
Anyway, lets start with the first question!


Merge sort!


```java
package myMergeSort;

import java.util.*;
```


```java
package myMergeSort;

import java.util.*;
```

```java
public class Sorter<T> {

   public List<T> sort(List<T> list){ ... }

}
```


```java
package myMergeSort;

import java.util.*;
```

```java
public class Sorter<T extends Comparable<T>>{

  public List<T> sort(List<T> list){ ... }

}
```


```java
package myMergeSort;

import java.util.*;
```

```java
public class Sorter<T extends Comparable<? super T>>{

  public List<T> sort(List<T> list){ ... }

}
```

```
public List<T> sort(List<T> list){
    ...
}
```


```
public List<T> sort(List<T> list){
    ...
}
```


```
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  . . . . . .
  . . . . . sort(. . . . . );
  . . . . . sort(. . . . . );
  return merge(. . . . .);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```


```
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  . . . . . .
  . . . . . sort(. . . . . );
  . . . . . sort(. . . . . );
  return merge(. . . . . .);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```

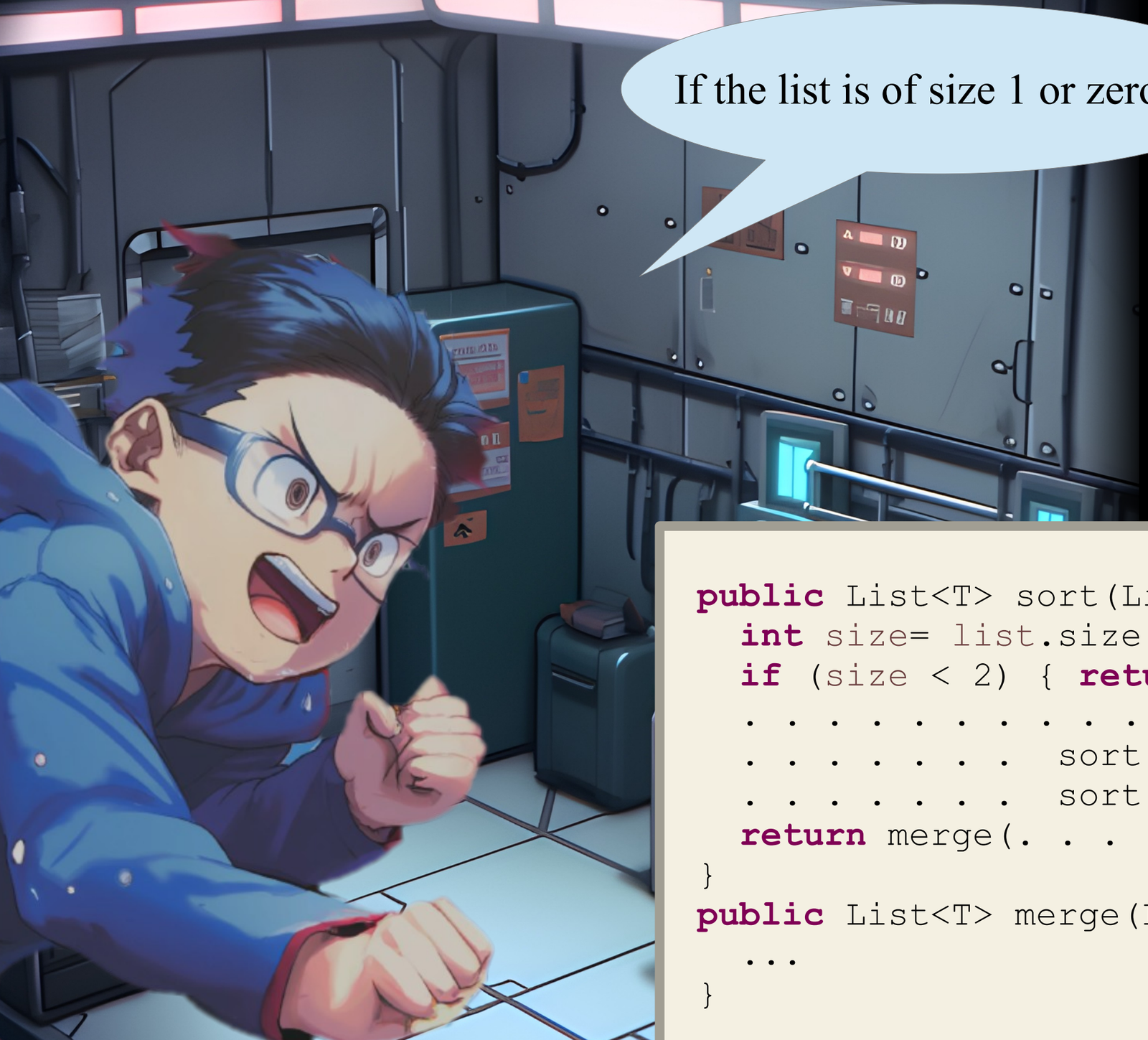

```
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  . . . . . .
  . . . . . sort(. . . . . );
  . . . . . sort(. . . . . );
  return merge(. . . . .);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```

Now, for the body of the sort method itself!

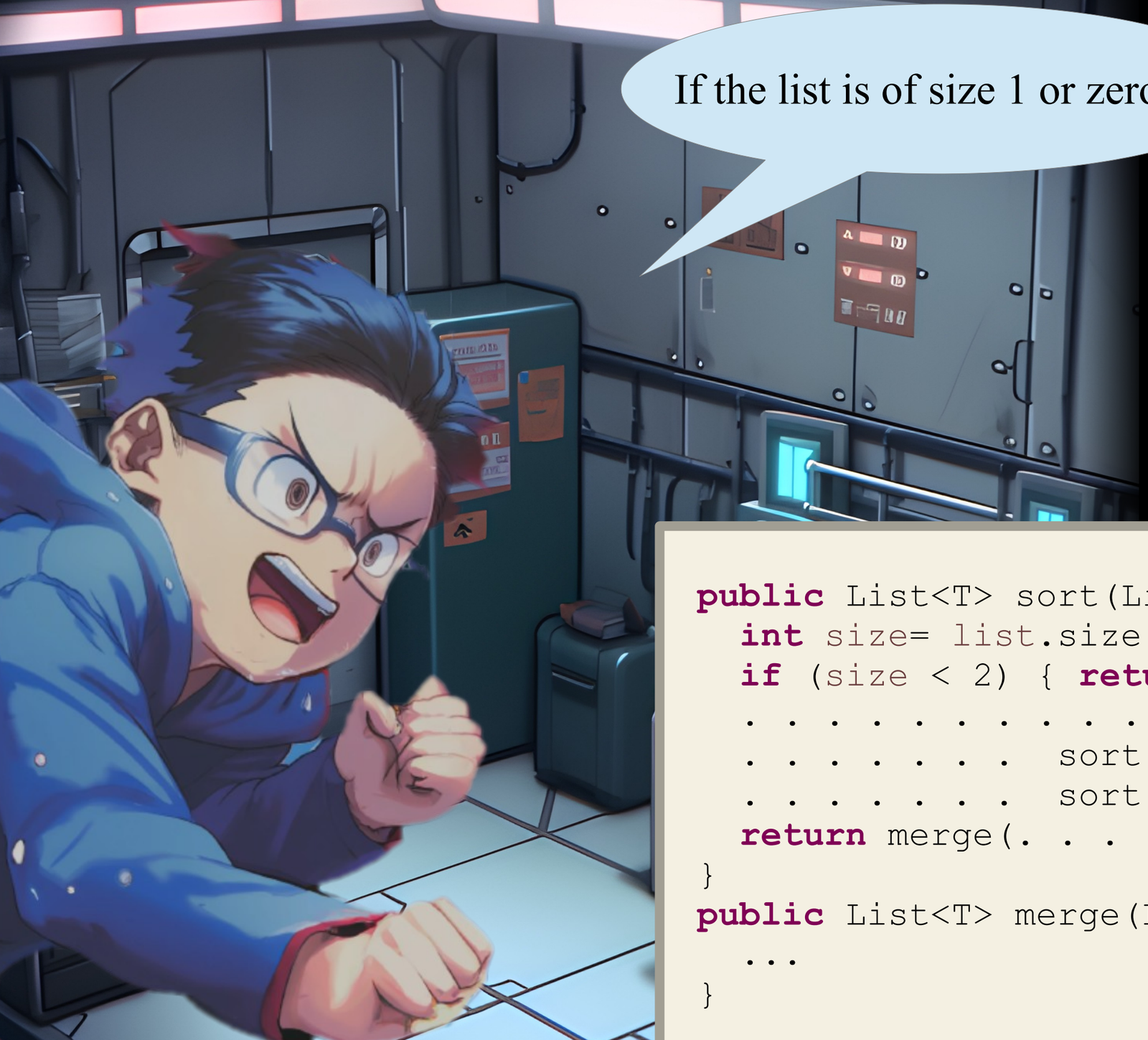
If I remember well, we have two most important methods: The recursive method ‘sort’ and the auxiliary method ‘merge’

