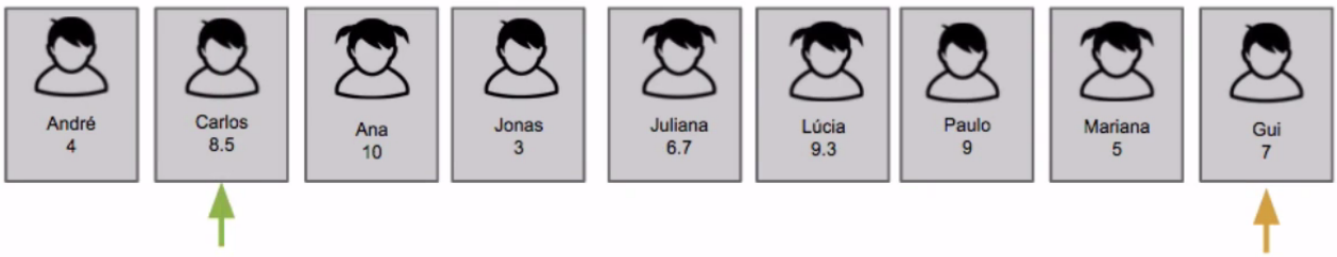
Agora, queremos rodar algum processo que seja capaz de indicar quais elementos são menores que o Guilherme. Então, analisaremos todos os itens. Para isto, iremos usar a variável `analisando`. Contamos as pessoas com menores notas no `menoresAtéAgora`. Vamos fazer isto?

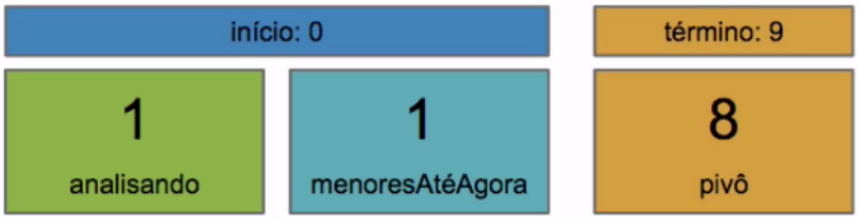


Primeiro passo, o André é menor do que o Guilherme? Sim. Então `menoresAtéAgora` será igual a 1.

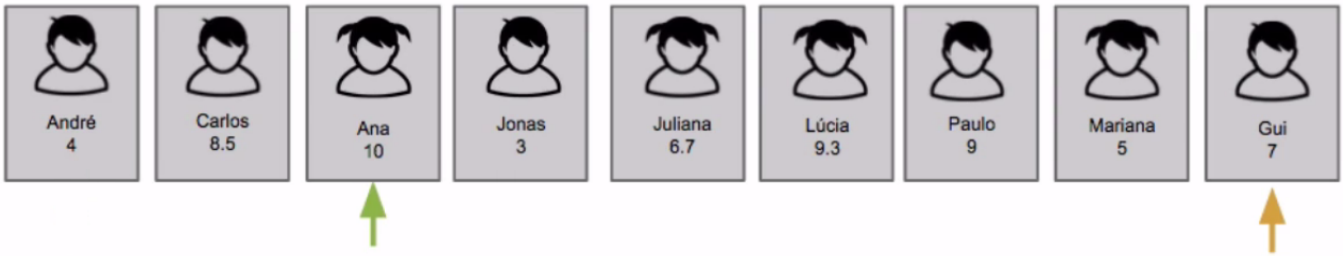


O próximo é o Carlos. A variável `analisando` será igual a 1.

