04

Ordenando cartas de baralho, jogando buraco, jogo da vida, provas do enem etc

Ordenando cartas de baralho, jogando buraco, jogo da vida, provas do Enem etc

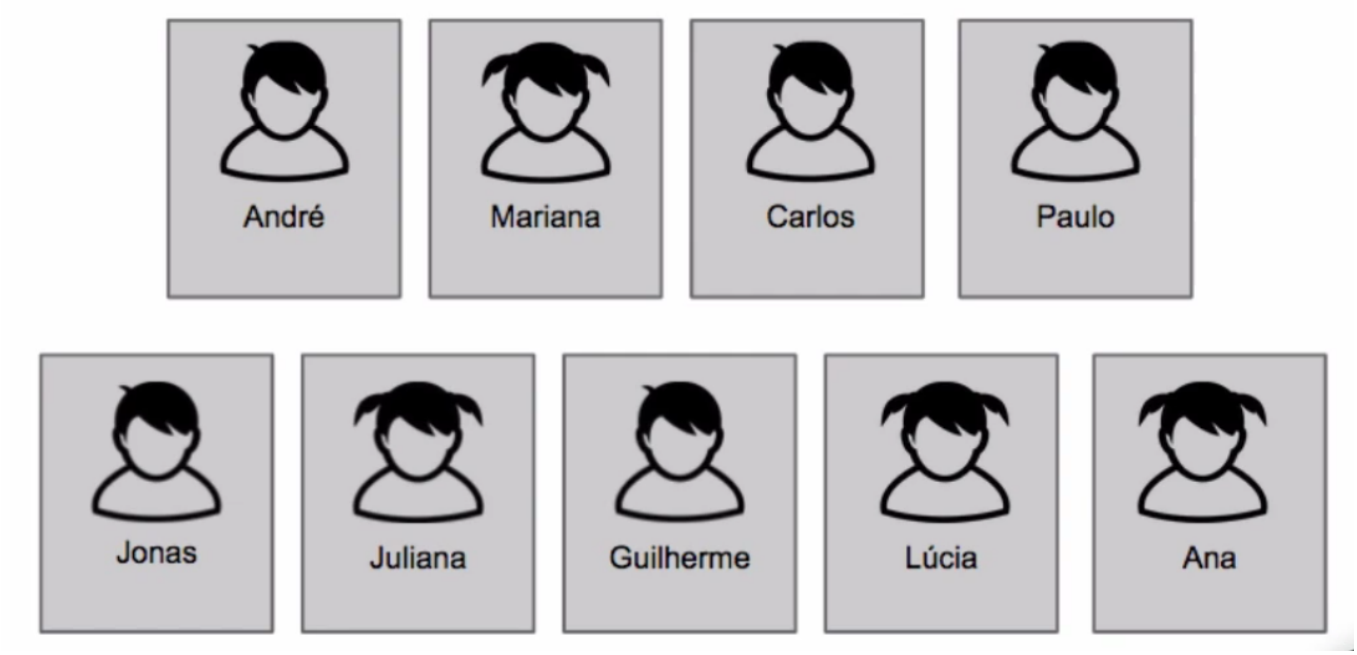
O que acontece quando eu recebo cinco cartas de baralho? Em diversos jogos é interessante que as cartas menores fiquem separadas em uma parte, enquanto as maiores fiquem em outra. Assim, fica mais fácil encontrar os elementos que tenho na mão. Como é costume trabalhar desta maneira, é comum que as pessoas façam esta ordenação manualmente.

É simples ordenar cinco cartas do poker, três cartas do truco. Mas quando ordenamos treze cartas do buraco, já aumenta a dificuldade. Quanto maior for a quantidade de itens para serem ordenamos, mais trabalhoso será o processo. Por quê? Porque os algoritmos que as pessoas usam são o *Selection Sort* e *Insertion Sort*, e à medida que o número de elementos cresce, eles começam a ficar muito lentos. Serão muitas operações e trocas, e isto justifica a demora. Nós não queremos passar por isto. Vamos usar um exemplo que tenha mais de 13 cartas? Imagine um jogo de buraco ou de poker, em que no fim da partida todas as cartas estejam na mesa. E então, você pede para que alguém ordene todas as cartas... Ninguém gostaria de fazer isto. Outro exemplo, seria o jogo da vida. No fim da partida, precisamos que alguém reordene as notas do dinheirinho falso. Ninguém quer ser a pessoa que executará a tarefa de ordenação. No fim do jogo de baralho, do jogo da vida, ou organizar as notas de 300 alunos, ninguém quer ficar responsável por ordenar, porque são muitos elementos. É trabalhoso fazer um *Selection Sort* e um *Insertion Sort*, porque teremos que executar muitas operações. Será que não existe outros algoritmos ou processos mais rápidos?

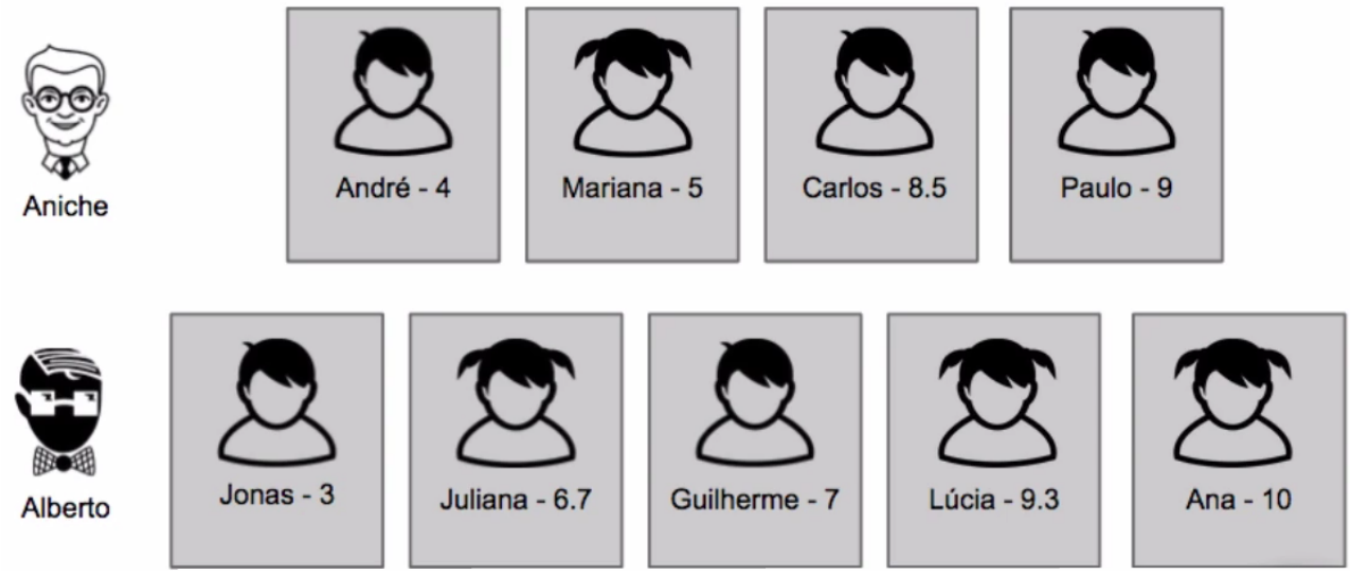
Eu gosto de encontrar formas mais simples, e por isso, vou contar um segredo que costuma funcionar e é utilizado por muitas pessoa... Quando o jogo de buraco acaba e todas as cartas estão na mesa, se alguém decide ordenar os itens, devemos agrupar todos eles e dividi-los entre os participantes. Se temos um monte de cartas - ou de dinheirinho - e tivemos uma partida com quatro jogadores, distribuímos 1/4 do total de elementos para cada participante e todos serão responsáveis pela ordenação da parte recebida. **Ordenar 1/4 do total de cartas é menos trabalhoso do que organizar todos os itens!** Após cada um ordenar a sua parte, reagrupamos as cartas em um único monte que já estará organizado.

O que nós fizemos? Quebramos o nosso problema em quatro pedaços (poderiam ter sido um número maior ou menor) e depois de organizados, juntamos as partes. Podemos fazer o mesmo processo com as provas de alunos. Por exemplo, vamos corrigir as provas do Enem e precisamos ordenar 1 milhão delas. Que trabalhoso executar o *Insertion Sort*! E se encontrássemos uma outra maneira de fazer a ordenação?

Vamos usar a maneira que já conhecemos ao organizarmos as cartas de baralho, o dinheirinho do Jogo da Vida... Usaremos o exemplo das provas dos nove alunos:



Ao invés de ordenar todos os itens, eu divido os alunos entre duas pessoas: o Aniche e o Alberto. Então, cada um deles será responsável pela ordenação considerando as notas dos alunos. Vamos observar como ficam os grupos do Aniche e do Alberto ordenados:



Agora que já temos os dois grupos ordenados, o nosso trabalho será uni-los. A sacada quando trabalhamos com um número grande de elementos, é dividir a tarefa entre as pessoas. No nosso exemplo, o grupo de aluno foi dividido entre o Aniche e o Alberto e cada um teve que organizar a metade.

Então, o problema que quero resolver é: considerando o grupo do Aniche e o grupo do Alberto já ordenados - ou seja, dados os *arrays* ordenados dos dois - o que teremos que fazer é intercalar os elementos das duas sequências para ter um *array* todo organizado.

Então, dado o grupo do Aniche e o do Alberto, o que farei é intercalar todos os objetos para que eles fiquem ordenados em um único *array*. Dado dois *arrays* ordenados, intercale os elementos e monte um *array* ordenado. Se formos capazes de fazer isto, será um enorme avanço, porque basta dividir a ordenação entre as pessoas e depois, unir os elementos ordenados - tarefa que já sabemos fazer.

Vamos tentar unir dois *arrays* de uma maneira mais rápida do que faríamos com outros algoritmos. E assim conseguir ordenar uma quantidade de dados muito maiores. Com o algoritmos que estamos montando, nós seremos capazes. O

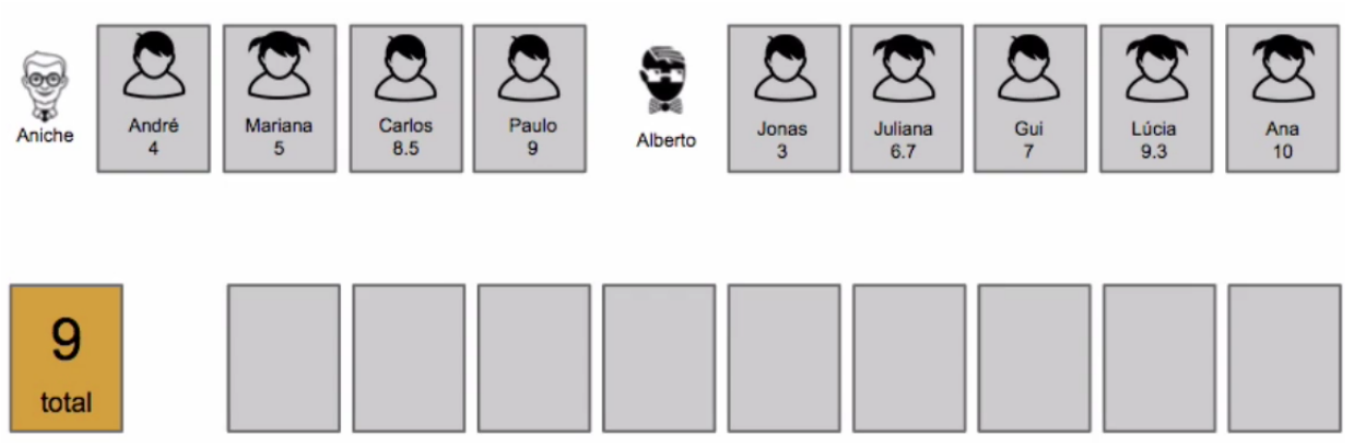
primeiro passo será: considerando que os grupos do Aniche e do Alberto já estão ordenados, como iremos unir estes dois?

Como juntar ou intercalar duas listas ordenadas

Baseado no que usamos cotidianamente, no jogo de truco, no buraco, no Jogo da Vida, qualquer atividade que exija a distribuição de diversos itens. Podemos distribuir entre os participantes para que todos ajudem a organizar e depois juntamos tudo novamente. Queremos resolver o problema de reagrupar os elementos.

