03

Ajustando as pontas do algoritmo

Transcrição

Chegou a hora de aproveitarmos o que sobrou do nosso *array*. Isto é: nós verificamos tanto em um *array* como em outro. De repente, acabaram os elementos de um deles, enquanto do outro acabou e alguns itens sobraram. Precisaremos usar este elementos que restaram.

Se sobrou como no caso em que `atualDoAlberto < notasDoAlberto.length` o que teremos que fazer? Colocaremos todos que sobraram da lista do Alberto, no *array* de resultados.

Isto significa que enquanto (`while`) a condição for verdadeira, vamos inserir o atual no *array* de resultado.

```
System.out.println("Estou saindo");
System.out.println(atualDoMauricio < notasDoMauricio.length);
while(atualDOAlberto < notasDoAlberto) {
    resultado[atual] = notasDoAlberto[atualDoAlberto];

}
```

Depois que colocamos o elemento no *array* de resultado. E andamos com a variável para a direita (`atualDoAlberto++`). Vamos colocar isto no código:

```
while(atualDoAlberto < notasDoAlberto) {
    resultado[atual] = notasDoAlberto[atualDoAlberto];
    atual++;
    atualDoAlberto++;
}
```

Enquanto sobrar elementos, vamos seguir copiando os elementos.

Vamos rodar a aplicação e ela irá mostrar o seguinte resultado:

```
Estou saindo
false
- jonas 3.0
- andre 4.0
- mariana 5.0
- juliana 6.7
- guilherme 7.0
- carlos 8.5
- paulo 9.0
- lucia 9.3
- ana 10.0
```

O que faltava era verificarmos se estava sobrando alguém nos *array*s. Entender qual era o *bug*. Analise o código novamente e tente descobrir qual é o *bug* ainda precisamos corrigir.

Intercalando os elementos que sobraram, independente do lado

Nós ainda temos um *bug* no nosso código. Ele funciona bem quando sobra algum elemento no Alberto.

```
Nota[] rank = junta(notasDoAlberto, notasDoMauricio);
for(Nota nota : rank) {
    System.out.println(nota.getAluno() +" " + nota.getValor());

}
```

E o que acontece se chamarmos a nossa função, com outros valores? Por exemplo, o que acontecerá se trocarmos a posição dos valores na linha? Se dissermos que agora os alunos do Alberto são do Aniche, ou vice-versa.

```
Nota[] rank = junta(notasDoMauricio, notasDoAlberto);
```

Vamos testar rodar novamente? Se você mudar a ordem, observe o que acontecerá:

```
Estou saindo
true
- jonas 3.0
- andre 4.0
- mariana 5.0
- juliana 6.7
- guilherme 7.0
- carlos 8.5
- paulo 9.0
```

Voltaremos a ter o mesmo problema: dois alunos ficarão de fora do resultado. O problema em que o resultado estará sempre correto, quando não fizer diferença se sobraram elementos no primeiro ou no segundo *array*. Se sobram alunos, é porque a notas deles são maiores do que as do outro *array*. Precisamos inseri-los no *array* geral. Por isso, o mesmo `while` que criamos para o `notasDoAlberto` , teremos que fazer para o `notasDoMauricio` .

```
System.out.println("Estou saindo");
while(atualDoAlberto < notasDoAlberto.length) {
    resultado[atual] = notasDoMauricio[atualDoMauricio];
    atualDoMauricio++;
    atual++;
}
while(atualDoAlberto < notasDoAlberto.length) {
    resultado[atual] = notasDoAlberto[atualDoAlberto];
    atual++;
    atualDoAlberto++;
}
```

Testaremos novamente o programa e o resultado estará correto.

```
Estou saindo
true
- jonas 3.0
```

```
- andre 4.0
- mariana 5.0
- juliana 6.7
- guilherme 7.0
- carlos 8.5
- paulo 9.0
- lucia 9.3
- ana 10.0
```

Não importa mais se sobraram alunos em algum dos *array*s. Independente se sobrou no primeiro ou no segundo, caso tenha restado alguém, ele será inserido no resultado.