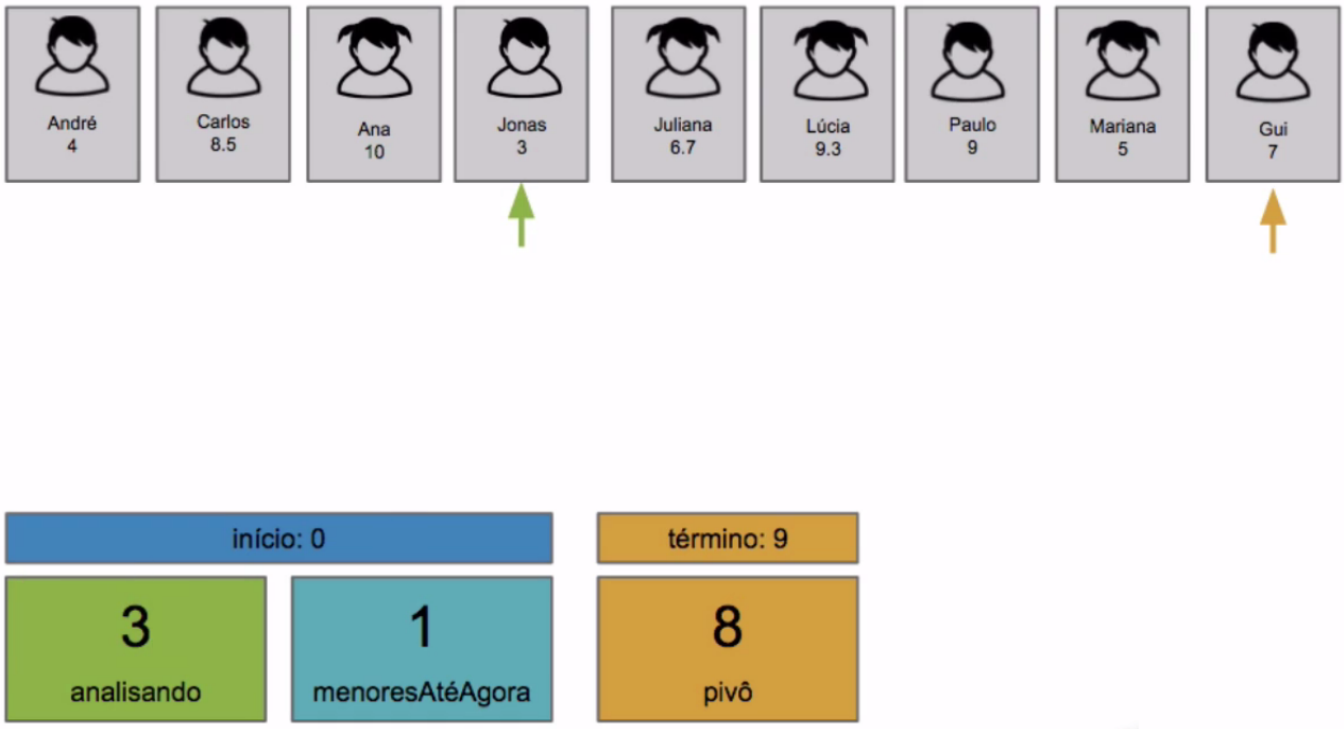




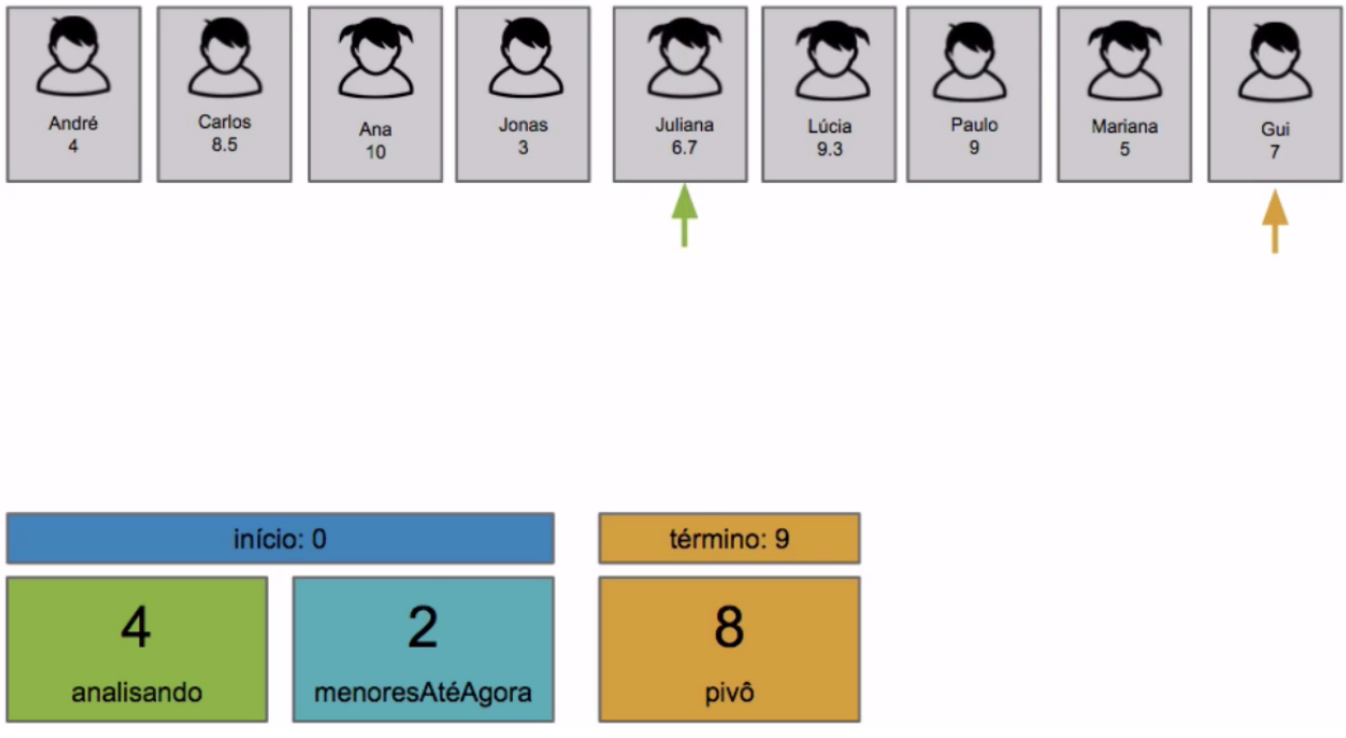
Ele é menor do que o Guilherme? Não. Somaremos +1 na