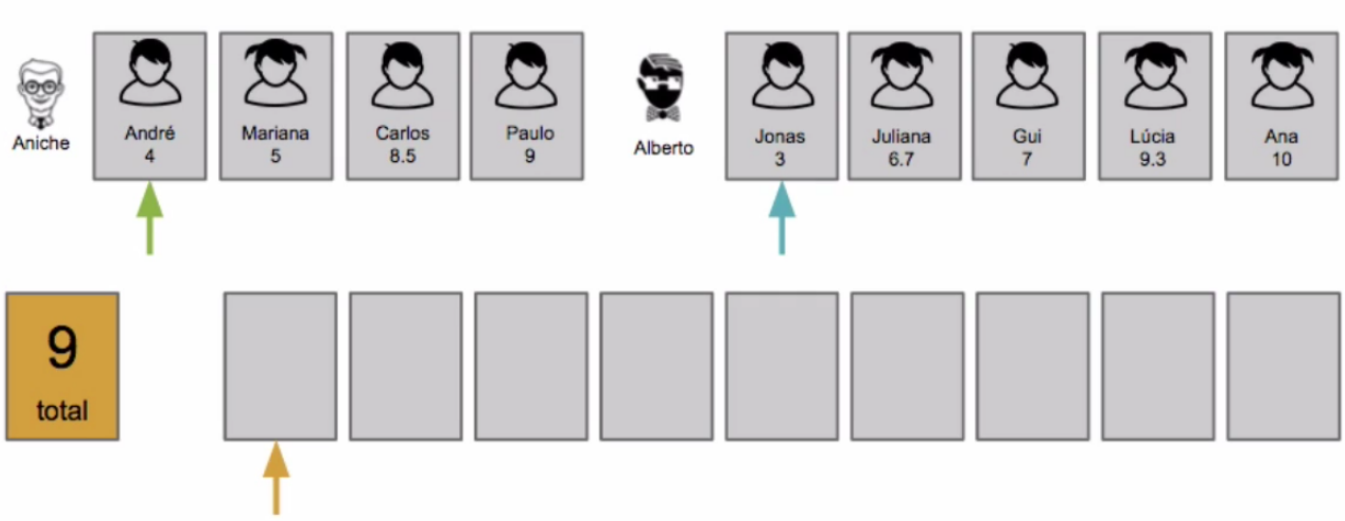
Considerando os alunos do Aniche e do Alberto, que já foram ordenados, como podemos intercalar os nove alunos? Teremos trabalhar com a variável de 9.

Sempre que estamos trabalhando com ordenação, em linguagem de programação que o *array* já nos diz qual o número total de elementos, nós conseguimos extrair o número total. Como queremos generalizar para qualquer linguagem de programação, temos que saber o total de alunos. Que no caso são 9 alunos, iremos reagrupá-los em um único *array* que caiba todos eles. Então, nosso primeiro passo será criar um *array* com um tamanho que caiba os 9 elementos.

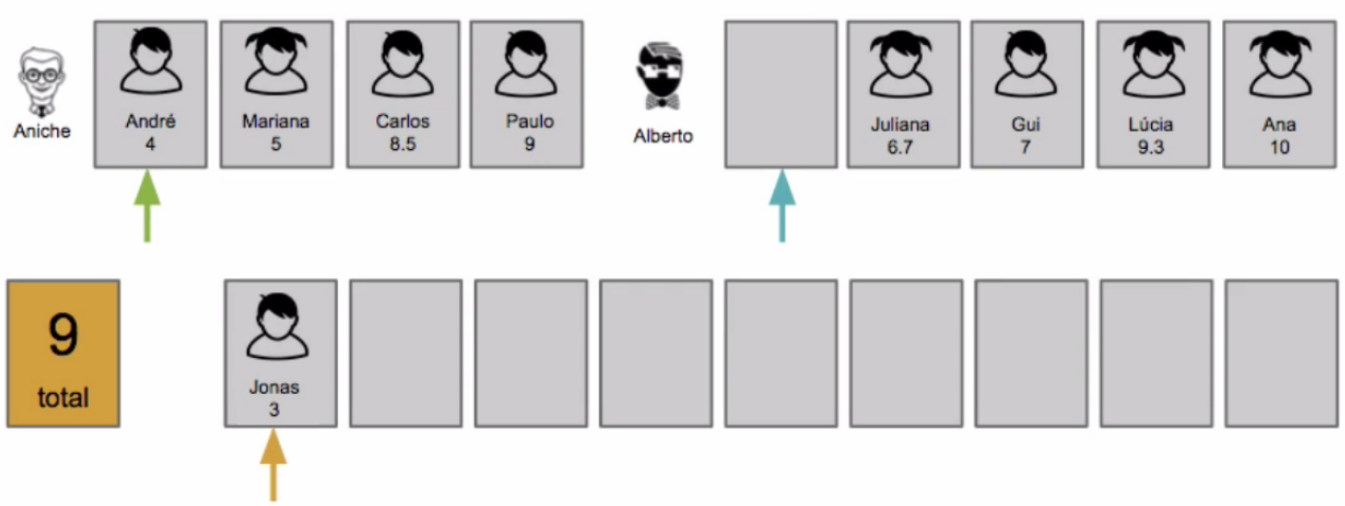


Caso contrário, não teremos como juntar todos os elementos em um único *array*. Agora que temos um lista com tamanho 9, podemos começar.

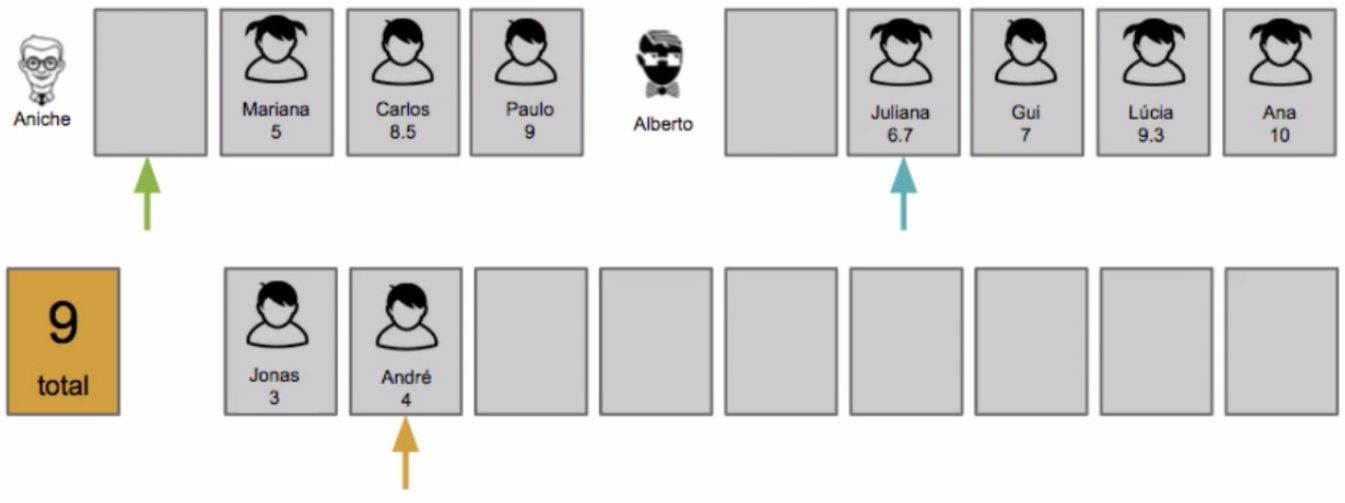
Iremos iniciar com o primeiro elemento de cada grupo ordenado, porque começar pelo meio não faria sentido. Seguiremos a opção mais simples.



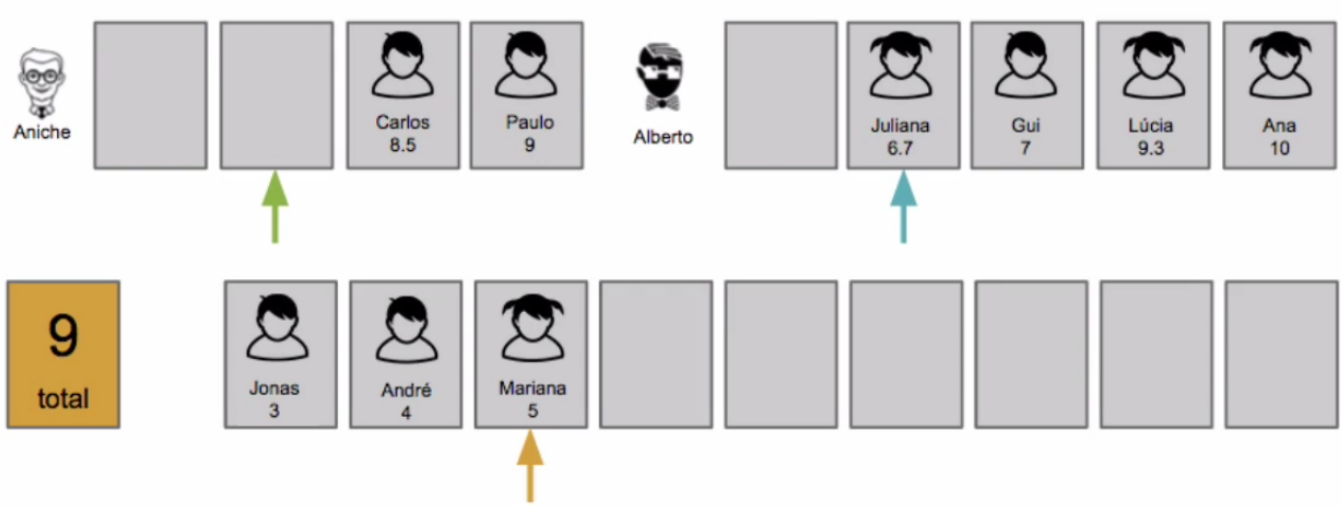
Vamos comparar o André com o Jonas. Um teve uma nota 4 e o outro teve uma nota 3. O Jonas não se saiu bem nas provas! Ele será o menor do nosso novo *array*.



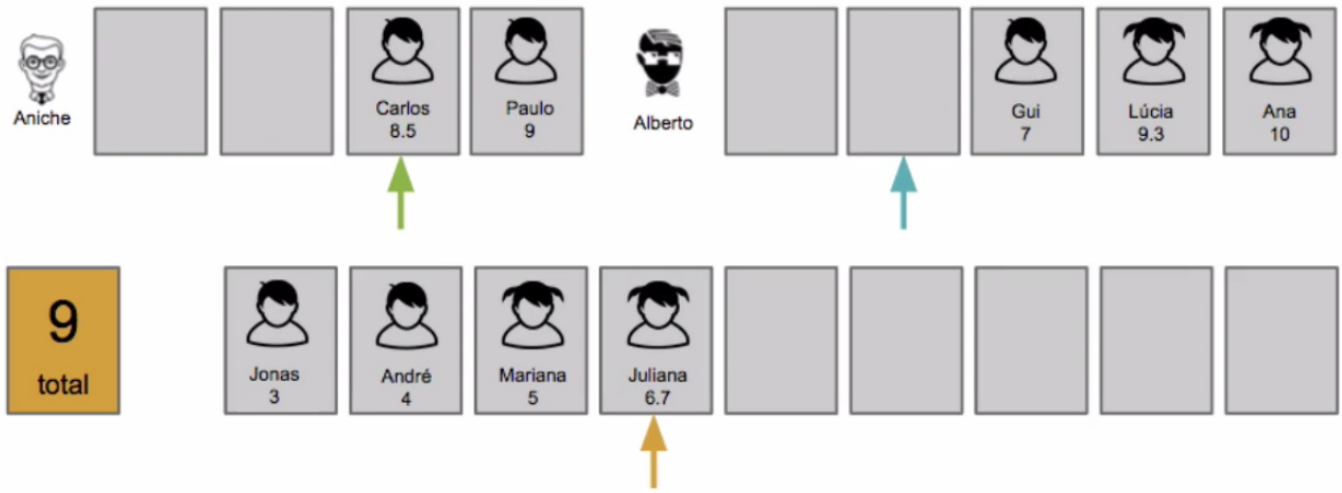
Como já colocamos o Jonas no início da lista, vamos comparar agora o próximo dos grupos do Aniche e do Alberto: o André e a Juliana. Um aluno tirou uma nota 4 enquanto o outro tirou uma nota 6,7. Qual dos dois tirou uma nota menor? O André.Vou descer o André para o *array*.