Pequenas refatorações possíveis ao intercalar os elementos

Chegou o momento de melhorarmos um pouco o nosso código. Ficaram alguns detalhes sobrando...

```
System.out.println("Estou saindo");
while(atualDoAlberto < notasDoAlberto.length) {
    resultado[atual] = notasDoMauricio[atualDoMauricio];
    atualDoMauricio++;
    atual++;
}
while(atualDoAlberto < notasDoAlberto.length) {
    resultado[atual] = notasDoAlberto[atualDoAlberto];
    atual++;
    atualDoAlberto++;
}
```

Por exemplo a linha:

```
System.out.println("Estou saindo");
```

Nós já podemos removê-la, porque sabemos que o código funciona bem.

Podemos fazer alterações também no `if` :

```
    Nota nota1 = notasDoMauricio[atualDoMauricio];
    Nota nota2 = notasDoAlberto[atualDoAlberto];
    System.out.println("Estou comprando " + nota1.getAluno() + " com" + nota2.getAluno());

    if(nota1.getValor() < nota2.getValor()) {
        // mauricio
        resultado[0] = nota2;
        atualDoMauricio++;
    } else {
        // alberto
        resultado[0] = nota2;
        atualDoAlberto++;
    }
    atual++;
```

Os comentários `// mauricio` e `// alberto` também podemos removê-los.

```
 if(nota1.getValor() < nota2.getValor()) {
        resultado[0] = nota2;
        atualDoMauricio++;
    } else {
        resultado[0] = nota2;
        atualDoAlberto++;
    }
```

Observe este trecho:

```
while(atualDoAlberto < notasDoAlberto.length) {
    resultado[atual] = notasDoMauricio[atualDoMauricio];
    atualDoMauricio++;
    atual++;
}
```

O resultado na posição `atual` é o `notasDoMauricio` na posição `atualDoMauricio`. Depois somamos +1 no `atualDoMauricio` e no `atual`. Temos a opção de escrever tudo isto em uma única linha.

```
while(atualDoAlberto < notasDoAlberto.length) {
    resultado[atual++] = notasDoMauricio[atualDoMauricio++];
}
```

E removeremos as duas linhas finais.

Faremos as mesmas alterações com o Alberto:

```
while(atualDoAlberto < notasDoAlberto.length) {
    resultado[atual++] = notasDoAlberto[atualDoAlberto++];

}
```

As duas formas de escrever estão corretas. Porém, eu deixarei da maneira como estava antes. Porque acredito que o código ficará mais legível de outra forma.

Não será um *Enter* a mais que deixará o código mais difícil de ser mantido. O compilador otimiza este tipo de tarefa, não precisamos nos preocupar com isto. Deixo o computador se responsabilizar. Mas o nosso código está claro especificando: primeiro será copiada as notas do Mauricio, e depois somaremos +1 n o `atualDoMauricio` e no `atual`. Ficou claro e separado cada passo do processo.

```
while(atualDoAlberto < notasDoAlberto.length) {
    resultado[atual] = notasDoMauricio[atualDoMauricio];
    atualDoMauricio++;
    atual++;
}
```

Escolho deixar desta maneira, mas sabemos que é possível escrever de outra forma o código.

Continuamos procurando o que podemos melhorar. Temos um `System.out` que também iremos remover.

```
while(atualDoMauricio < notasDoMauricio.length &&
      atualDoAlberto < notasDoAlberto.length) {


    Nota nota1 = notasDoMauricio[atualDoMauricio];
    Nota nota2 = notasDoAlberto[atualDoAlberto];
    System.out.println("Estou comprando " + nota1.getAluno() + " com" + nota2.getAluno());
```

Podemos remover o `System.out` , porque sabemos que o código está funcionando bem.