variável `analisando` .



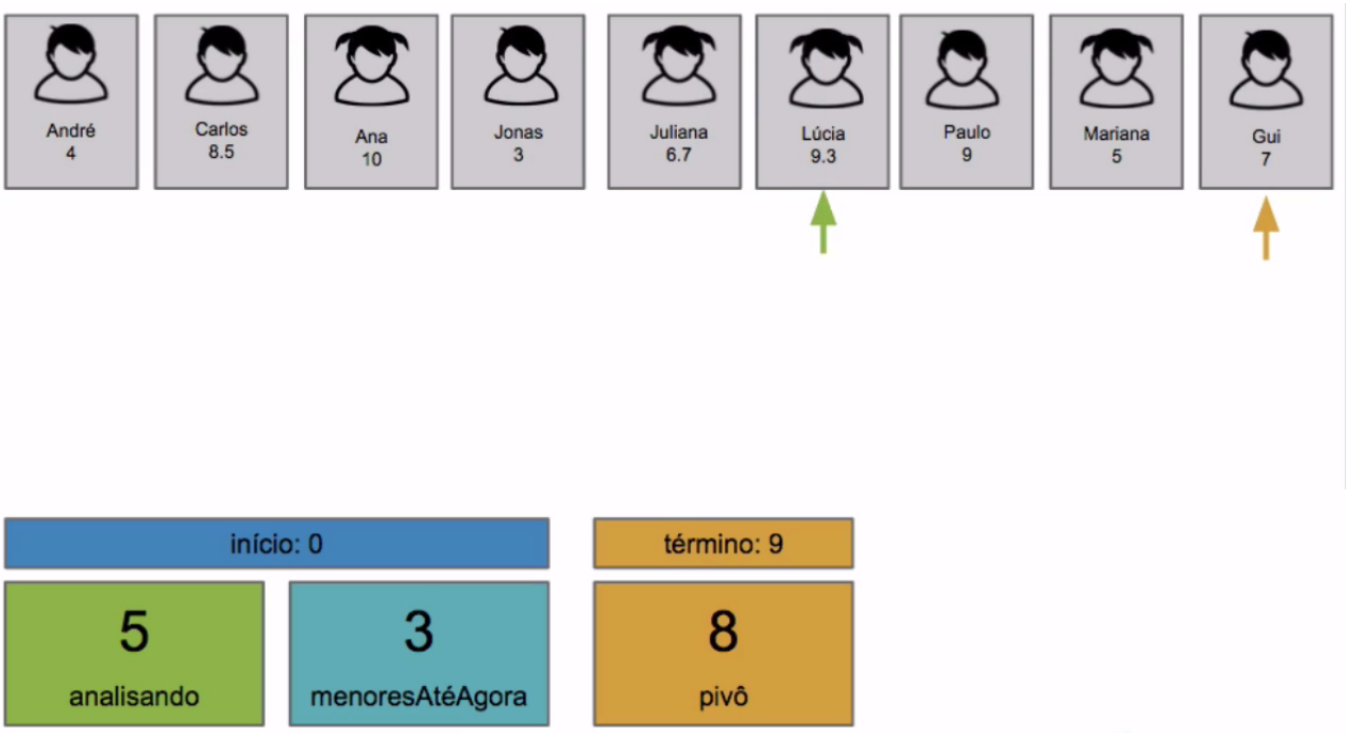
A Ana é melhor do que o Guilherme? Não. O `analisando` será igual a 3.