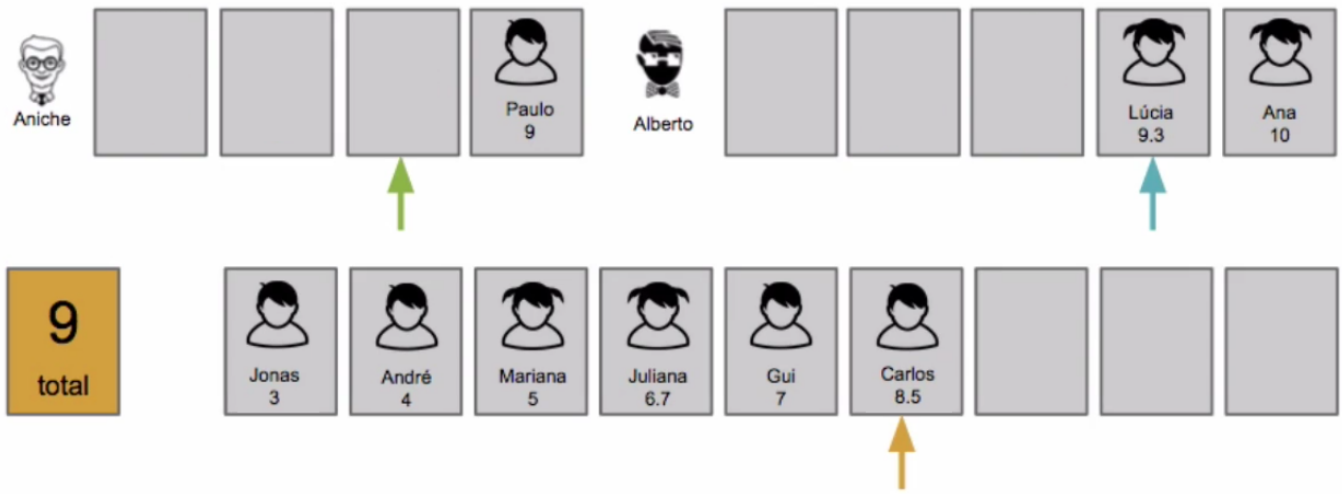
Seguimos comparando as próximas alunas de cada grupo: Mariana e Juliana. Qual das duas tirou uma nota menor? A Mariana. Vamos desce-la para o *array*.



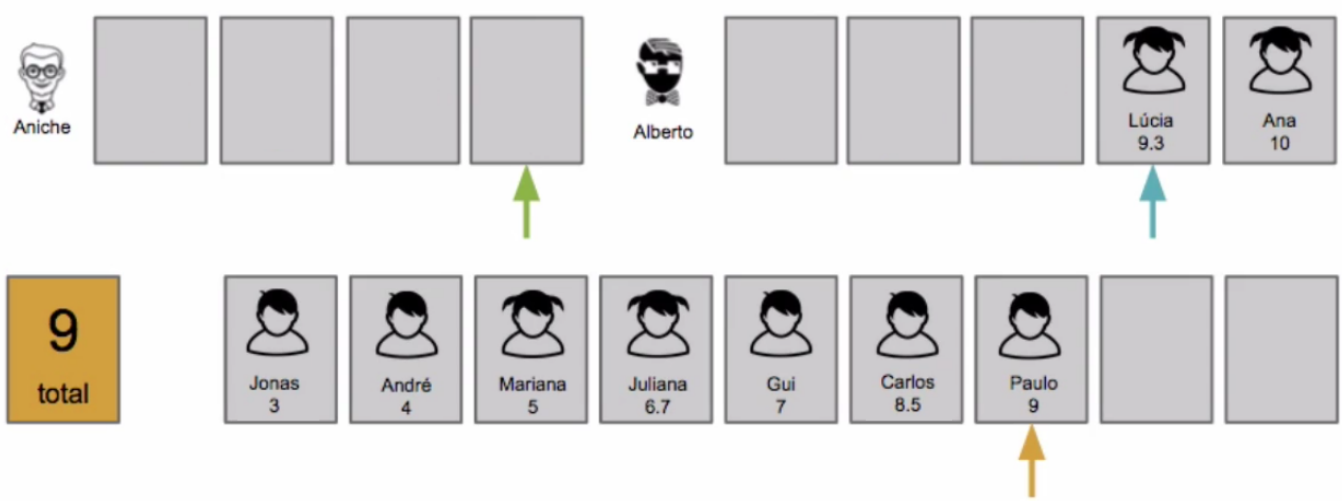
Os próximos alunos a terem as notas comparadas serão: o Carlos e a Juliana. Quem se saiu pior na prova? A Juliana. Vamos desce-la para o novo *array*.



Iremos comparar as notas do Guilherme e do Carlos. O Gui tirou uma nota 7, enquanto o Carlos tirou 8,5. Quem teve a menor nota? O Guilherme. Então, vamos desce-lo para o *array*.

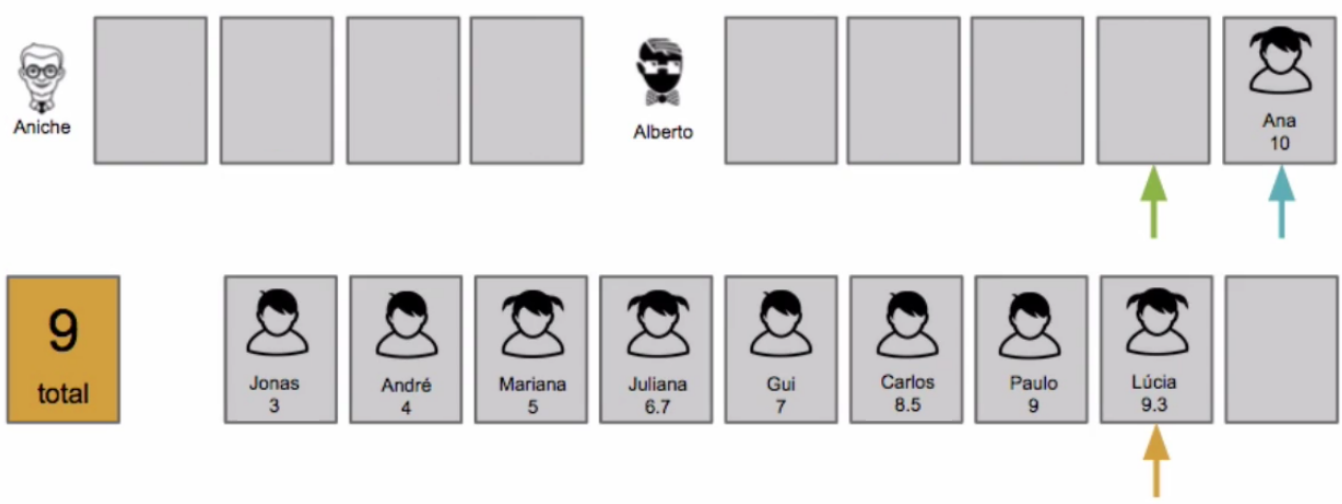


Seguimos para os próximos alunos. Quem se saiu melhor na prova, o Carlos ou a Lúcia? A Lúcia. Logo, é o Carlos que irá descer para o *array*.

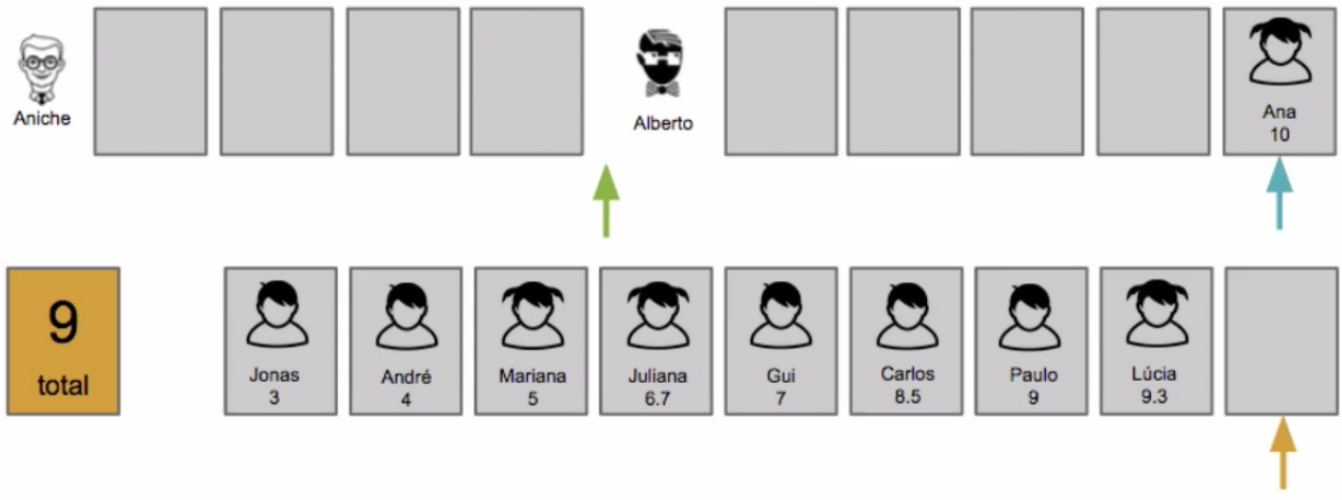


Comparando o Paulo e Lúcia, quem se saiu pior na prova? O Paulo. Vamos desce-lo para o *array*.

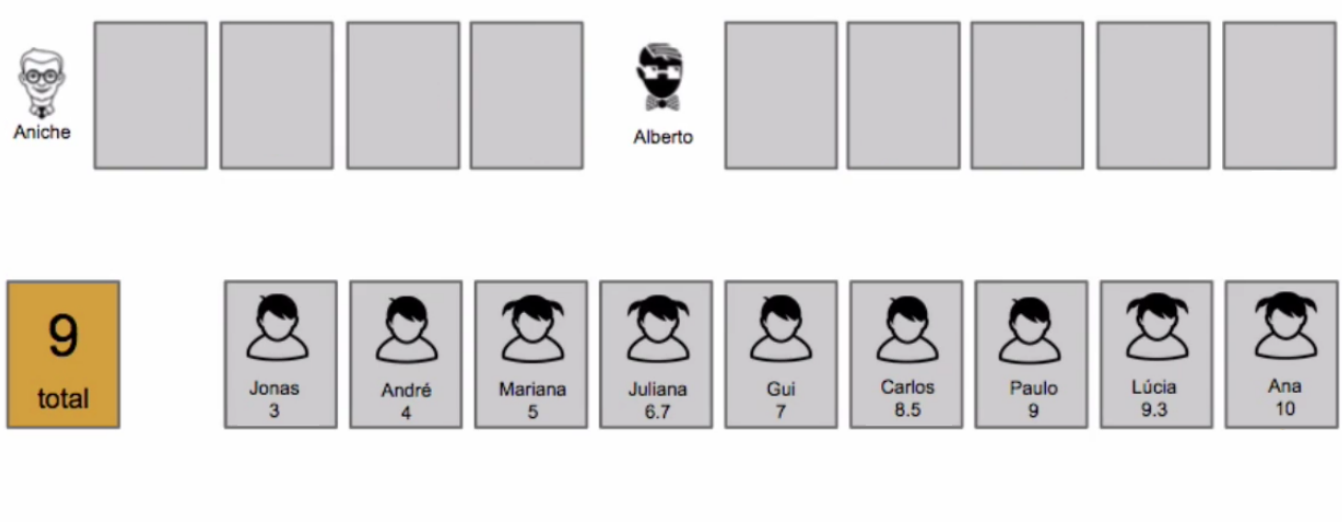
Então, um por um, fui comparando os alunos e o menor, foi sendo colocado no *array*. Com os dois alunos restantes, não faz mais sentido procurar o menor. Agora que já descemos todos os alunos de um dos grupos, vamos levar as duas restantes para o *array*. Colocaremos a Lúcia...



E depois, a Ana.



No fim, quando terminamos as comparações, sobraram duas alunas do grupo do Alberto. Pegamos os dois elementos e descemos para a lista única.