```
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  . . . . . .
  . . . . . sort(. . . . . );
  . . . . . sort(. . . . . );
  return merge(. . . . . .);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```

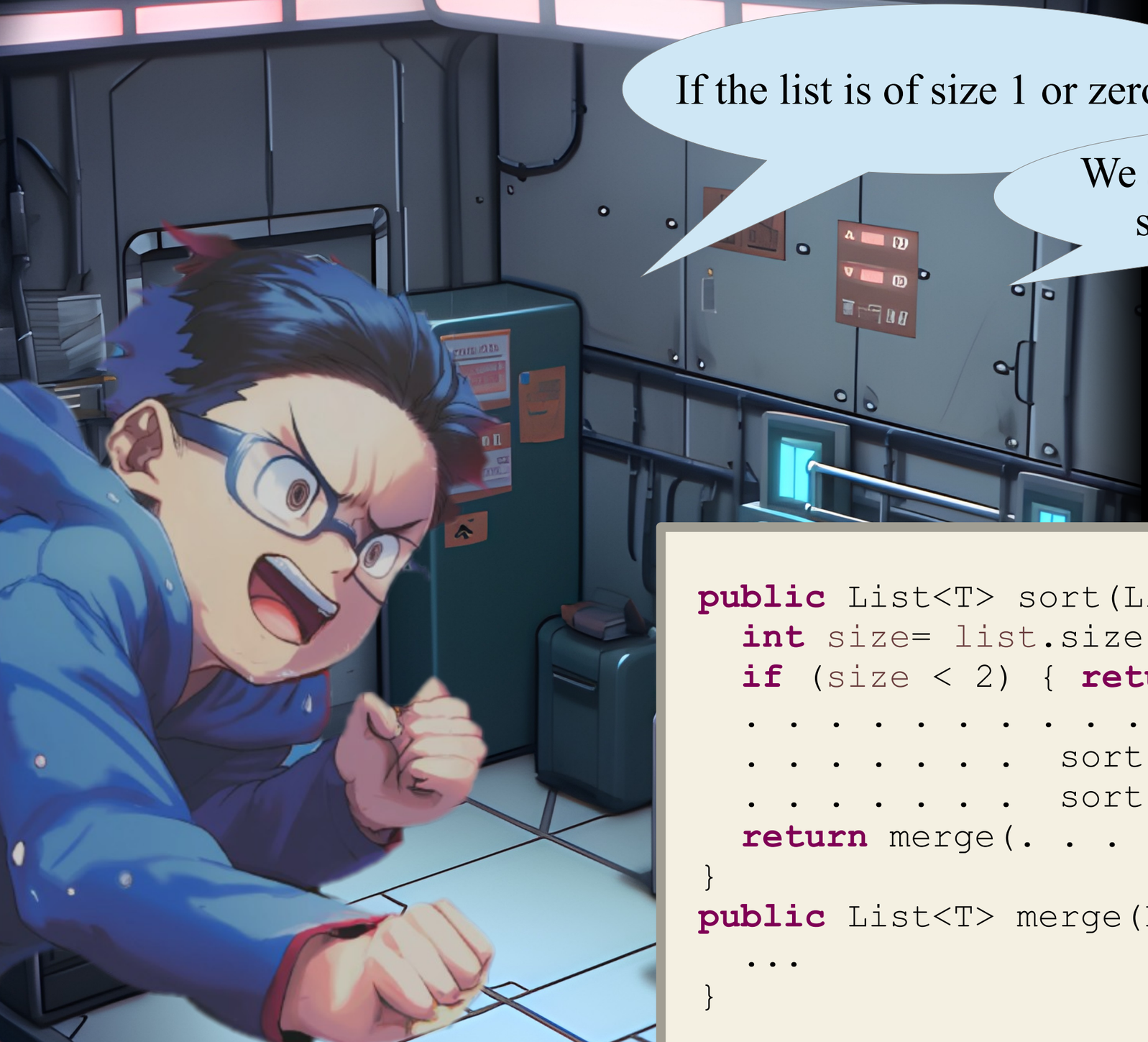
```
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  . . . . . .
  . . . . . sort(. . . . . );
  . . . . . sort(. . . . . );
  return merge(. . . . . .);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```