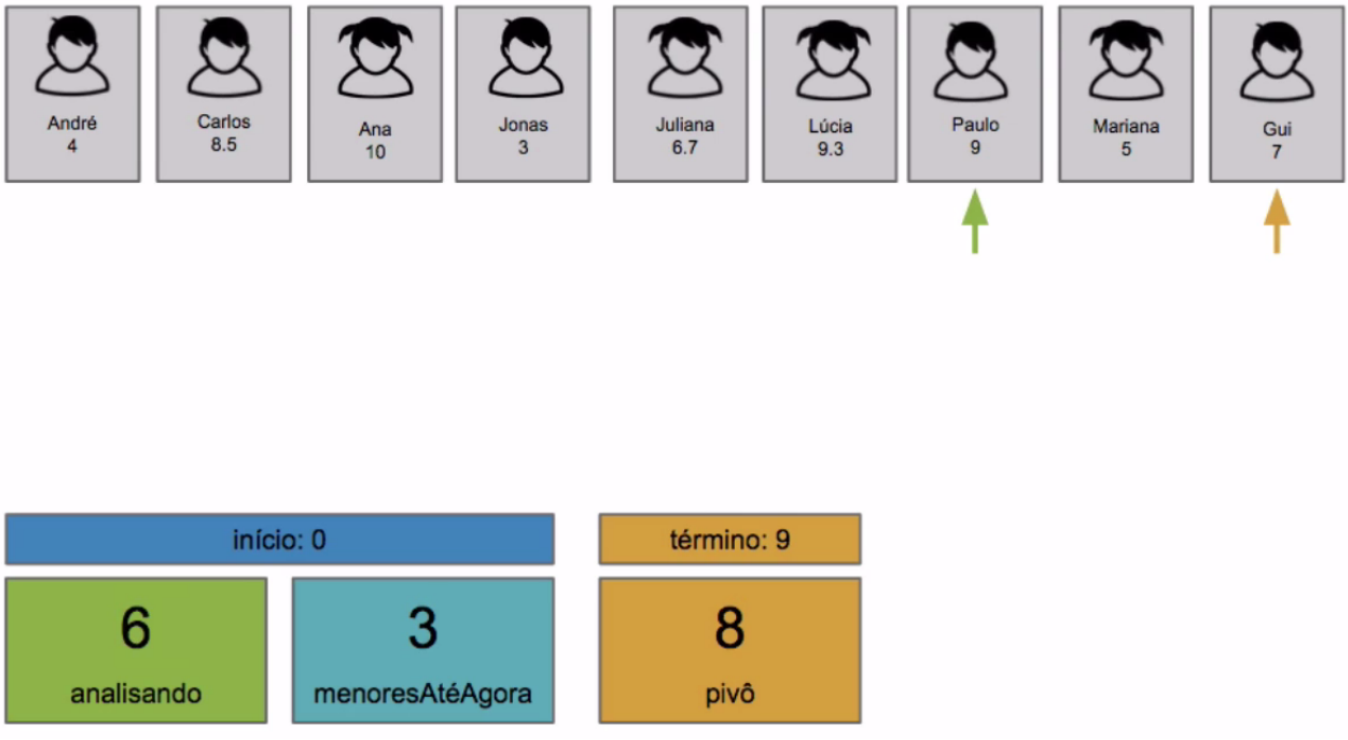
Jonas é menor do que o Guilherme? Sim. Quantos são menores até agora? **Duas** pessoas. A variável `analisando` será igual a 4 e seguiremos para a Juliana.