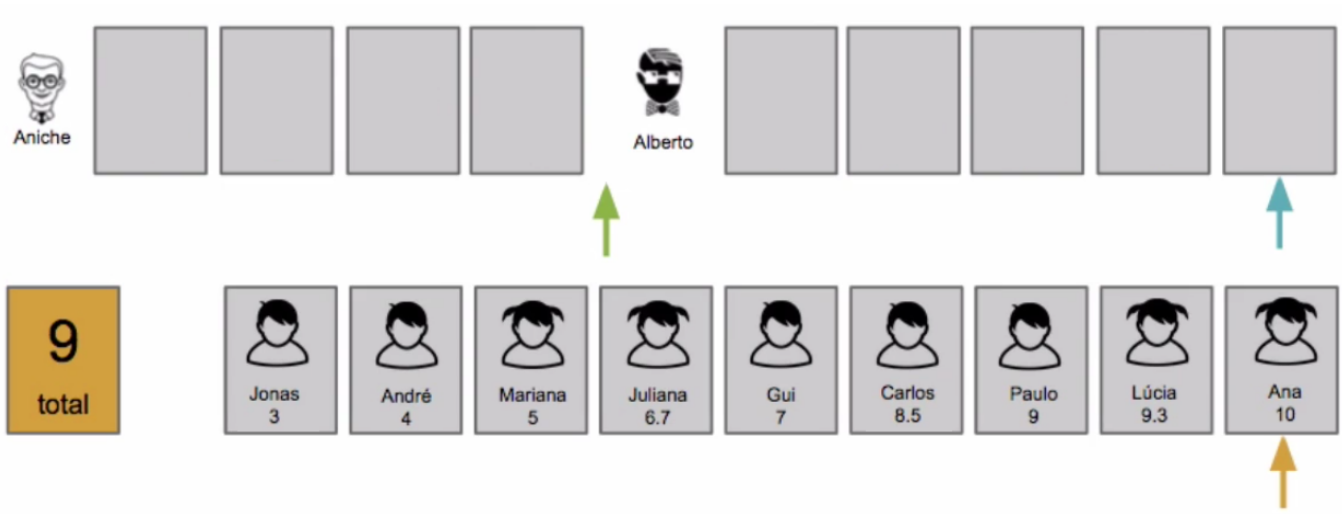


O que fizemos? Comparamos o elemento de um grupo com o de outro e identificamos qual era o menor. O maior permanecia e o menor descia para o novo *array*. Seguimos comparando as notas dos alunos e os menores eram colocados na lista abaixo. No fim, temos um *array* quase completo, porém sobraram algumas alunas no grupo do Alberto. Nós simplesmente descemos as duas e o *array* ficou ordenado.

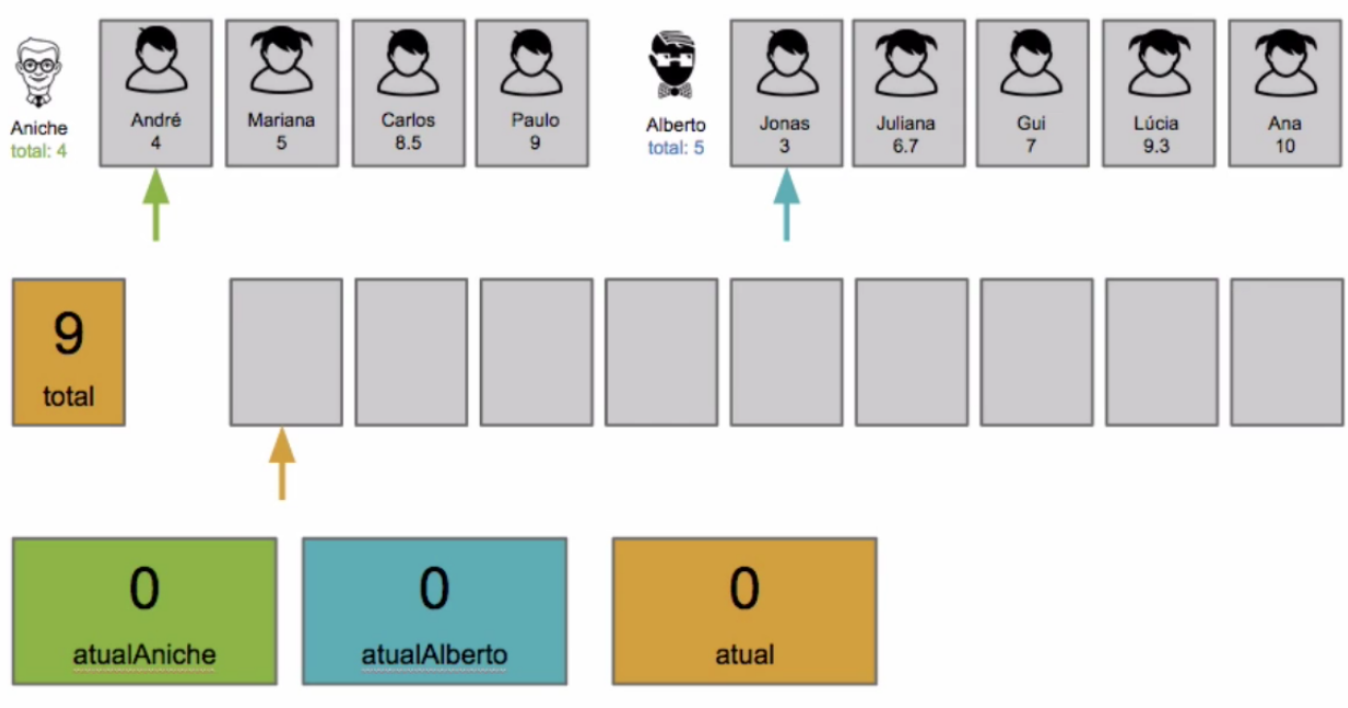
Com isto, na próxima vez que formos jogar baralho e todas as cartas terminarem embaralhadas, já sabemos como organizar: dividimos o monte em partes e distribuímos para os participantes ordenarem com seus próprios algoritmos. No fim, analisamos os grupinhos de cartas e comparamos os itens: "A menor carta é minha, coloca no monte. Agora é a sua, coloca no monte". As menores cartas serão agrupadas em um novo monte, até que não seja mais possível compará-las e as cartas estão ordenadas.

Simulando com as variáveis

Então já sabemos intercalar dois *array*s ordenados. Agora, quais variáveis eu preciso para ver isto acontecer?

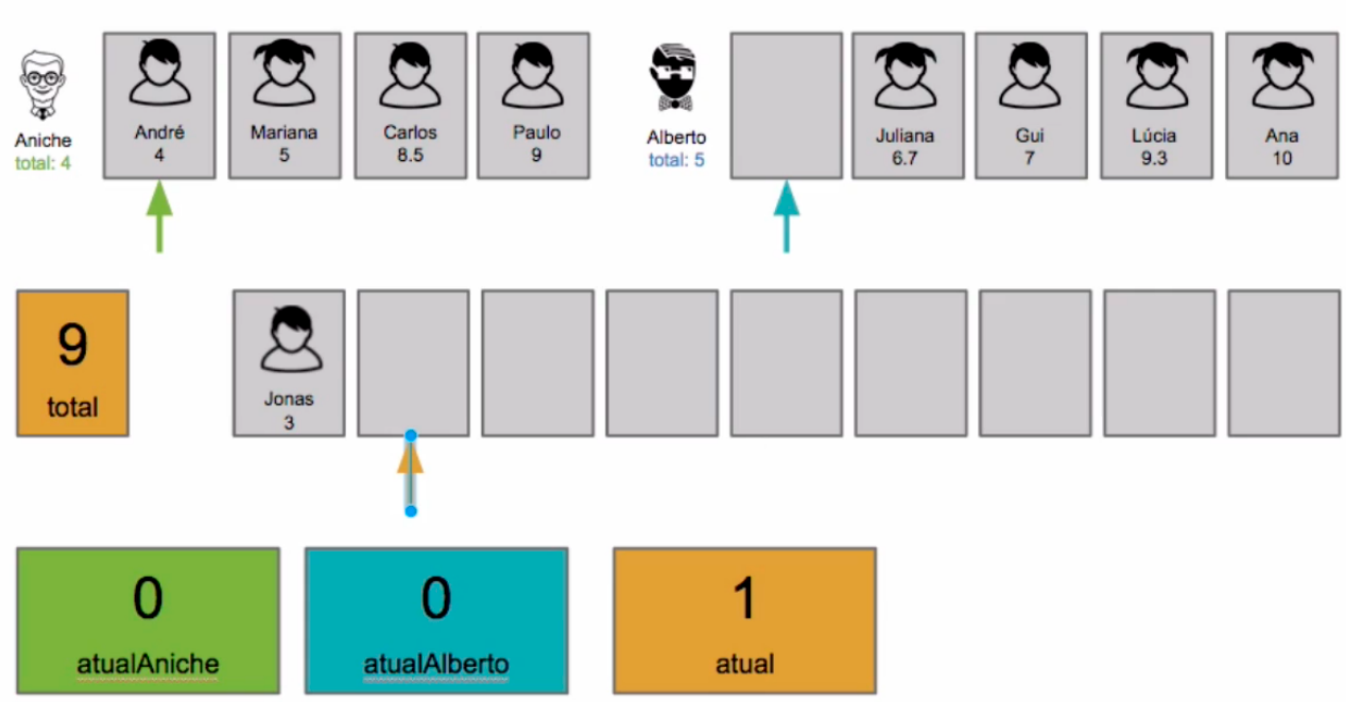


Nós utilizamos três setinhas, três variáveis, que nós brincamos e mudamos os valores. Além da variável referente ao total de elementos, que precisávamos saber desde o principio.

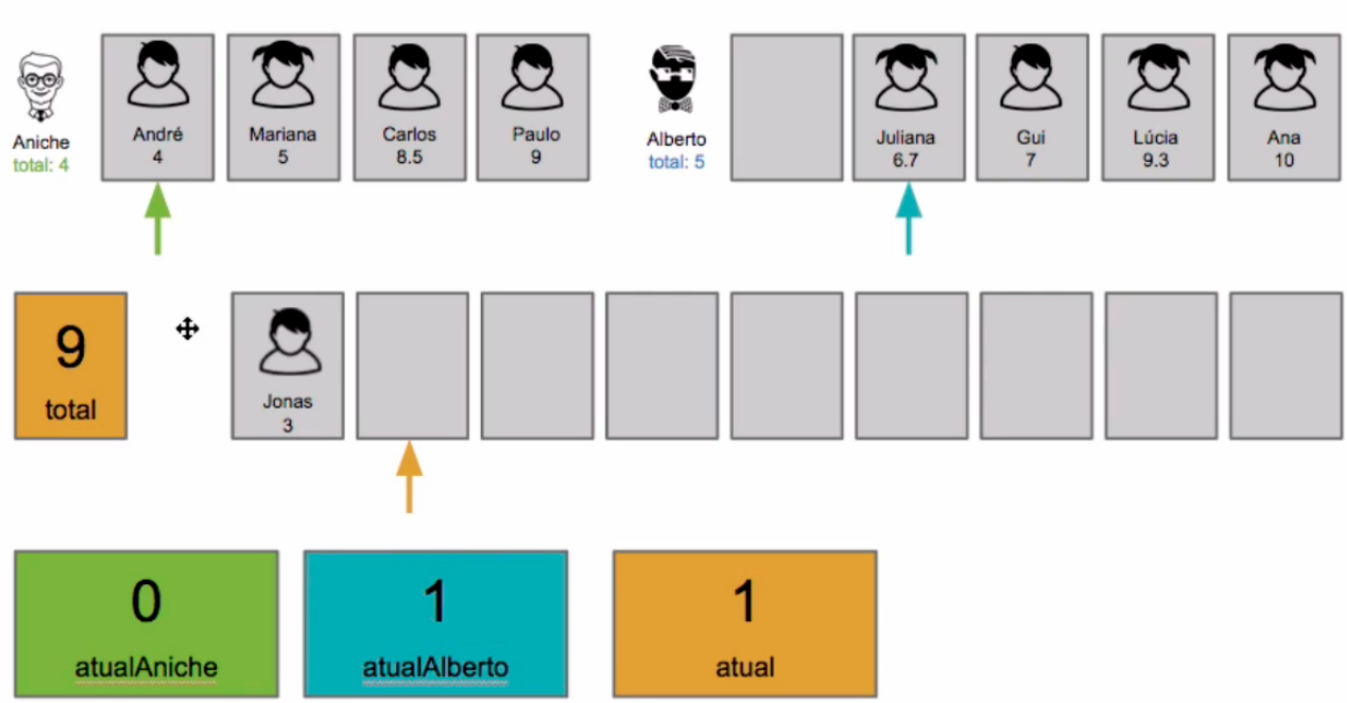


Com o total, nós criamos o tamanho do nosso *array*, precisamos descobrir qual era o aluno **atual** do Aniche e qual era o do Alberto. Também precisávamos saber o número total de elementos em cada um dos grupos e assim vamos brincando com o atual das três listas, incluindo a geral. Como farei isto? Vamos simular de novo o algoritmo que usamos no exemplo, mas agora usaremos as variáveis.