```
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  . . . . . .
  . . . . . .  sort(. . . . . .);
  . . . . . .  sort(. . . . . .);
  return merge(. . . . . .);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```




```
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  . . . . . .
  . . . . . .  sort(. . . . . .);
  . . . . . .  sort(. . . . . .);
  return merge(. . . . . .);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```



If the list is of size 1 or zero ...

We are done, we return the list, since it is already sorted.

```
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  . . . . . .
  . . . . . sort(. . . . . );
  . . . . . sort(. . . . . );
  return merge(. . . . . .);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```


```
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  . . . . . .
  . . . . . sort(. . . . . );
  . . . . . sort(. . . . . );
  return merge(. . . . . .);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```

```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(/* left half of list*/);
  List<T> right= sort(/*right half of list*/);
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```


```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(/* left half of list*/);
  List<T> right= sort(/*right half of list*/);
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```


```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(/* left half of list*/);
  List<T> right= sort(/*right half of list*/);
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```


```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(list.subList(0, half));
  List<T> right= sort(list.subList(half, size));
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```


```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(list.subList(0, half));
  List<T> right= sort(list.subList(half, size));
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```

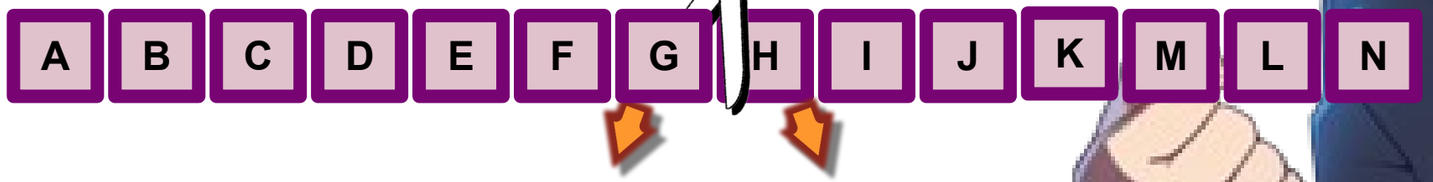
```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(list.subList(0, half));
  List<T> right= sort(list.subList(half, size));
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```

```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(list.subList(0, half));
  List<T> right= sort(list.subList(half, size));
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```