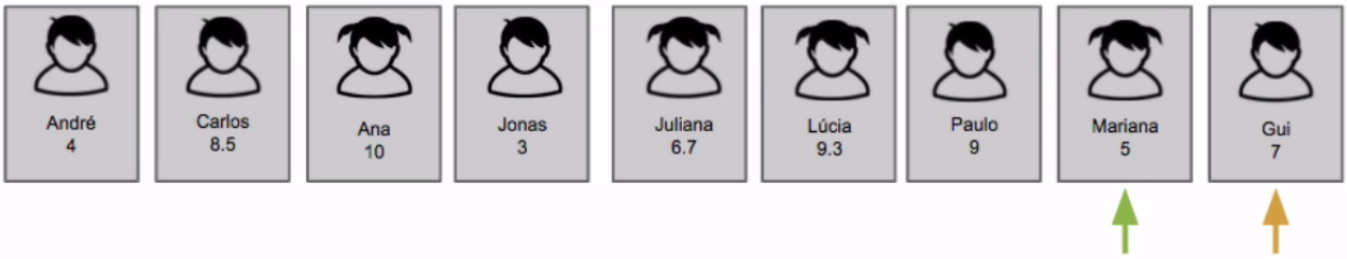
Ela é menor do que o Guilherme? Sim. Vamos analisar o próximo item e somaremos +1 nas variáveis `analisando` , que será igual a 5, e `menoresAtéAgora` , que será igual a 3.



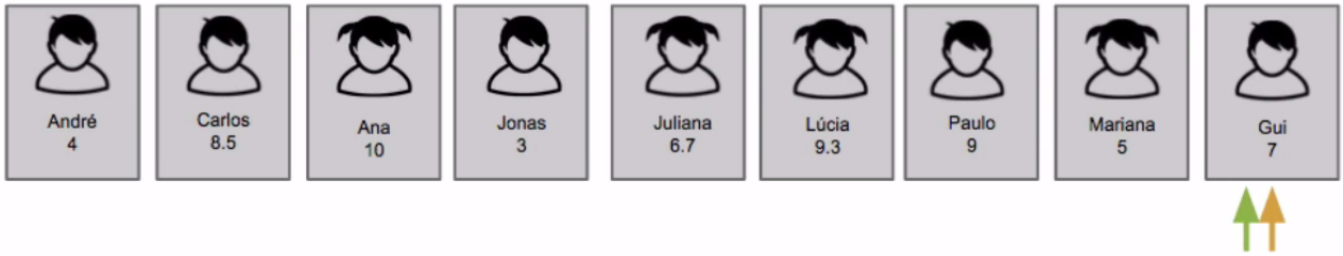
A Lúcia é menor do que o Guilherme? Não. Então, vamos para o próximo e `analisando` será igual a 6.