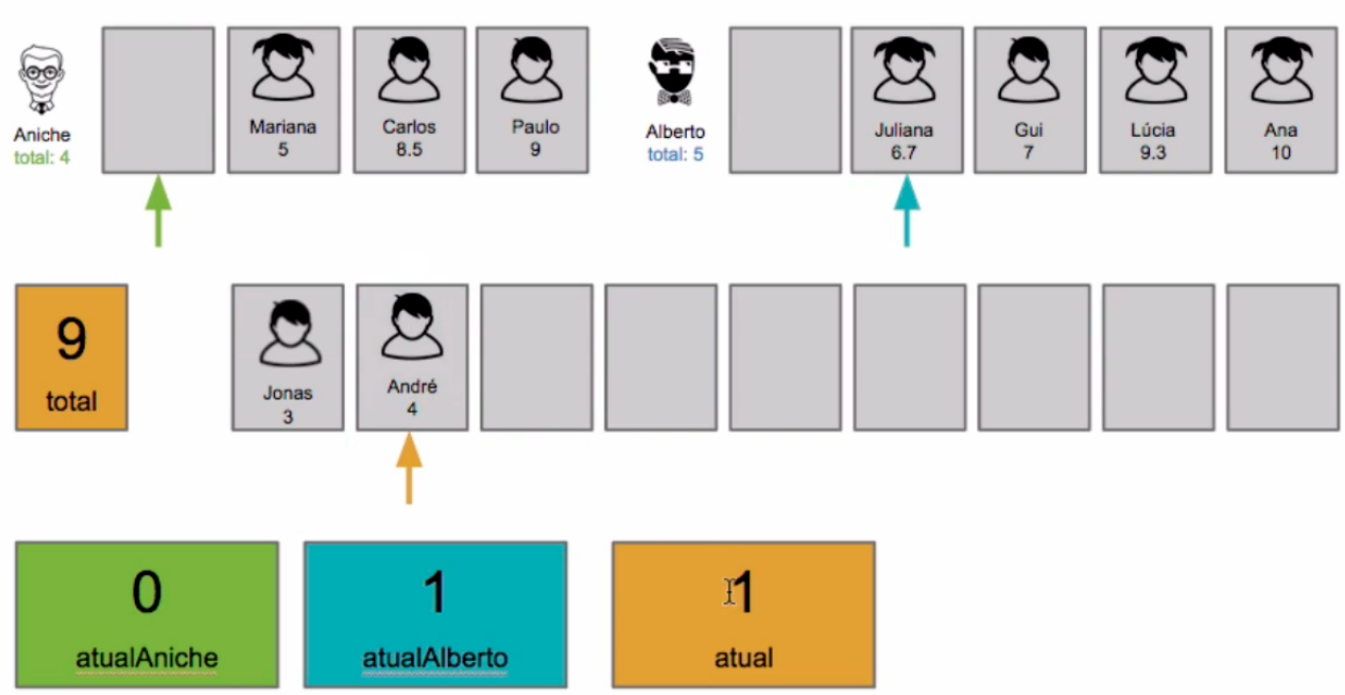
Quando comparamos as notas dos alunos André e Jonas, qual dos dois teve o pior resultado? O Jonas. Então, iremos movê-lo para a posição atual do *array* geral. Em seguida, o que devemos fazer? Vamos somar 1 com a posição atual.



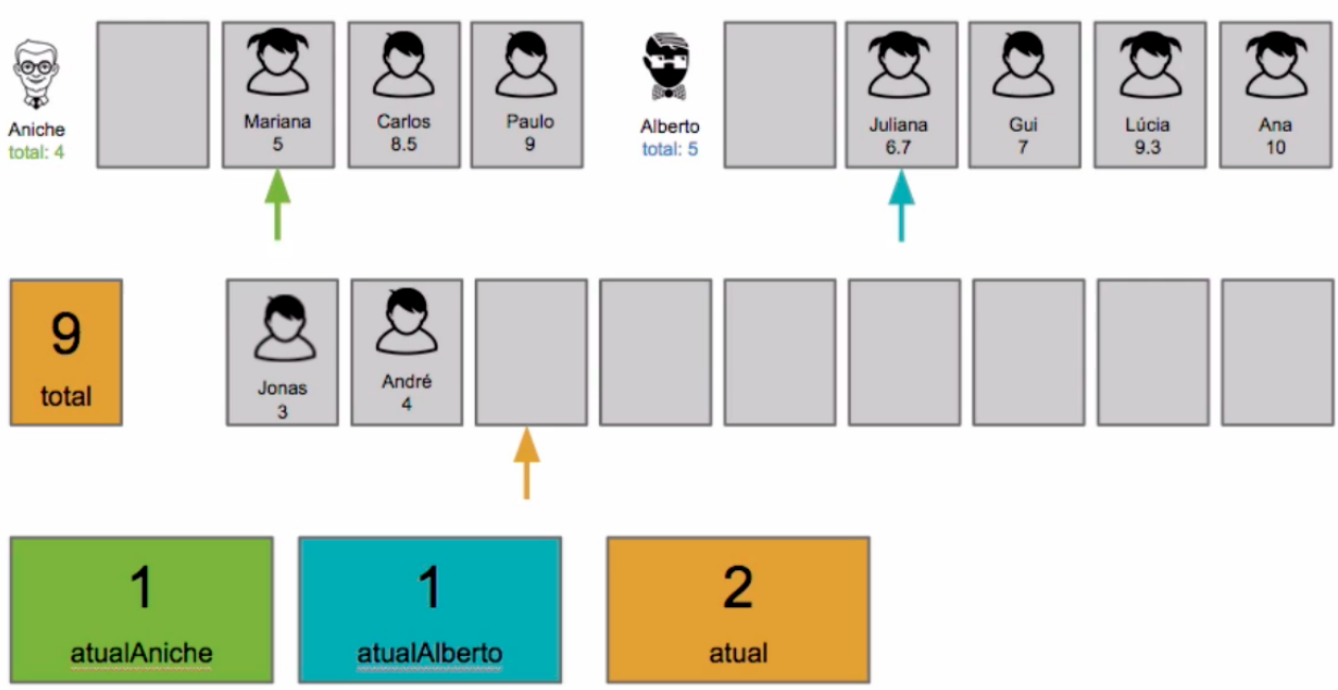
Faremos o mesmo com a atual do Alberto, porque queremos analisar o próximo elemento.



O elemento atual do Aniche (André) é maior ou menor que o atual do Alberto (Juliana)? É menor. Então, iremos movê-lo para a posição atual do *array* geral.



Para continuar, precisamos aumentar 1 do atual e 1 do atual do Aniche, porque queremos analisar o próximo item do grupo dele.

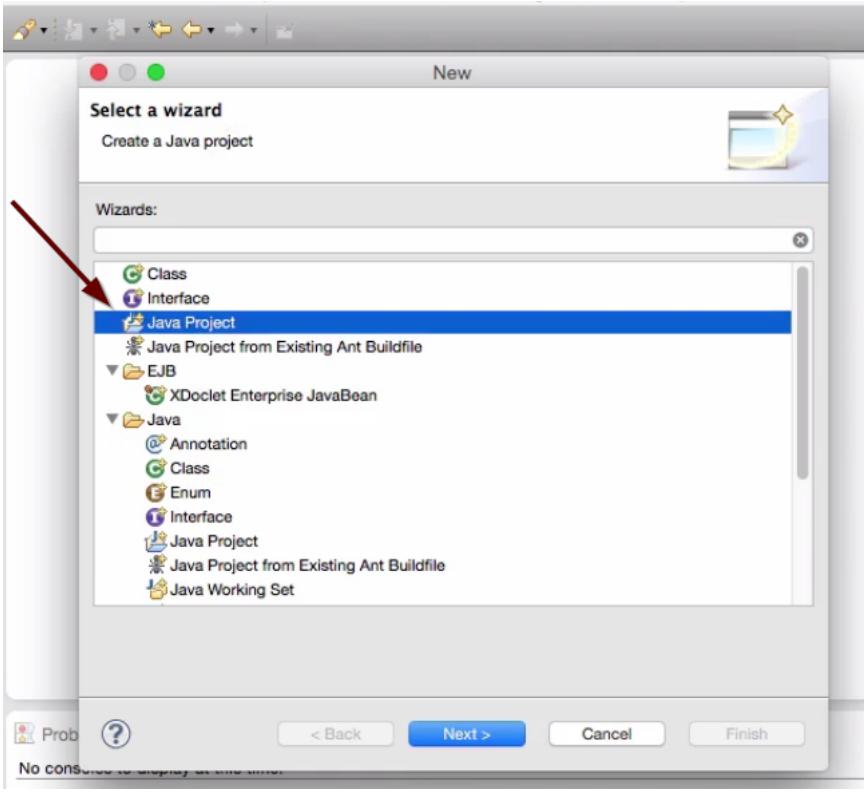


Isto significa que precisamos ter três indicadores apontando para qual posição estamos trabalhando em cada um deste *array*s.

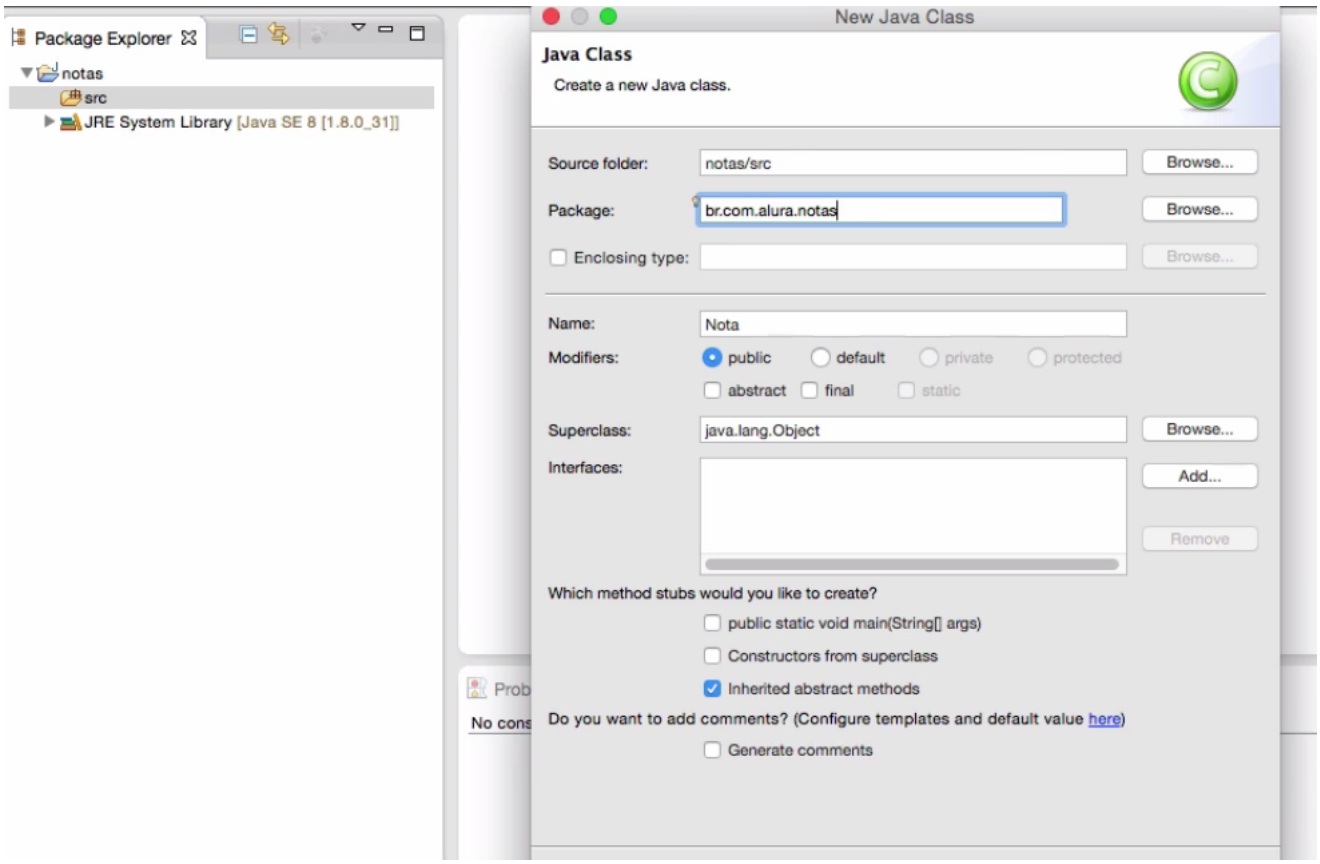
Estou trabalhando com o aluno na primeira posição do grupo do Aniche, do grupo do Alberto e do nosso novo *array*. Depois vamos caminhando com os indicadores, colocando-os nas posições adequadas. Estas são as variáveis que iremos precisar logo em seguida. Vamos implementar o código?