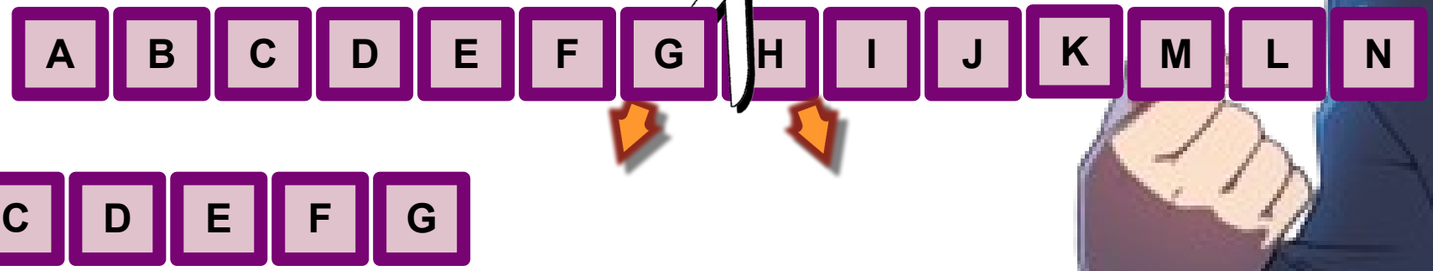
A	B	C	D	E	F	G	H	I	J	K	M	L	N

```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(list.subList(0, half));
  List<T> right= sort(list.subList(half, size));
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```

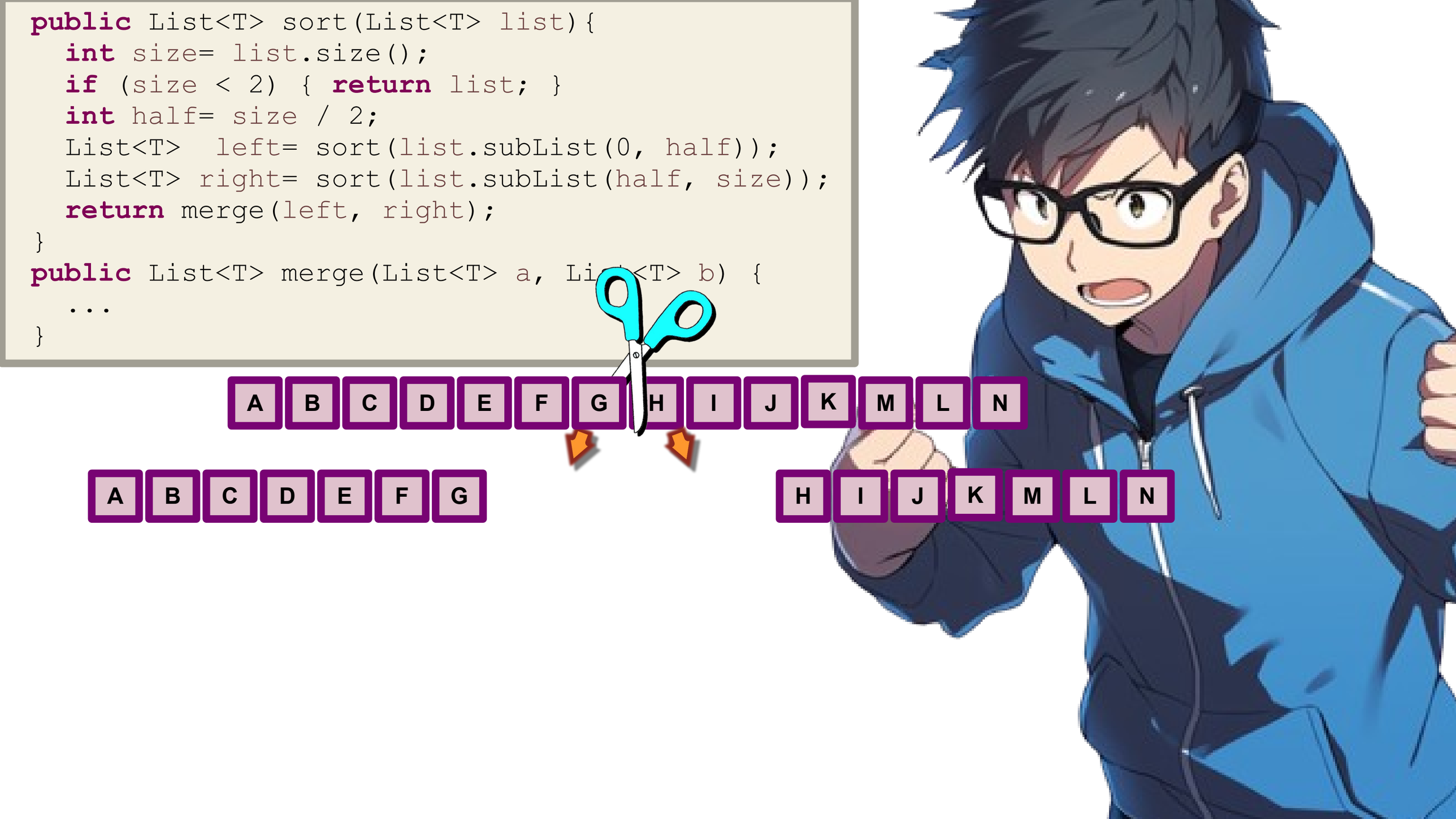
```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(list.subList(0, half));
  List<T> right= sort(list.subList(half, size));
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```



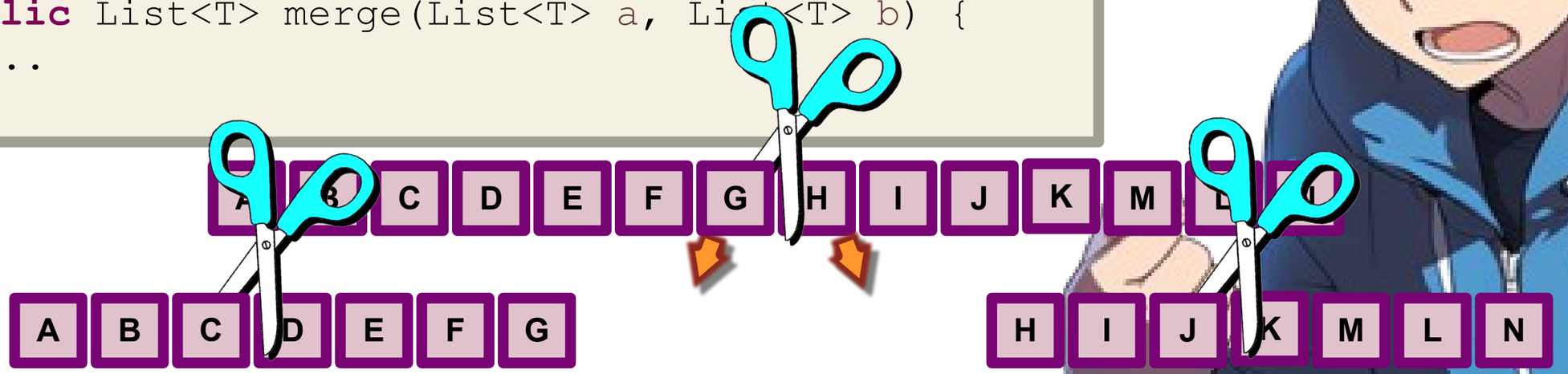
```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(list.subList(0, half));
  List<T> right= sort(list.subList(half, size));
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```



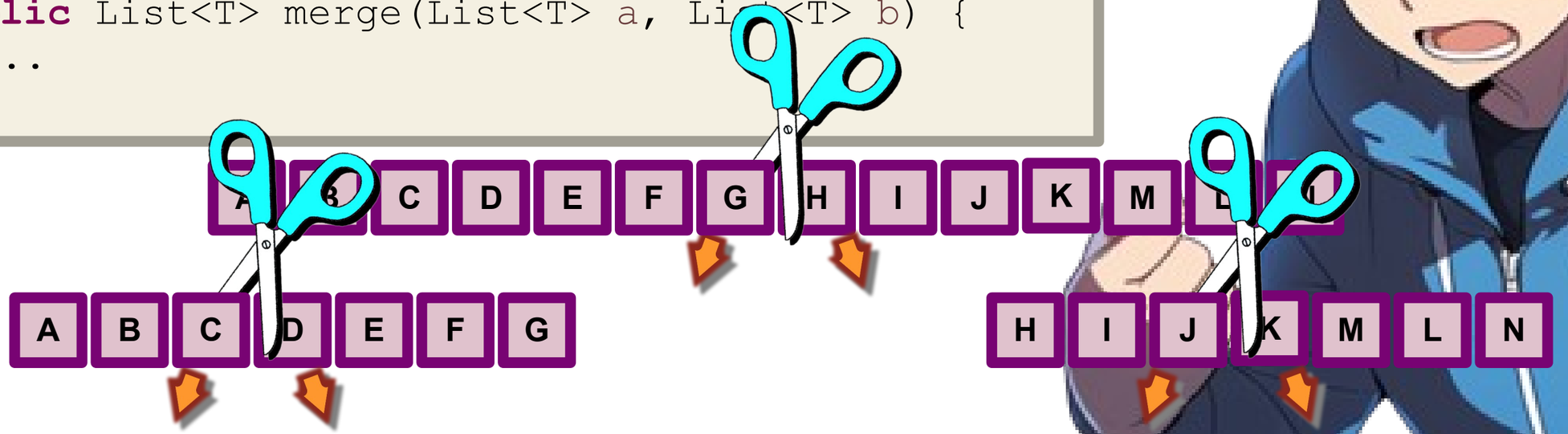
```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(list.subList(0, half));
  List<T> right= sort(list.subList(half, size));
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```