Agora temos outra questão: precisamos especificar que vamos juntar as notas do Maurício ou do Alberto?

```
Nota[] rank = junta(notasDoAlberto. notasDoMauricio);
for(Nota nota : rank) {
    System.out.println(nota.getAluno() +" " + nota.getValor());
}
```

Não importa se as notas são do Maurício, do Alberto, do Paulo ou do Adriano.. O importante é que temos o primeiro e o segundo *array de notas.

```
private static Nota[] junta(Nota[] notasDoMauricio, Nota[] notasDoAlberto) {
    int total = notasDoMauricio.length + notasDoAlberto.length;
    Nota[] resultado = new Nota[total];

   }
```

Então, vamos substituir `notasDoMauricio` e `notasDoAlberto` por `notas1` e `notas2` , respectivamente.

```
private static Nota[] junta[] notas1, Nota[] notas2) {
    int total = notas1.length + notas2.length;
    Nota[] resultado = new Nota[total];

}
```

Observe que usar um número para distinguir, como nós fizemos com `notas1` e `notas2` , não é o melhor padrão. Porque pode ficar difícil identificar a quem eles se referem. Porém, no nosso exemplo temos dois grupos e queremos unir os elementos em um único *array* de notas. Então, a alteração nos nomes fazem sentido.

Faremos o mesmo em outros trechos do código:

```
private static Nota[] junta[] notas1, Nota[] notas2) {
    int total = notas1.length + notas2.length;
    Nota[] resultado = new Nota[total];

    int atual1 = 0;
    int atual2 = 0;
    int atual = 0;
```

```
    while(atual1 < notas1.length &&
    atual2 < notas2.length) {

        Nota nota1 = notas1[atual1];
        Nota nota2 = notas2[atual2];

        if(nota1.getValor() < nota2.getValor()) {
            resultado[atual] = nota1;
            atual1++;
        } else {
            resultado[atual] = nota2;
            atual2++
    }

}
```

Agora que estamos usando o número 1 e 2 para distinguir os elementos, você pode dizer que o código ficou confuso. Existe alguma outra maneira para renomearmos as variáveis? Não queremos escrever `notasDoMauricio`, porque só faria referência ao Maurício, e nós iremos receber as notas de qualquer pessoa.

Logo, temos nossas variáveis e quero juntá-las em um único *array*, que será o resultado. Nosso código já diz isto. Poderíamos juntar o `++` das variáveis no `while` em uma única linha. Mas optei em deixar de outra maneira.

Também sabemos que se invertermos `nostasDoAlberto` e `notasDoMauricio` não irá interferir no resultado.

```
Nota[] rank = junta(notasDoMauricio, notasDoAlberto);
for(Nota nota : rank) {
    System.out.println(nota.getAluno() +" "  + nota.getValor());
```

A nossa função que junta os *array*s, além de unir os elementos, intercala os valores que estão dentro da lista de uma maneira ordenada. Então, iremos alterar o nome da função de `junta()` para `intercala()`.

```
Nota[] rank = intercala(notasDoMauricio, notasDoAlberto);
for(Nota nota : rank) {
    System.out.println(nota.getAluno() +" " + nota.getValor());
    }
```

A função intercala os elementos de uma maneira ordenada, considerando que eles já estavam organizados em cada um dos *array*s.

Após renomear as variáveis e funções, vamos verificar se o código continua funcionando?

Ao rodarmos novamente, temos o seguinte resultado:

```
- jonas 3.0
- andre 4.0
- mariana 5.0
- juliana 6.7
- guilherme 7.0
```

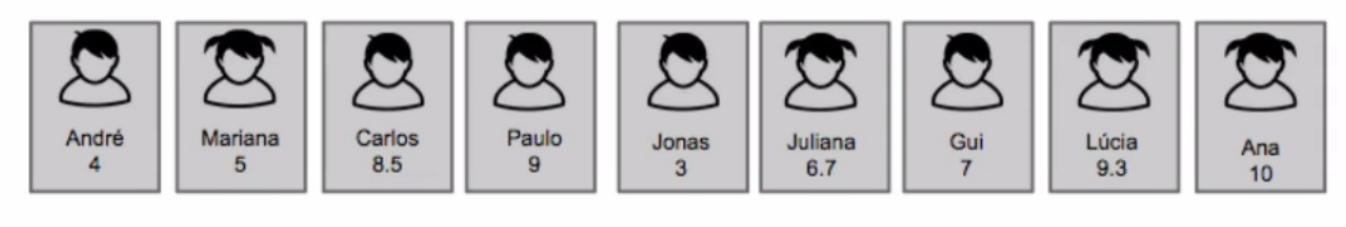
```
- carlos 8.5
- paulo 9.0
- lucia 9.3
- ana 10.0
```