Criando o projeto e preparando para juntar ordenado

Vamos criar o nosso projeto, que será um *Java project*.



Ele será chamado **Notas**. Dentro, teremos um classe para representar cada uma das notas que foram tiradas pelos alunos. A classe irá receber o nome **Nota**, no pacote **br.com.alura.notas**.



Esta nota tem o nome do aluno (`String aluno`) e tem o valor da nota, que pode sr quebrado (`valor`), como 7,5 ou 9,7, por exemplo. O valor será entre 0 e 10.

```
package br.com.alura.notas;

public class Nota {

    private String aluno;
    private double valor;
}
```

Quero adicionar um construtor que receba o o nome do aluno e o valor das notas. Ele aceitará tanto o aluno, como o valor:

```
public Nota(String aluno, double valor) {
    this.aluno = aluno;
    this.valor = valor;
}
```

Iremos criar o *getter* apenas quando for necessário. Momentaneamente, é o suficiente. Fecharemos esta classe e em seguida, criaremos um arquivo de teste, porque criar diversas notas, já ordenadas por dois professores. Para depois pedir para o programa reunir estas notas. Iremos implementar a **junção**, quando fazemos o ***merge***. Vamos juntar, fundir os dois grupos de notas ordenadas.

Iremos fazer o `TestaMerge` , vamos testar fundir as listas, com o método `main` . Vamos trabalhar dentro desta classe, digitando diversas notas de alunos.

```
public class TestaMerge {

    public static void main(String [] args) {
        mariana 5
        andre 4
        paulo 9
        carlos 8.5
        juliana 6.7
        jonas 3
        lucia 9.3
        ana 10
        guilherme 7
    }

}
```

Temos disponíveis as notas que iremos trabalhar. Os alunos estão divididos em dois grupos. No primeiro, teremos: Mariana, André, Paulo e Carlos. No segundo, teremos: Juliana, Jonas, Lucia, Ana e Guilherme. O que queremos fazer é juntar os elementos em uma única lista.

O primeiro grupo é do professor Maurício. Então, iremos criar um *array* indicando isto:

```
public class TestaMerge {

    public static void main(String [] args) {
        Nota[] notasDoMauricio = {
            new Nota("mariana", 5),
            new Nota("andre", 4),
            new Nota("paulo", 9),
            new Nota("carlos", 8.5)
        };
    }
}
```

O professo Maurício foi gentil e já ordenou as notas, do menor para o maior.

```
public class TestaMerge {

    public static void main(String [] args) {
        Nota[] notasDoMauricio = {
            new Nota("mariana", 5),
            new Nota("andre", 4),
            new Nota("carlos", 8.5),
            new Nota("paulo", 9)

        };
    }
}
```

Recebemos também as notas do segundo grupo, que é do professor Alberto. Vamos indicar no código, que as notas são Alberto (`notasDoAlberto`) :

```
    Nota[] notasDoAlberto = {
    juliana 6.7
    jonas 3
    lucia 9.3
    ana 10
    guilherme 7
}
```

As notas do Alberto também já foram entregues em ordem. O professor já fez um *Insertion Sort* ou um *Selection Sort* e recebemos a ordenação pronta.

```
    Nota[] notasDoAlberto = {
        new Nota("jonas", 3),
        new Nota("juliana", 6.7),
        new Nota("guilherme", 7),
        new Nota("lucia", 9.3),
        new Nota("ana", 10)

    };
```

Então, temos os dois grupos de nota ordenados. O faremos agora? Iremos juntar os dois grupos, fundiremos as duas coleções de itens ordenados. Vamos mostrar que somos capazes de unir todos os elementos em uma única lista ordenada. Isto significa que quero criar uma lista com um ranking final, ordenado pelas notas dos alunos, do menor para o maior.

Queremos ter um rank que irá fundir todas as notas:

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

Depois que o programa criar meu rank, quero que ele imprima todos os valores ordenados corretamente. Isto é: para cada nota dentro do rank, vou querer imprimir a nota e o nome do aluno (`nota.getAluno()`).

```
for(Nota nota ) : rank) {
    System.out.println(nota.getAluno());
}
```

Será o `getAluno` que irá devolver o próprio `aluno` :

```
public String getAluno() {
    return aluno;
}
```

Nós precisamos implementar o `junta` . Isto é: dado duas coleções, com os elementos já ordenados, como fazemos para juntar os dois *arrays* e resolver rapidamente esta ordenação? Se já paralelizamos, e dividimos o trabalho entre as pessoas, vamos unir as diversas partes ordenadas. Iremos juntar todos em um.

Este método `junta` recebe as primeiras (`notasDoMauricio`) e segundas (`notasDoAlberto`) notas. É o método que iremos implementar.

Implementando o junta/intercala

Temos que implementar agora a função `junta()` que, dados dois *array*s ordenados, irá uni-los em uma única lista com todos os elementos. Ele irá juntar os dois *arrays* e reorganizar os itens.

Vamos juntar os *arrays*?

```
private static Nota[] junta(Nota[] notasDoMauricio, Nota[] notasDoAlberto) {
    return null;
}
```

Precisamos identificar o número de elementos dos dois grupos, para criar um novo *array* que caiba todos eles. Isto significa que:

```
int total = notasDoMauricio.length + notasDoAlberto.length;
```

Iremos também criar o resultado, que é a lista final.

```
Nota[] resultado = new Nota[total];
```

Nós iremos devolver o `resultado`. O nosso código ficará assim:

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

Ainda falta inserir as notas. Tanto `notasDoMauricio` como `notasDoAlberto` já estão ordenados. Vamos incluir os elementos de ambos no nosso *array*.

Como fizemos a ordenação antes? Verificava qual era a menor nota do grupo do Maurício e depois, do grupo do Alberto. Comparava os dois elementos e colocava o menor na lista geral. Seguimos para o próximo elemento de cada grupo, e depois para o próximo, até não ser mais possível comprar os grupos. Então, precisamos começar com a primeira posição de cada lista. Isto significa que:

```
int atualDoMauricio = 0;
int atualDoAlberto = 0;
```

Assim vamos começar pela primeira posição dos dois grupos. Observo a primeira posição de cada um, quem está lá? Será a nota do Maurício (`nota1`) é a `notasDoMauricio` que está na posição `atualDoMauricio`. A linha ficará assim:

```
Nota nota1 = notasDoMauricio[atualDoMauricio];
```

Qual será a nota 2? Será `notasDoAlberto` que está na posição `atualDoAlberto`.

```
Nota nota2 = notasDoAlberto[atualDoAlberto];
```

Vamos descobrir qual das duas notas é a menor. Se (`if`) a nota 1 (nota1.getValor) for menor do que a nota 2 (`nota2.getValor`), a menor nota será a do Maurício. Senão (`else`), será a do Alberto.

```
Nota nota1 = notasDoMauricio[atualDoMauricio];
Nota nota2 = notasDoAlberto[atualDoAlberto];
if(nota1.getValor() < nota2.getValor()) {
    // mauricio
} else {
    // alberto
}
```

É isto que queremos comparar. Iremos identificar quando uma ou outra é a menor, para então colocá-la no *array* geral. Vamos criar o método `getValor` , que vai devolver um `double` que é o `valor` e salvamos a classe `Nota` :

```
public double getValor() {
    return valor;
}
```

Temos nosso `getValor` . Se a `nota1` for menor que a `nota2` , usaremos a `nota1` . Caso a menor seja a `nota2` , ela que será usada. Temos que fazer isto dentro do laço. Precisamos ir andando dentro do *array*. Isto significa que todo o código deverá ser executado, enquanto pudermos comparar as notas dos dois grupos. Enquanto (`while`) o `atualDoMauricio` não ultrapassar o total do grupo (`notasDoMauricio.length`) e o `atualDoAlberto` não ultrapassar o total do outro grupo (`notasDoAlberto.length`), podemos seguir colocando itens na nova lista.

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

Enquanto o `while` for verdadeiro, devo continuar analisando. Vou recortar parte do `if` do código e colá-lo dentro do `while` :

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

    Nota nota1 = notasDoMauricio[atualDoMauricio];
    Nota nota2 = notasDoAlberto[atualDoAlberto];
    if(nota1.getValor() < nota2.getValor()) {
        // mauricio
    } else {
        // alberto
    }

}
```

Enquanto tivermos elementos para serem analisados, seguimos com o processo de comparação.

Qual é o primeiro elementos de cada grupo? Temos o André, com 4, e o Jonas, com 3. Então, ele irá cair no caso do Alberto. A `nota2.getValor` é referente ao Jonas, e é a menor. Então, vamos colocá-la no resultado. Para isso vamos escrever no nosso código que o `resultado` na posição 0 é igual a `nota2` .

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

Se o menor fosse o do Maurício, iríamos escrever que o `resultado` na posição 0 é a `nota1` .

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

Agora que analisamos a `nota1` e a `nota2` , temos que ir para o próximo do Alberto (atualDoAlberto++). Se o do Mauricio for o menor, também seguiremos para o próximo (atualDoMauricio++)

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

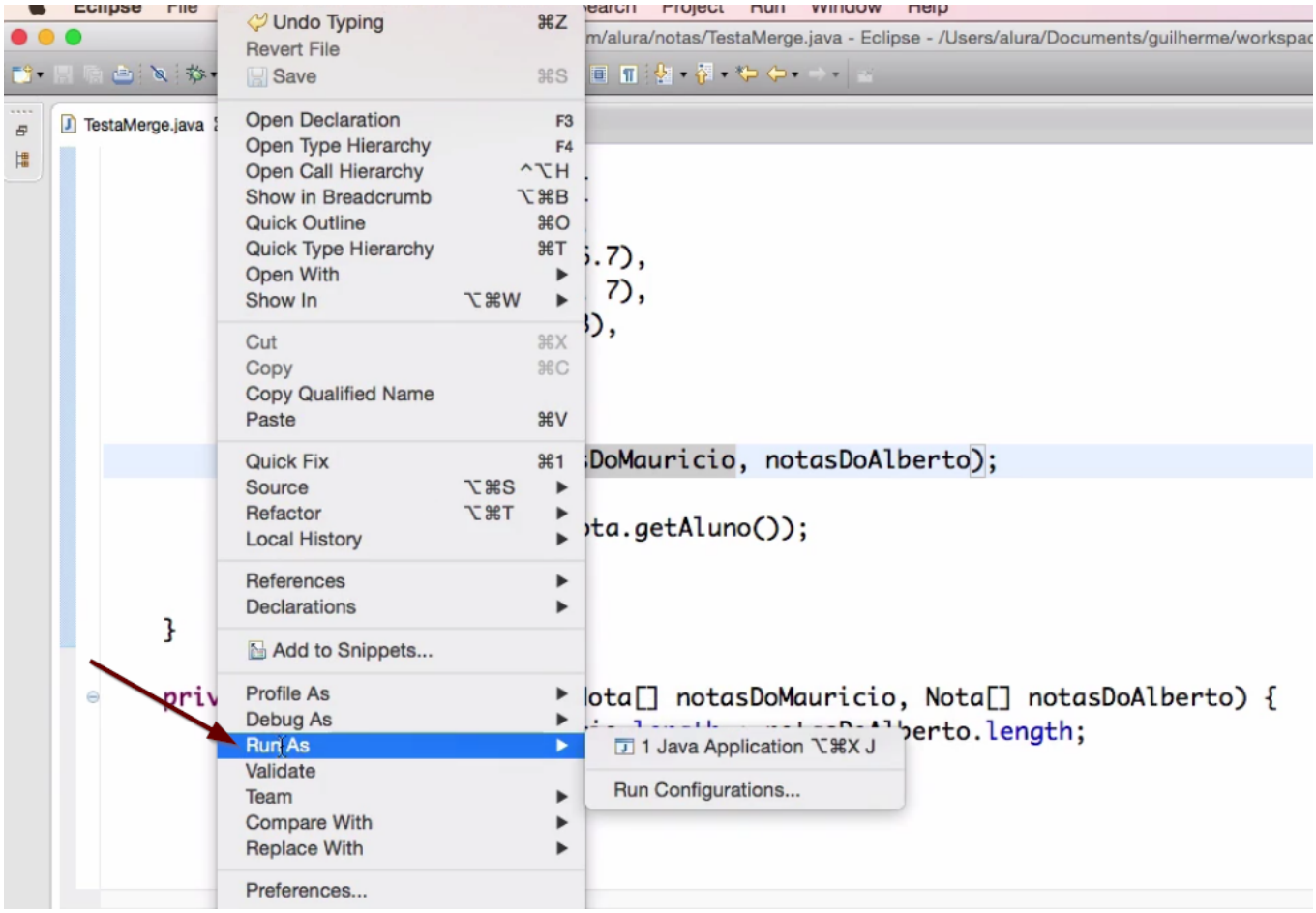
Depois, voltamos para o laço e comparamos novamente os elementos para identificar qual é o menor elemento. Quando descobrimos qual é o menor, o que fazemos com ele? Colocamos na posição 0? Não. Vamos colocando os itens em um pouco mais para frente no *array*. Esta posição onde coloco o elemento tem que ir se deslocando. Logo, iremos precisar de uma posição (`int atual`) que comece no 0. Vamos incluir o atual no resultado também. E indiferente se é o Mauricio ou o Alberto usaremos `atual++` . Vamos sempre passar para a próxima casa...