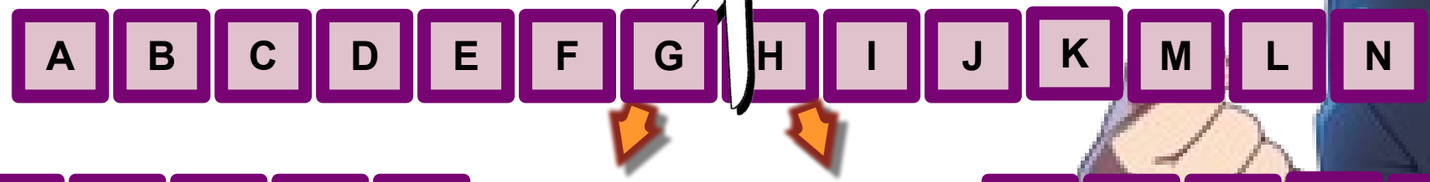
```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(list.subList(0, half));
  List<T> right= sort(list.subList(half, size));
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```

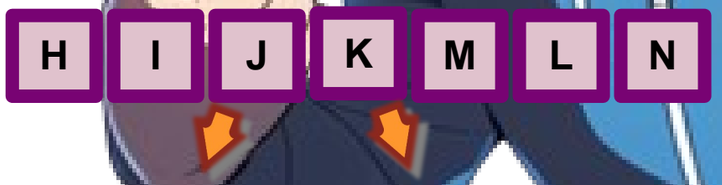


```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(list.subList(0, half));
  List<T> right= sort(list.subList(half, size));
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```