O código roda corretamente, agora com os nomes adequados para a função e para as variáveis que representam o que realmente são.

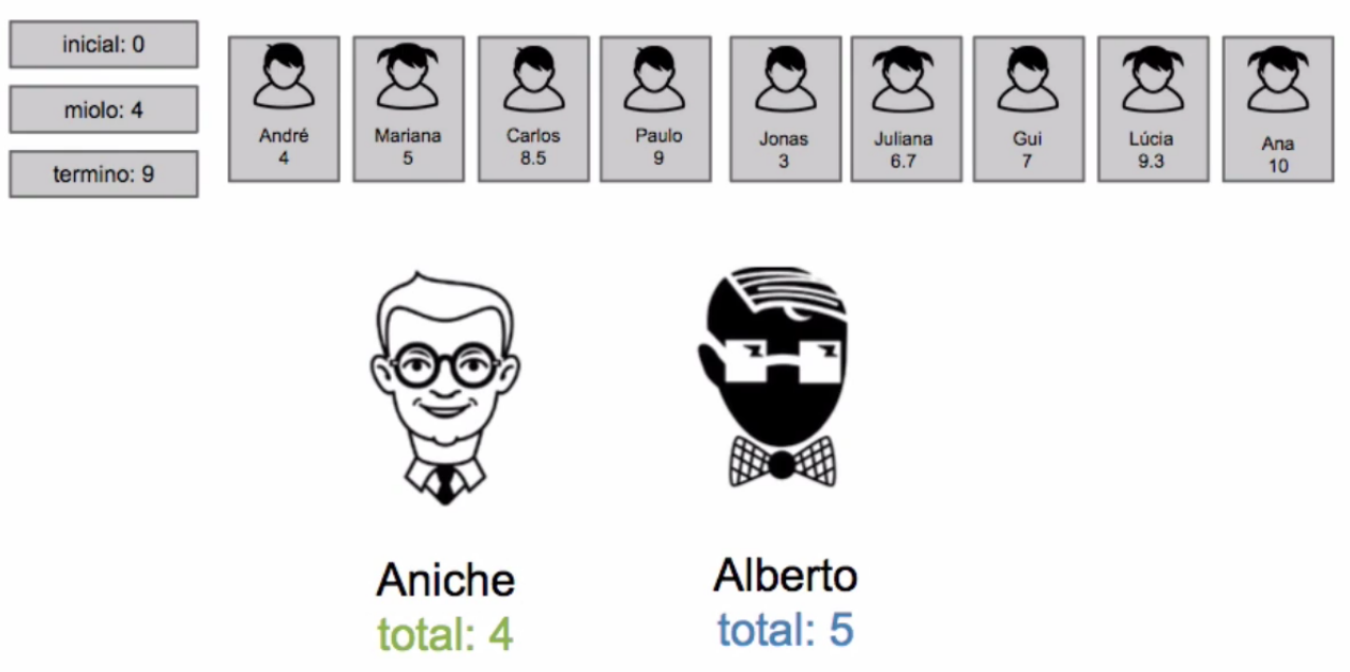
O problema de intercalar dados em um único *array*

Fomos capazes de intercalar dois *array*s já ordenados. Se dividirmos as cartas de baralho - o dinheirinho falso, notas dos alunos ou outros elementos - entre diversas pessoas e cada uma ordenar uma parte do total, intercalar é uma tarefa mais simples. É assim quando temos vários *array*s...