```
int atual = 0;

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

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

    }
```

Fizemos o nosso código, a parte inicial do `junta()`. O que ele fará? Irá criar um *array* que caiba todos os elementos. Depois, vamos analisar o primeiro de cada grupo e identificar qual é o menor dos dois, para então colocá-lo na primeira posição do resultado. Temos duas notas para compararar? Sim. Após compará-los e descobrir qual é o menor, vamos colocá-lo na posição adequada. Em seguida, avançamos para o próximo. Repetimos o processo até analisarmos o total de elemento. Quando acabar, devolve para o resultado.

Aparentemente, tudo está pronto. Vamos testar o código? Iremos rodá-lo e o programa deve imprimir o nome de todos os alunos. Clicamos em *Run As* e depois em *Java Aplication*



O resultado no console será:

```
jonas
andre
```

```
mariana
juliana
guilherme
carlos
paulo
```

No resultado, além de faltar a Ana, apareceu uma mensagem de erro: `NullPointerException` . Nós quase acertamos, mas ficou faltando algum detalhe.

Iremos descobrir qual é este detalhe, em seguida. Por que a Ana não apareceu?

Logando informações para procurar o erro em um algoritmo

Nós vimos que ao rodar o nosso programa, aconteceu algum erro no fim. Ele imprimiu no resultado sete alunos, porém deixou de fora dois. Quais ficaram faltando? Vamos conferir detalhadamente qual informação estamos imprimindo? Não nos limitaremos aos nomes (`nota.getAluno`), mas também ao valor das notas (`nota.getValor`).

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

Vamos rodar de novo o programa e ele irá imprimir novamente a lista de alunos do Jonas até o Paulo e suas notas.

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

Quais alunos ficaram faltando no resultado? Do primeiro *array* não faltou nenhum elemento. Porém, do segundo, faltaram a Lúcia e a Ana. Por que será que elas não apareceram? Vamos tentar imprimir quem passou pelo processo de comparação. Para cada uma das comparações, iremos imprimir os resultados com o texto:

```
System.out.println("Estou comparando " + nota1.getAluno() + " com " + nota2.getAluno());
```

Vamos voltar a rodar o código com a nova linha e ver o que acontece?

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

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

    if(nota1.getValor() < nota2.getValor()) {
```

```
            // mauricio
            resultado[atual] = nota2;
            atualDoMauricio++;
        } else {
            // alberto
            resultado[atual] = nota2;
            atualDoAlberto++;
        }
        atual++;

    }
```

O programa irá imprimir diversas comparações:

```
Estou comparando andre com jonas
Estou comparando andre com juliana
Estou comparando mariana com juliana
Estou comparando carlos com juliana
Estou comparando carlos com guilherme
Estou comparando carlos com lucia
Estou comparando paulo com lucia
jonas 3.0
andre 4.0
mariana 5.0
juliana 6.7
guilherme 7.0
carlos 8.5
paulo 9.0
```

Ele comparou o Paulo com a Lúcia e depois interrompeu o processo. O Paulo está no primeiro *array*, enquanto a Lúcia está no *segundo*. Ele comparou as duas notas e identifico que a do Paulo era a menor. Se ele escolheu o Paulo, ele aumentou +1 no `atualDoMauricio` . Com isto, acabaram os elementos do *array* do Mauricio e assim, ele interrompeu as comparações. Vamos ver se foi isto que realmente aconteceu? Iremos incluir um `System.out.println` no fim do código.

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

O resultado foi:

```
Estou comparando andre com jonas
Estou comparando andre com juliana
Estou comparando mariana com juliana
Estou comparando carlos com juliana
Estou comparando carlos com guilherme
Estou comparando carlos com lucia
Estou comparando paulo com lucia
Estou saindo
jonas 3.0
andre 4.0
mariana 5.0
juliana 6.7
guilherme 7.0
```

```
carlos 8.5
paulo 9.0
```

Por que ele saiu? Porque estas duas condições do `while` precisavam ser verdadeiras.

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

Vamos imprimir cada uma dela e conferir se as duas condições são verdadeiras? A primeira será:

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

Para a segunda faremos a mesma coisa para o Alberto:

```
System.out.println("Estou saindo");
System.out.println(atualDoMauricio.length);
System.out.println(atualDoAlberto.length);
```

Isto significa estamos perguntando: "sobrou alguém no Mauricio?" e "Sobrou alguém no Alberto?"

Vamos rodar e o resultado será:

```
Estou comparando andre com jonas
Estou comparando andre com juliana
Estou comparando mariana com juliana
Estou comparando carlos com juliana
Estou comparando carlos com guilherme
Estou comparando carlos com lucia
Estou comparando paulo com lucia
Estou saindo
false
true
jonas 3.0
andre 4.0
mariana 5.0
juliana 6.7
guilherme 7.0
carlos 8.5
paulo 9.0
```

O `false` indica que não sobrou ninguém no Maurício. Já o `true` nos diz que sobraram alunos no Alberto.

Qual foi o nosso problema? Qual foi o *bug*? Nós intercalamos todos os elementos, um por um. Quando chegamos no fim de um dos *array*s, ainda existia elementos no outro. Com esta condição pode ter sobrado diversos elementos em cada um dos grupos. Mas nós precisamos incluir estas pessoas, afinal se só restaram elementos do grupo do Maurício, vamos colocá-los no *array*. Se sobraram alunos do Alberto, vamos incluí-los lá também. Em breve, é isto o que iremos fazer.