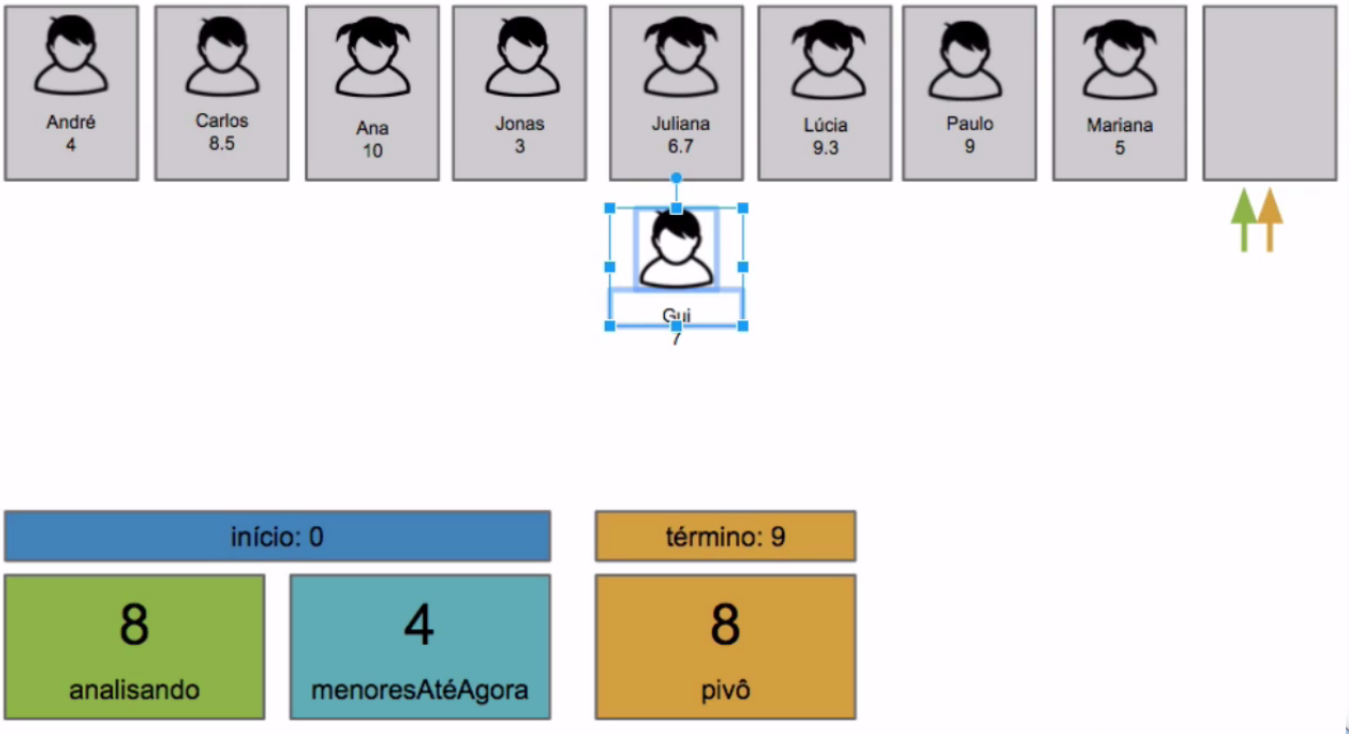
O Paulo é menor do que o Guilherme? Não. Iremos para o próximo elemento e `analisando` será igual a 7.



A Mariana é menor do que o Guilherme? Sim. Então somaremos +1 nas variáveis `analisando` e `menoresAtéAgora`. Quatro pessoas tiraram notas menores.