Mas o que acontece muitas vezes? Um professor chega e deixa uma pilha de provas. Depois outro professor coloca em cima uma nova pilha. Ou seja, as provas vão sendo amontoadas uma nas outras. Desta forma, não teremos dois *array* separados e organizados, um do Aniche e outro do Alberto. Na verdade, teremos uma única pilha, um só *array*, formada por dois grupos ordenados separadamente.



Não teremos um *array* de tamanho 4 e outro de tamanho 5, para depois criarmos uma lista de tamanho 9. O que costuma acontecer é que temos um *array* de tamanho 9, em que os quatro primeiros elementos (que seguem até o "miolo") estão ordenados do menor para o maior, da mesma forma estão ordenados os próximos cinco itens. Começamos do 0 e terminamos no 9. O inicial é 0 e o termino é 9, enquanto o miolo é o 4. Então, na prática, o que acontece com frequência é que recebemos um *array (e não dois), com duas partes ordenadas. Nosso objetivo é intercalar estes dois pedaços. Alguém nos diz: "Aqui estão o meu monte de cartas e o seu. Agora encontre uma maneira de intercalar todos os itens." Nós fazemos um monte único e mandamos intercalar.



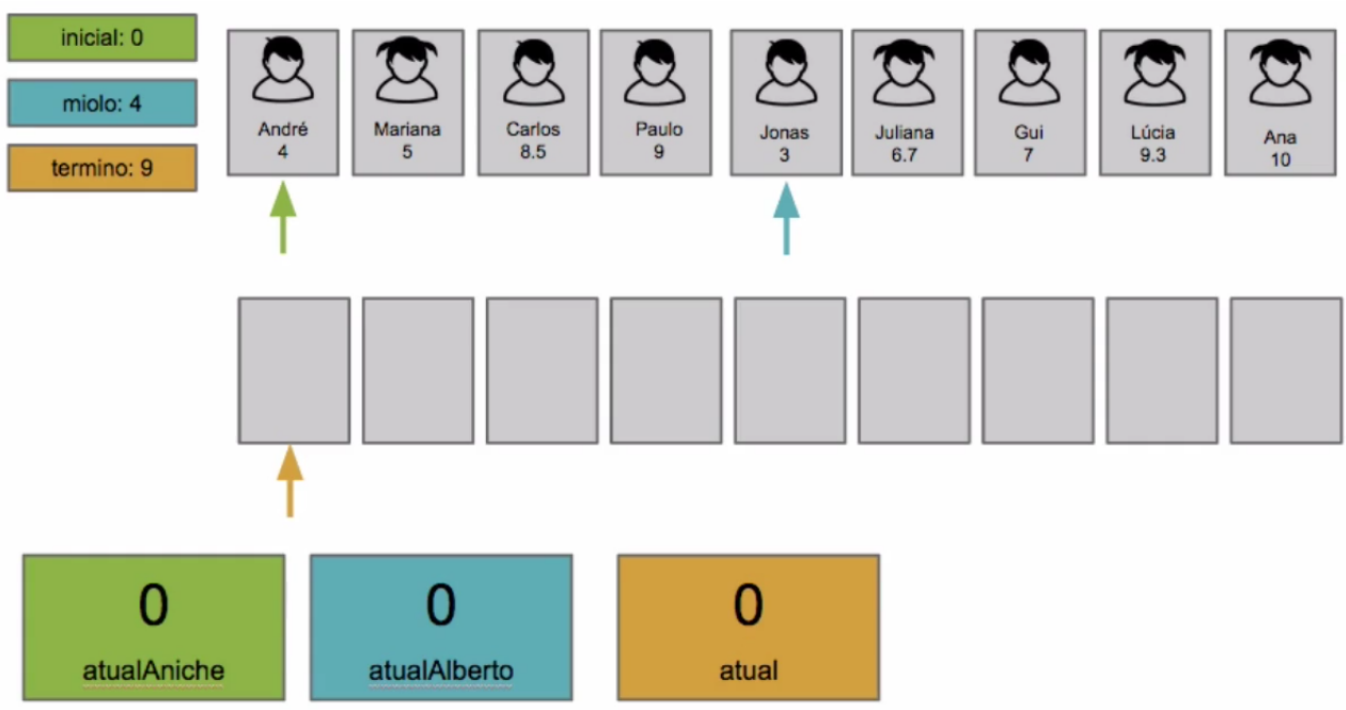
Logo, tudo o que fizemos até agora com dois *array*s, teremos que fazer algumas alterações no nosso código para trabalharmos com uma lista única.