Intercalando os elementos que sobraram

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
```

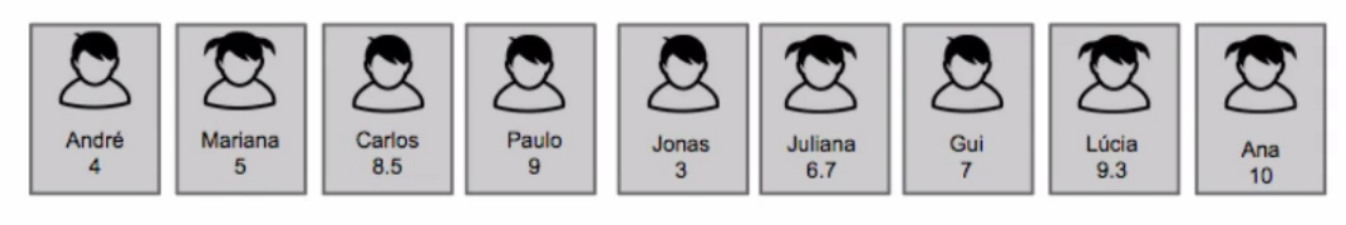
```
- lucia 9.3
- ana 10.0
```

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

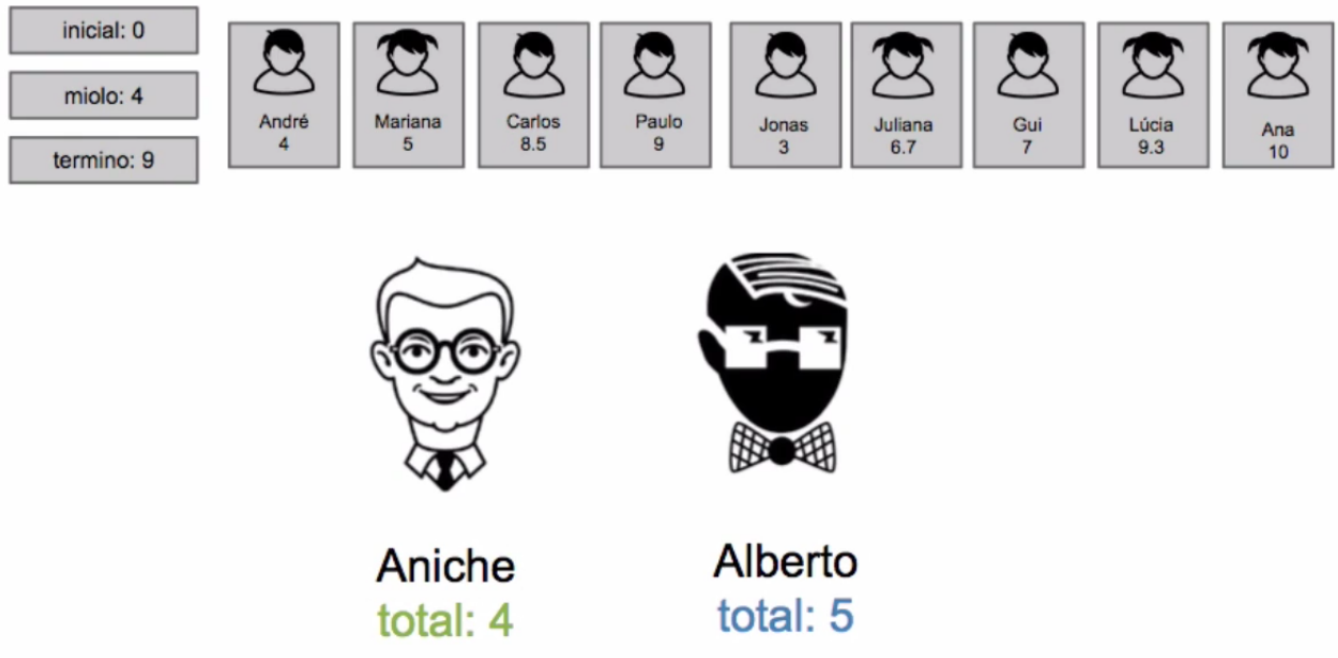
O problema de intercalar dados em um único *array*

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.



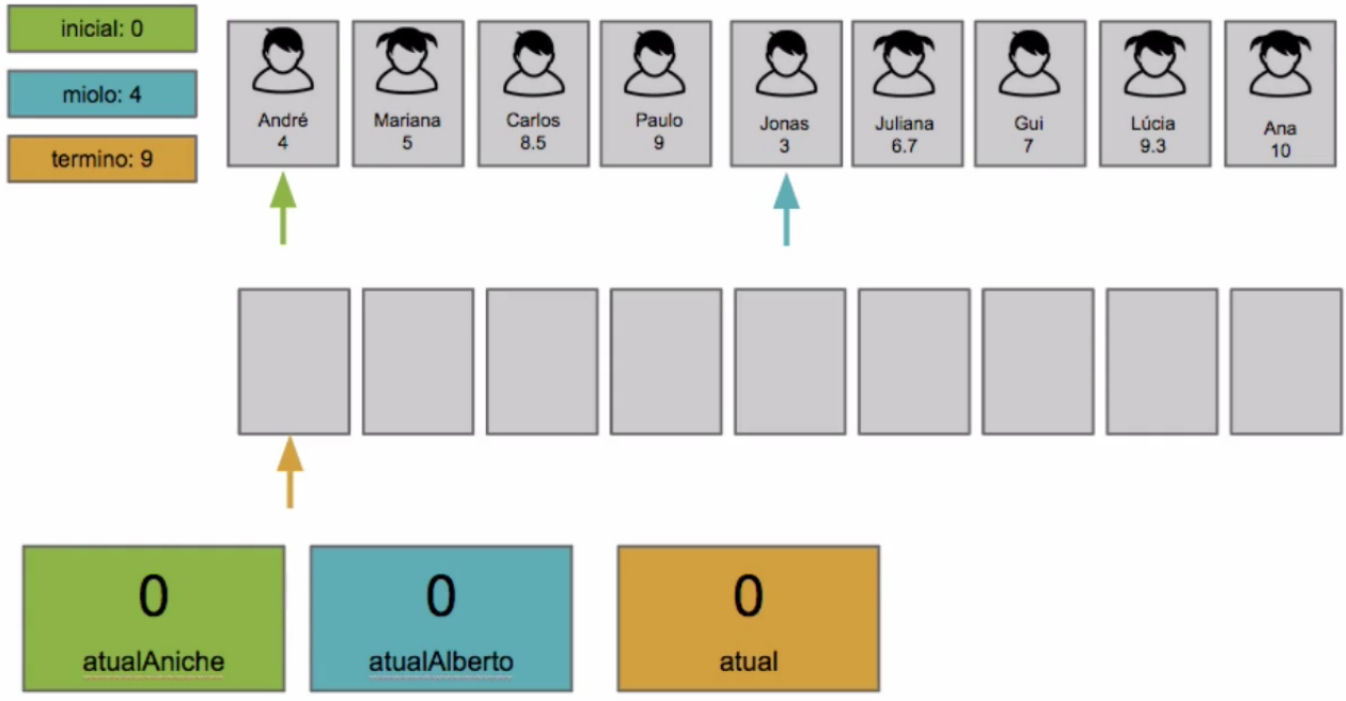
Logo, tudo o que fizemos até agora com dois *array*s, teremos que fazer algumas alterações no nosso código para trabalharmos com uma lista única.

Simulando a intercalação com um único array

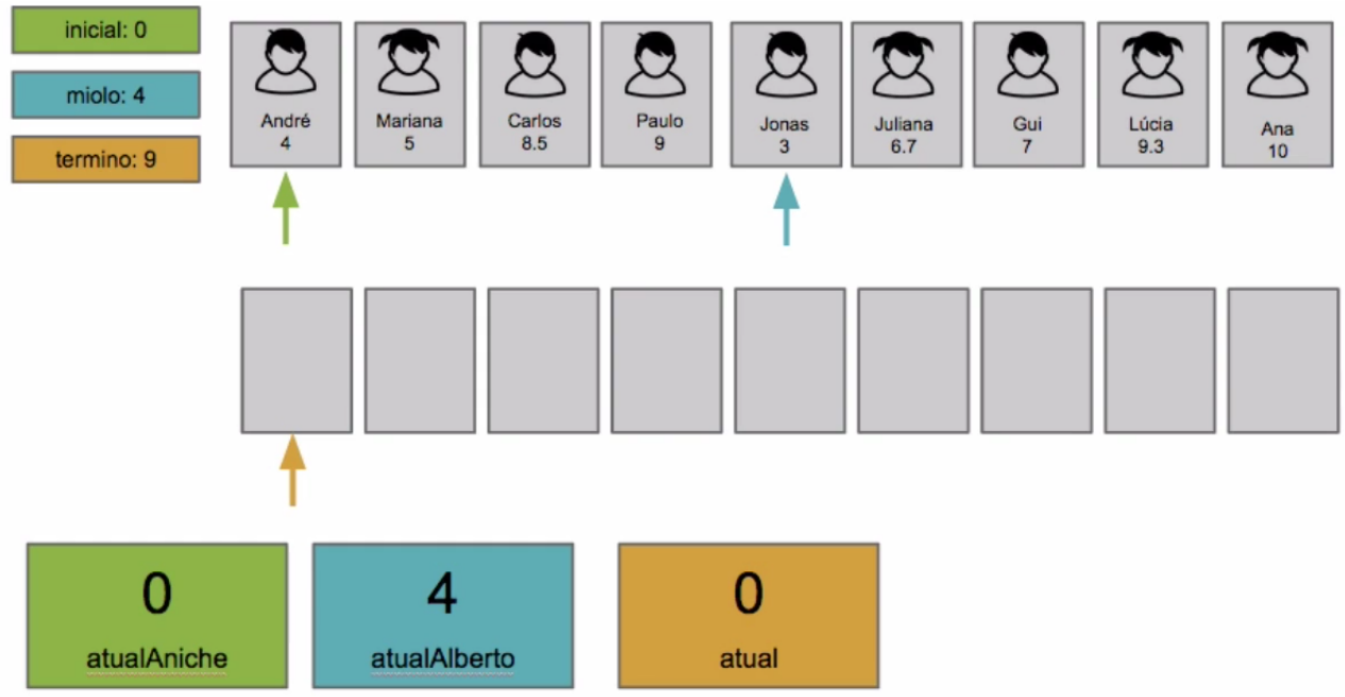
Voltando ao nosso algoritmo de intercalar dois *array*s, nós sabemos que recebemos um único *array*. O total de elementos se refere ao `termino`.

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.

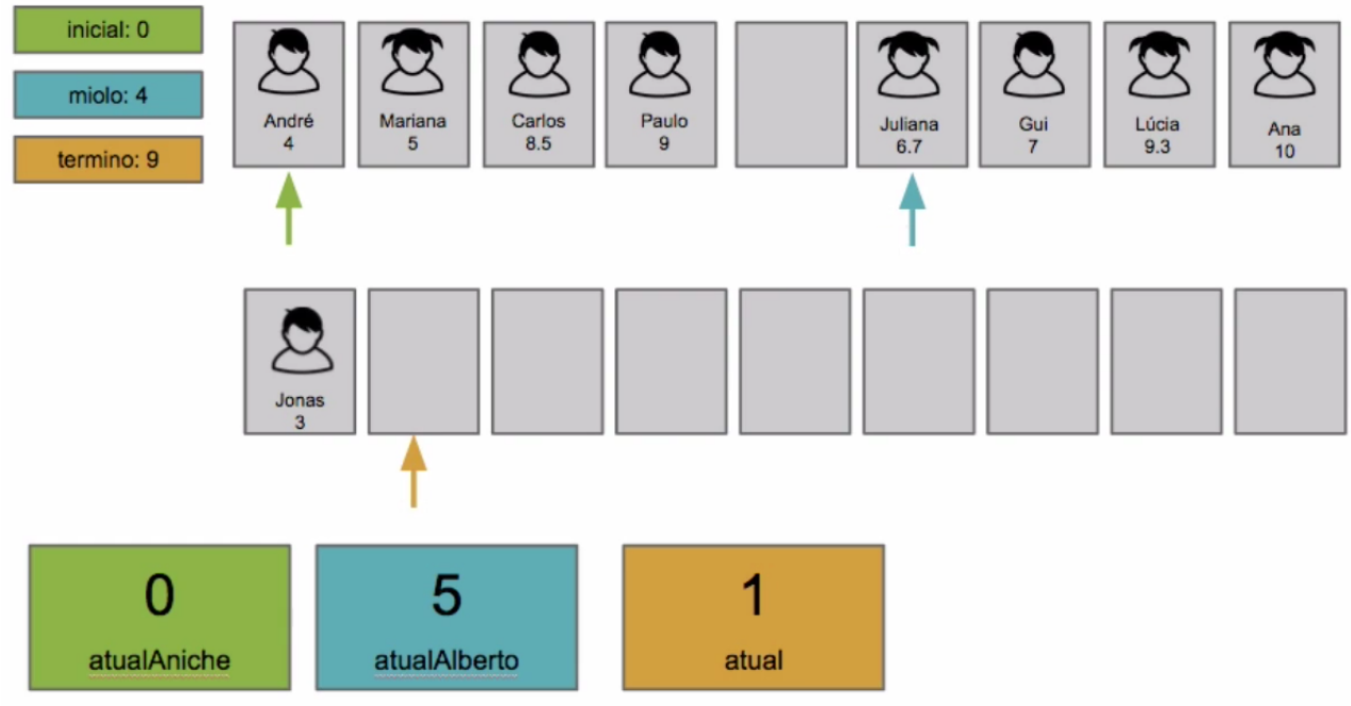


A melhor maneira para começarmos é pelo `atualAlberto`, que inicia no `miolo`. Isto significa que começaremos pelo valor 4. Vamos rodar o nosso algoritmo.

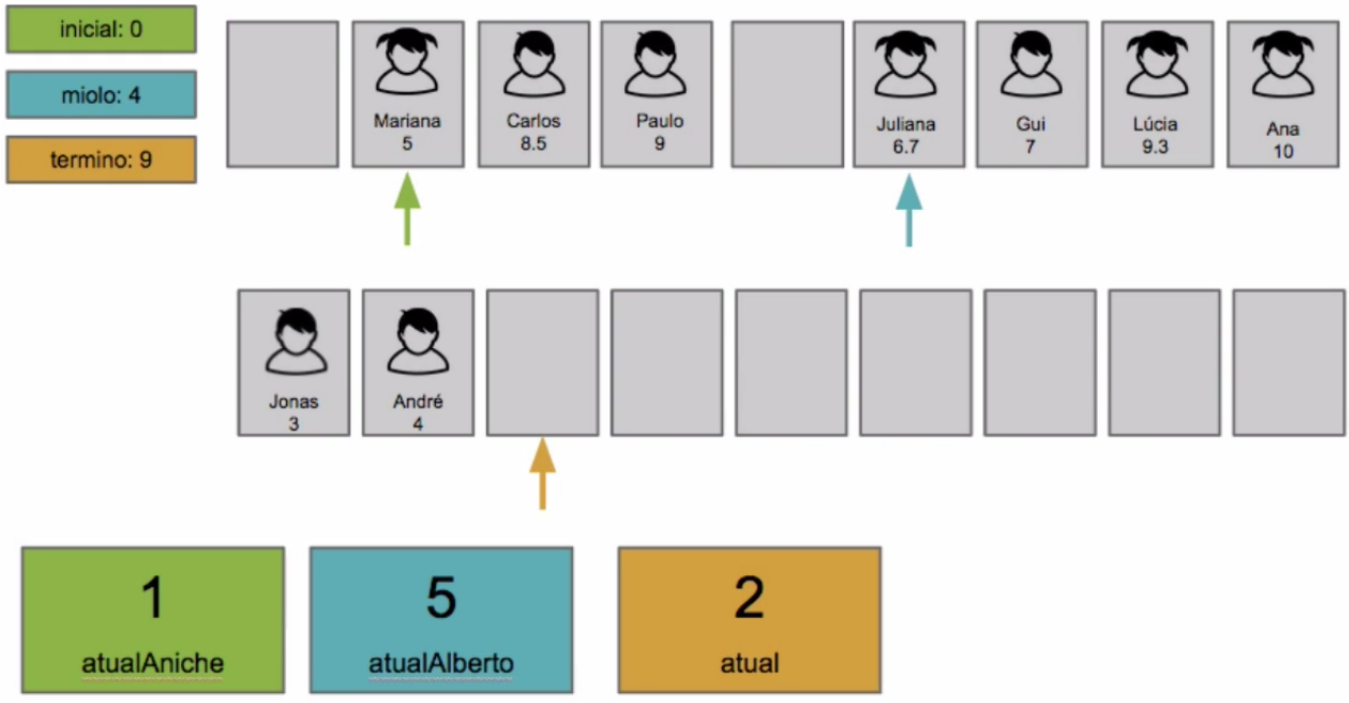


Iremos comparar as notas do André, que está no `atualAniche`, e do Jonas, que está no `atualAlberto`. Qual dos dois é o menor? Você irá perceber que é o mesmo algoritmo utilizado antes!

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.