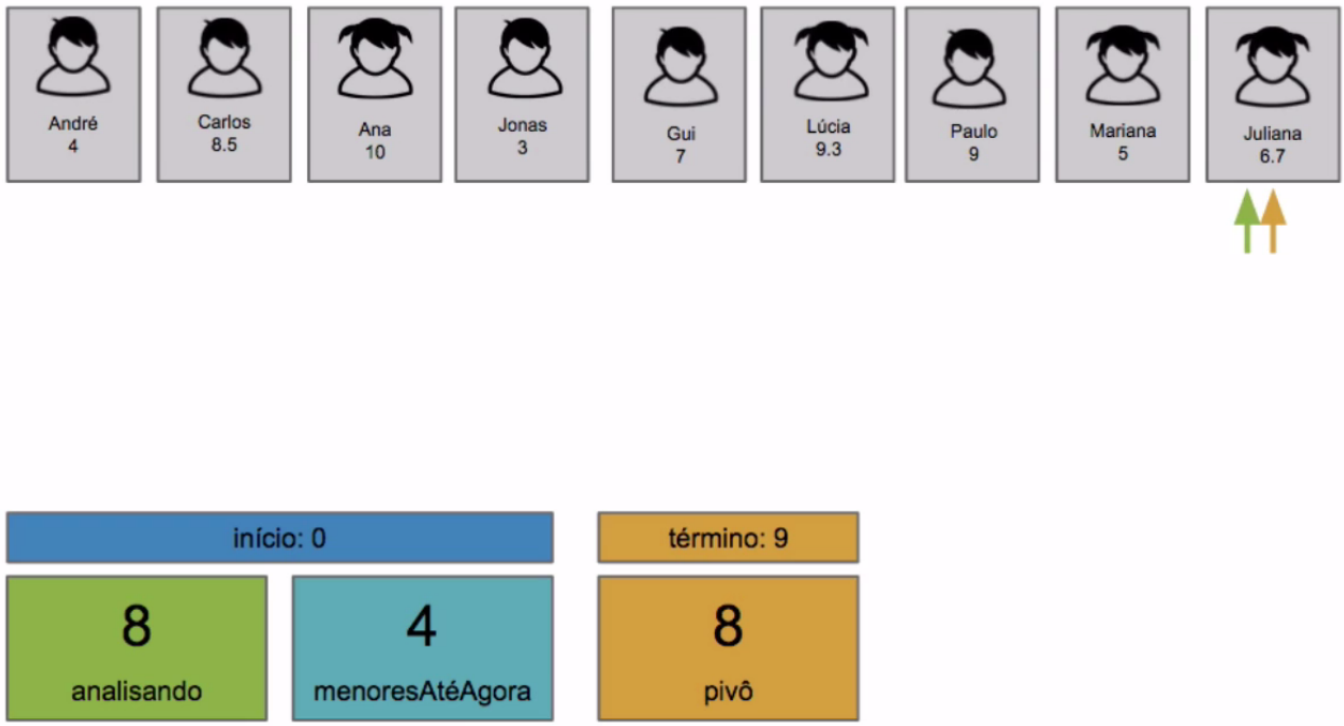


Ao passarmos para o próximos, chegamos ao fim. Nós analisamos todos os itens e descobrimos quantos são menores. Era o que desejávamos saber, porque assim poderemos indicar a posição em que o Guilherme deve ficar. Agora poderemos colocar o Guilherme na **posição 4**.



Se tinham quatro pessoas com notas menores do que a dele, esta será a posição adequada. Se existissem duas pessoas com notas menores, ele deveria ocupar a posição 2. Se ninguém fosse menor que o Guilherme, ele deveria ocupar a posição 0, que é a primeira do *array*.

Logo, quando terminamos de varrer todos os elementos, o pivô deve ficará na posição da variável `menoresAtéAgora` . Nós já encontramos a posição adequada. Porém, não basta apenas trocá-lo de lugar com a Juliana.



Por quê? Porque nem todos os demais elementos posicionados à esquerda são menores do que o Guilherme. Existem maiores e menores. Enquanto analisávamos, precisávamos organizar a ordem dos alunos, para garantir que os menores ficassem à esquerda e expulsado os que fossem maiores desta parte. Como podemos fazer isto? Veremos em seguida como é possível encontrar a posição adequada do Guilherme e expulsar todos que tiraram notas maiores para o lado direito do *array*. Temos que fazer isto de alguma maneira...

Implementando minha posição

Queremos escrever um algoritmo que não apenas encontre o número de elementos menores, mas que também posicione o pivô (o último elemento) adequadamente. Assim poderemos separar as notas menores das maiores.

Criaremos uma nova classe, que se chamará `TestaPivota`, porque nós iremos pivotar. Dentro da classe, criaremos o método `main()`, onde colocaremos as notas dos alunos.

```
public static void main(String[] args) {
    Nota guilherme = new Nota("guilherme", 7);
    Nota[] notas = {
            new Nota("andre", 4),
            new Nota("carlos", 8.5),
            new Nota("ana", 10),
            new Nota("jonas", 3),
            new Nota("juliana", 6.7),
            guilherme,
            new Nota("paulo", 9),
            new Nota("mariana", 5),
            new Nota("lucia", 9.3)
    };
}
```

Nossa lista terá uma única diferença: o Guilherme não ficará mais no meio, porque nós definimos que o pivô sempre será o último elemento. Então ele trocou de lugar com a Lúcia. Vamos trocar os elementos de posição:

```
public static void main(String[] args) {
    Nota guilherme = new Nota("guilherme", 7);
    Nota[] notas = {
            new Nota("andre", 4),
            new Nota("carlos", 8.5),
            new Nota("ana", 10),
            new Nota("jonas", 3),
            new Nota("juliana", 6.7),
            new Nota("paulo", 9),
            new Nota("mariana", 5),
            new Nota("lucia", 9.3)
            guilherme,
    };
}
```