Simulando a intercalação com um único array

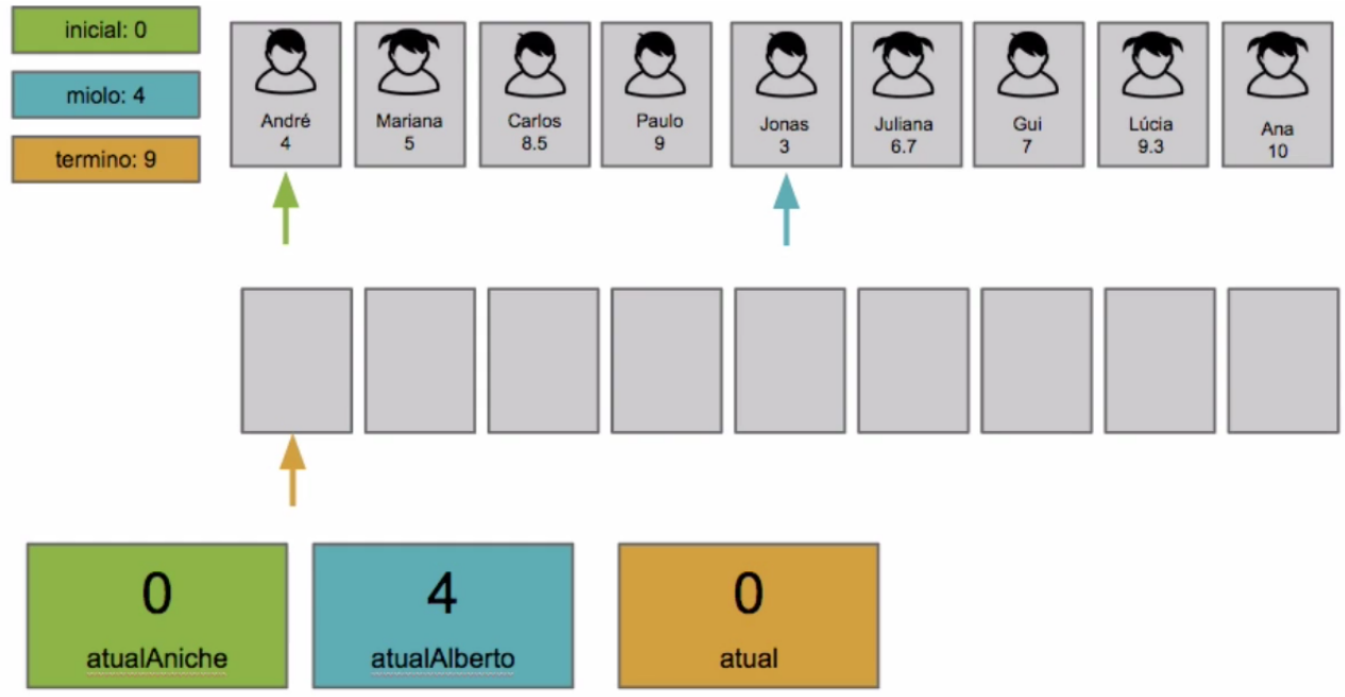
Voltando ao nosso algoritmo de intercalar dois *array*s, nós sabemos que recebemos um único *array*. O total de elementos se refere ao `termino`.

O primeiro elemento é o 0. Então como nós sabemos qual é a primeira parte e qual é a segunda? Com a variável `miolo`.

Na prática, o que temos agora é algo muito parecido com o que fizemos antes. Temos o `atualAniche` que começa com o valor do `inicial` e o `atualAlberto` que começa com o valor do `miolo`. O valor do `termino` é 9, o tamanho do *array*. A variável `atual` também como era antes, começa com 0.