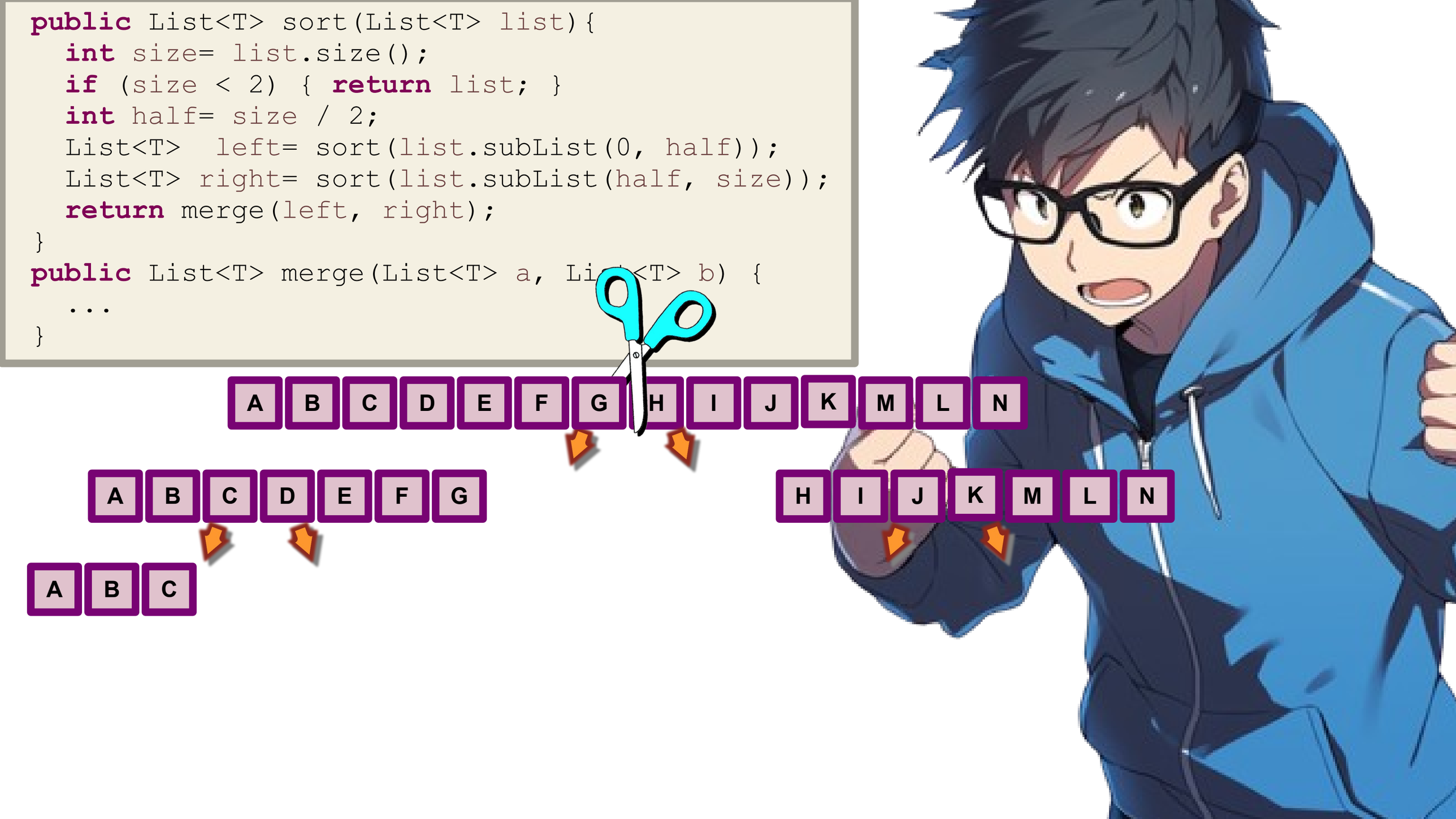


```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(list.subList(0, half));
  List<T> right= sort(list.subList(half, size));
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```