A nova ordem será: André, Carlos, Ana, Jonas, Juliana, Paulo, Mariana, Lúcia e Guilherme. Agora queremos que o nosso algoritmo quebre o *array* em duas partes (`quebraNoPivo`) e depois faça outras ações. Após dividir os elementos, as notas estarão disponíveis. Vamos especificar isto no código:

```
Nota[] notas = {
            new Nota("andre", 4),
            new Nota("carlos", 8.5),
            new Nota("ana", 10),
            new Nota("jonas", 3),
            new Nota("juliana", 6.7),
            new Nota("paulo", 9),
            new Nota("mariana", 5),
```

```
            new Nota("lucia", 9.3)
            guilherme,
    };

    quebraNoPivo(notas);
}
```

Em seguida, iremos implementar o método `quebraNoPivo`. Como ele funciona? Nós já tínhamos encontrado uma forma de descobrir o menor elemento da lista. Primeiro, especificávamos quais elementos seriam analisados, para isto precisávamos saber o valor do `inicial` e do `termino`.

```
private static void quebraNoPivo(Nota[] notas, int inicial, int termino) {

}
```

Isto significa que iremos do 0 até `notas.length`.

```
quebraNoPivo(notas, 0, notas. length);
```

Este era o problema que queríamos resolver. E quem é o pivô? O elemento que está na última posição que iremos analisar. Como temos nove elementos, isto significa que ele estará na posição 8. Logo, ele será o item da posição `termino - 1`. Caso tivéssemos cinco elementos, o pivô estaria na posição 4. Se fossem dezessete elementos, ele estaria na posição 16. O último elemento sempre será o pivô.

```
private static void quebraNoPivo(Nota[] notas, int inicial, int termino) {
    Nota pivo = notas[termino - 1];

}
```

Agora queremos varrer todo o *array*. Por isso, escreveremos o `for`:

```
for(int analisando = 0; analisando < termino; analisando++) {
}
```

Nosso código ficará assim:

```
private static void quebraNoPivo(Nota[] notas, int inicial, int termino) {
    Nota pivo = notas[termino - 1];
    for(int analisando = 0; analisando < termino; analisando++) {
    }
}
```

Iremos varrer o *array* inteiro, porém não será preciso passar por todos os itens, afinal o Guilherme será o último elemento. Por isso, podemos ignorá-lo e não o analisaremos. Vamos alterar o nosso `for` e especificar que devemos analisar até o `termino -1`.

```
 for(int analisando = 0; analisando < termino - 1; analisando++) {
    }
```

Não precisamos comparar as demais notas com o pivô. O que faremos em seguida? Analisaremos a nota atual (`notas[analisando]`).

```
for(int analisando = 0; analisando < termino - 1; analisando++) {
    Nota atual = notas[analisando];
}
```

Será que a nota atual é mais baixa? Se (`if`) a `atual.getValor()` for menor ou igual ao `pivo.getValor()` , ficará posicionado à esquerda. Para isto, usaremos a variável `menoresEncontrados++`

```
for(int analisando = 0; analisando < termino - 1; analisando++) {
    Nota atual = notas[analisando];
    if(atual.getValor() <= pivo.getValor()) {
        menoresEncontrados++;
    }
}
```

Agora precisaremos calcular os `menoresEncontrados`

```
private static void quebraNoPivo(Nota[] notas, int inicial, int termino) {
    int menoresEncontrados = 0;
}
```

Vamos revisar o que estamos fazendo: `menoresEncontrado` começa com 0, depois vamos analisar todos os elementos com exceção do pivô que será ignorado, e contaremos quantos são menores do que o Guilherme. No fim, o que faremos? Já sabemos que o pivô ficará na posição `menoresEncontrados` . Isto significa que iremos trocar no *array*, o elemento que estiver na posição `termino -1` com o que estiver na posição `menoresEncontrados` .

```
troca(notas, termino -1, menoresEncontrados);
```

Nosso objetivo é trocar os dois itens de posição no fim. Vamos implementar a função `troca()` ? Nós queremos receber o `de` e o `para` . Então, `nota1` será o `notas[de]` e o `nota2` será o `notas[para]` .

```
private static void troca(Nota[] notas, int de, int para) {
    Nota nota1 = notas[de];
    Nota nota2 = notas[para];
    notas[para] = nota1;
    notas[de] = nota2;
}
```

Trocamos de lugar os elementos que estavam no `de` para a posição dos que estavam no `para` e vice-versa.