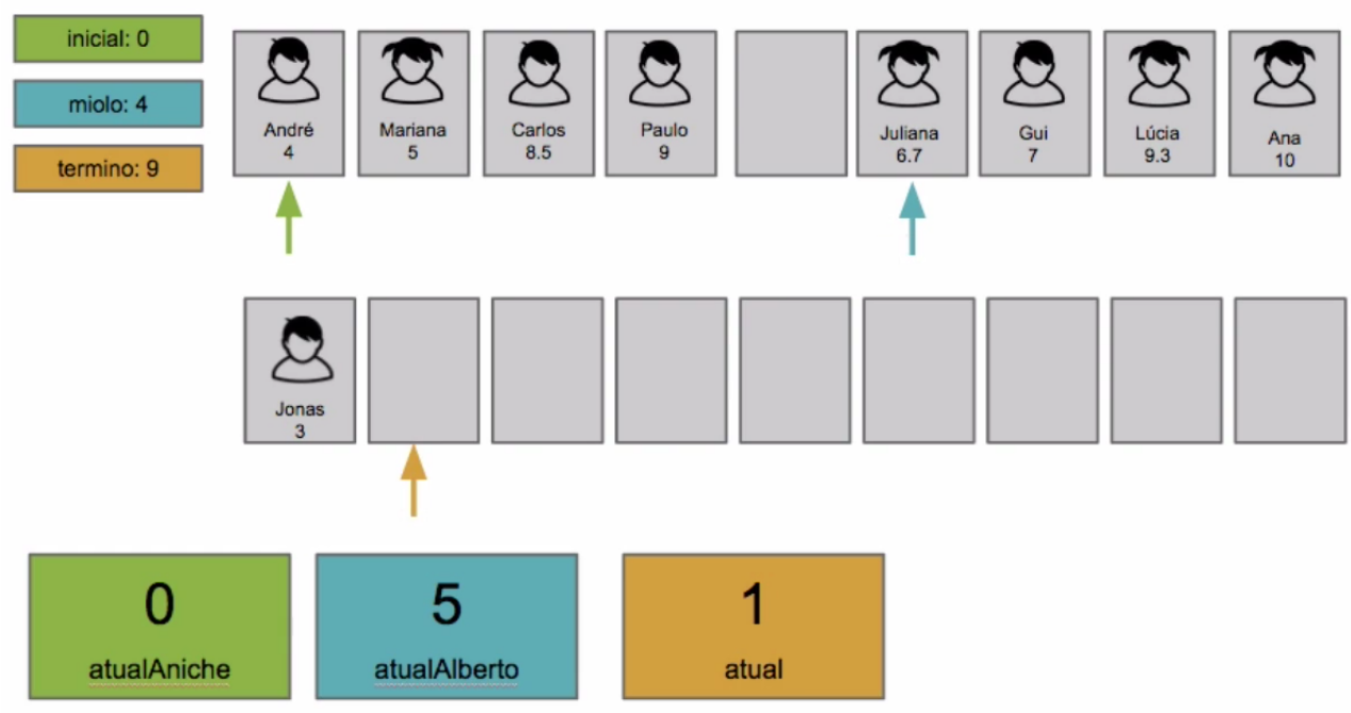


A melhor maneira para começarmos é pelo `atualAlberto`, que inicia no `miolo`. Isto significa que começaremos pelo valor 4. Vamos rodar o nosso algoritmo.