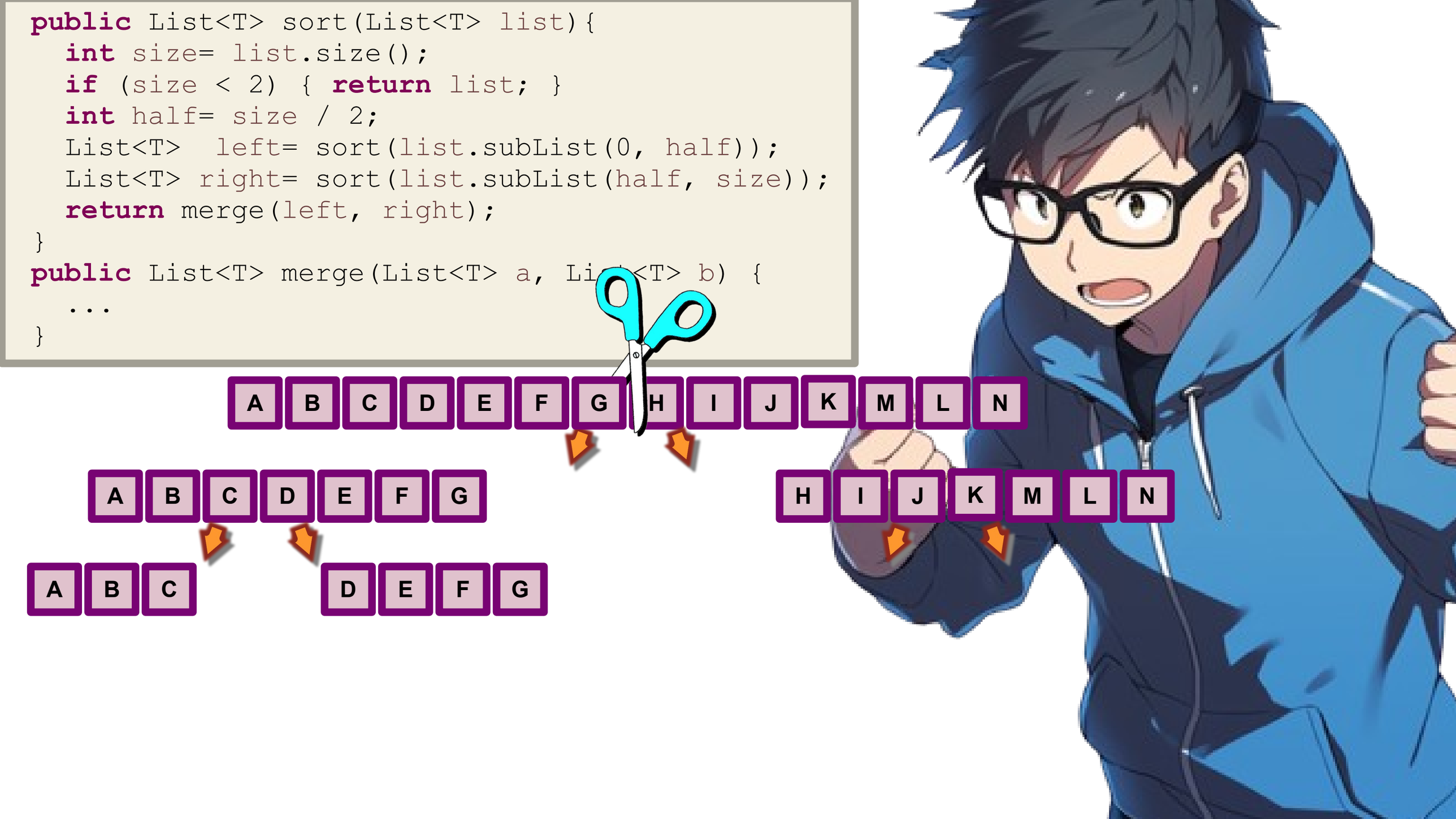




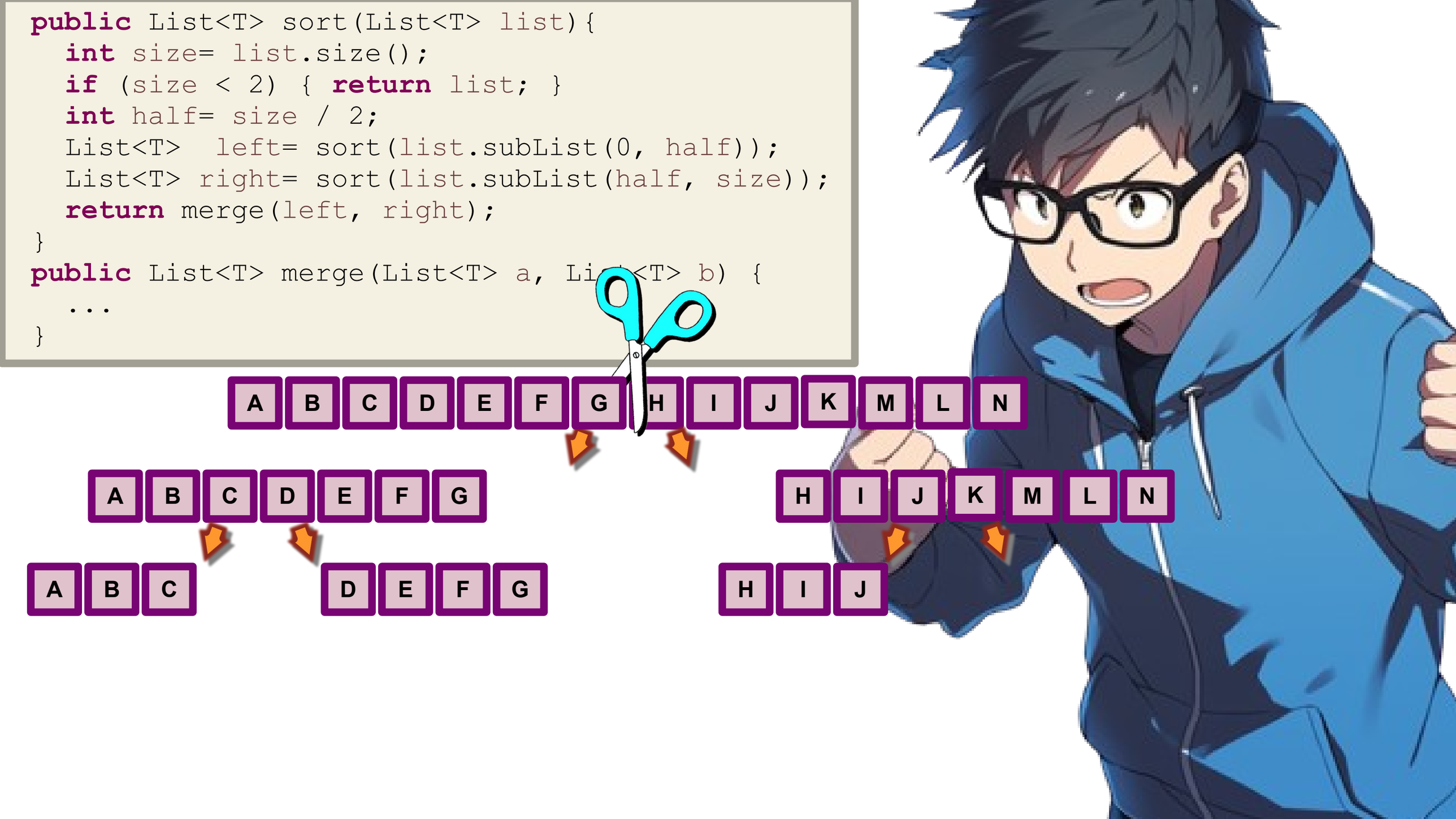
```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(list.subList(0, half));
  List<T> right= sort(list.subList(half, size));
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```