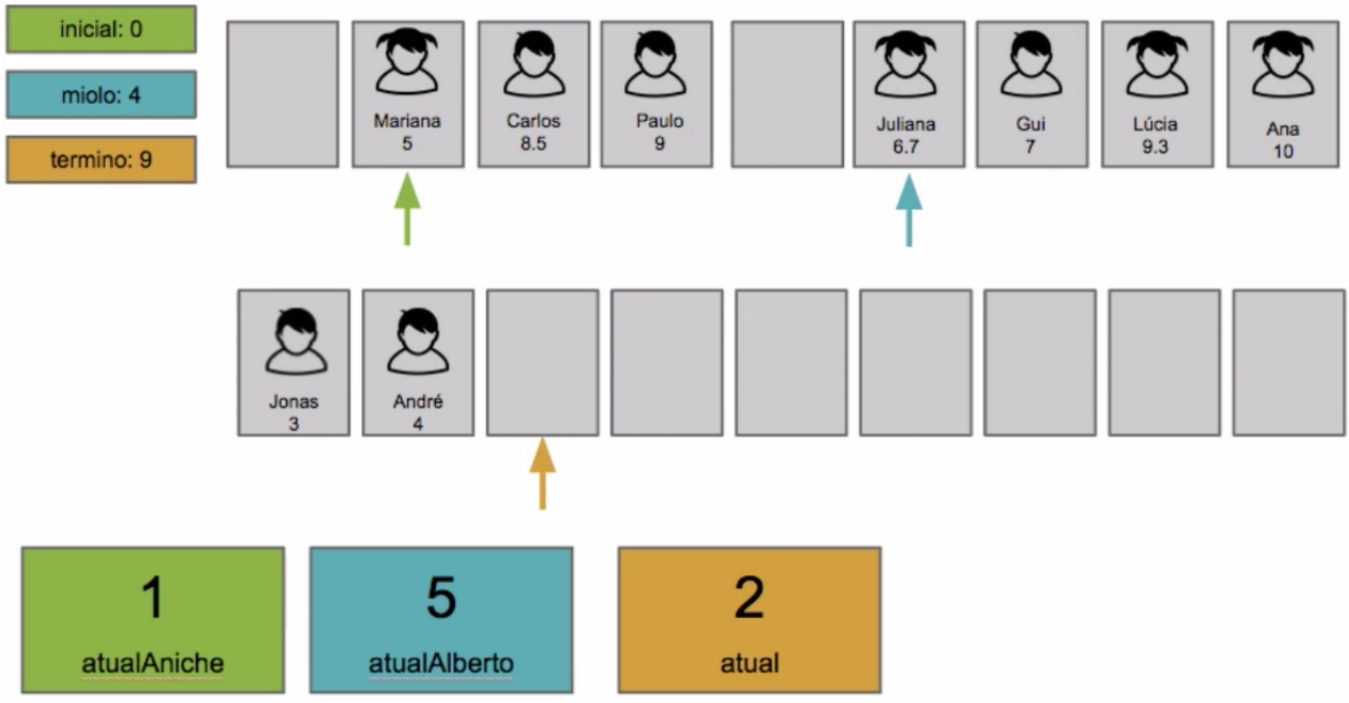


Iremos comparar as notas do André, que está no `atualAniche`, e do Jonas, que está no `atualAlberto`. Qual dos dois é o menor? Você irá perceber que é o mesmo algoritmo utilizado antes!

O menor elemento é o Jonas, então iremos movê-lo para o *array* geral, somamos +1 nas variáveis `atualAlberto` e `atual`.



Vamos comparar agora o André com a Juliana. Qual é o menor? É o André, por isso iremos movê-lo para o novo *array*. Em seguida, somaremos +1 no `atualAniche` e no `atual`.



Iremos continuar o algoritmo como fizemos anteriormente. Observe o que foi feito: decidimos que o `atualAlberto` começaria pelo `miolo`. Nossa função de ordenação tem que saber agora onde ficará o `inicial`, o `miolo` e o `termino`. O `atualAniche` irá começar do 0, como já era feito antes. O `atualAlberto` começará pelo `miolo`. Enquanto o `termino` é o tamanho do *array*. O algoritmo continua o mesmo. A única diferença é que iremos iniciar o `atualAlberto` pelo `miolo`.

Então ao invés de recebermos dois *array*s, nós receberemos um único *array*. Nós somos capazes de intercalar as duas partes, a primeira que vai 0 até o miolo, e a segunda segue do miolo até o fim.