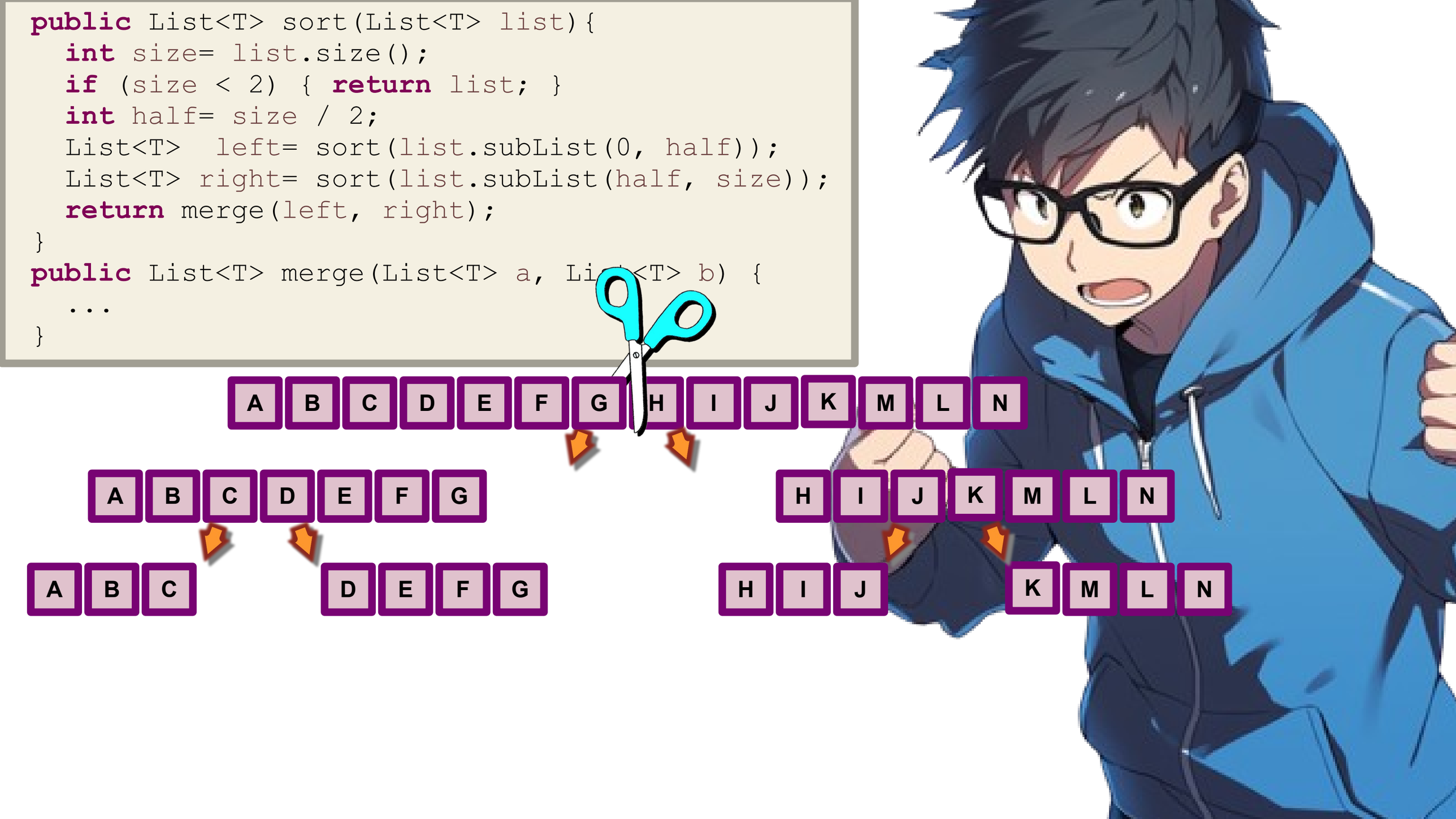
```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(list.subList(0, half));
  List<T> right= sort(list.subList(half, size));
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```



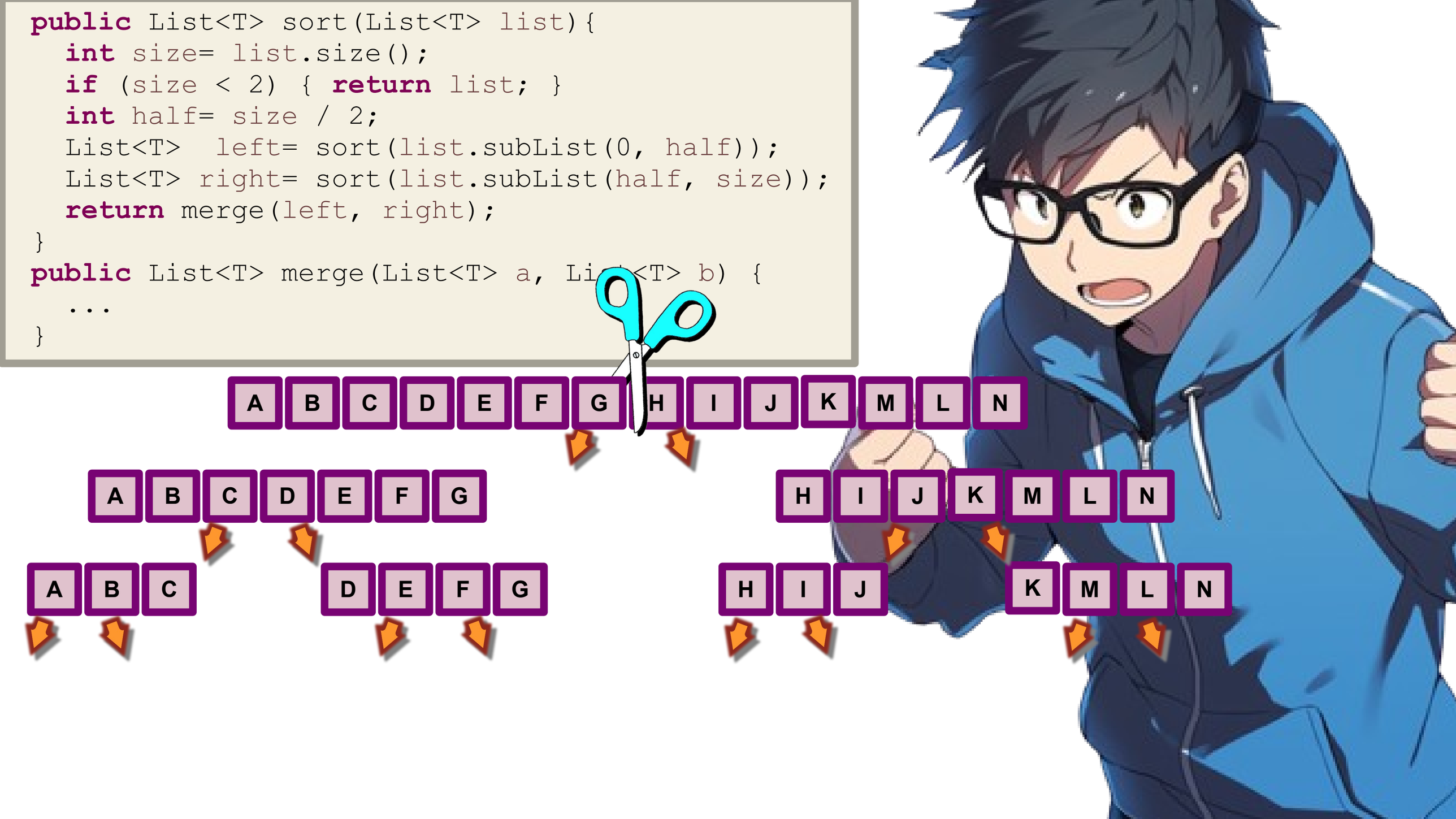
```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(list.subList(0, half));
  List<T> right= sort(list.subList(half, size));
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```



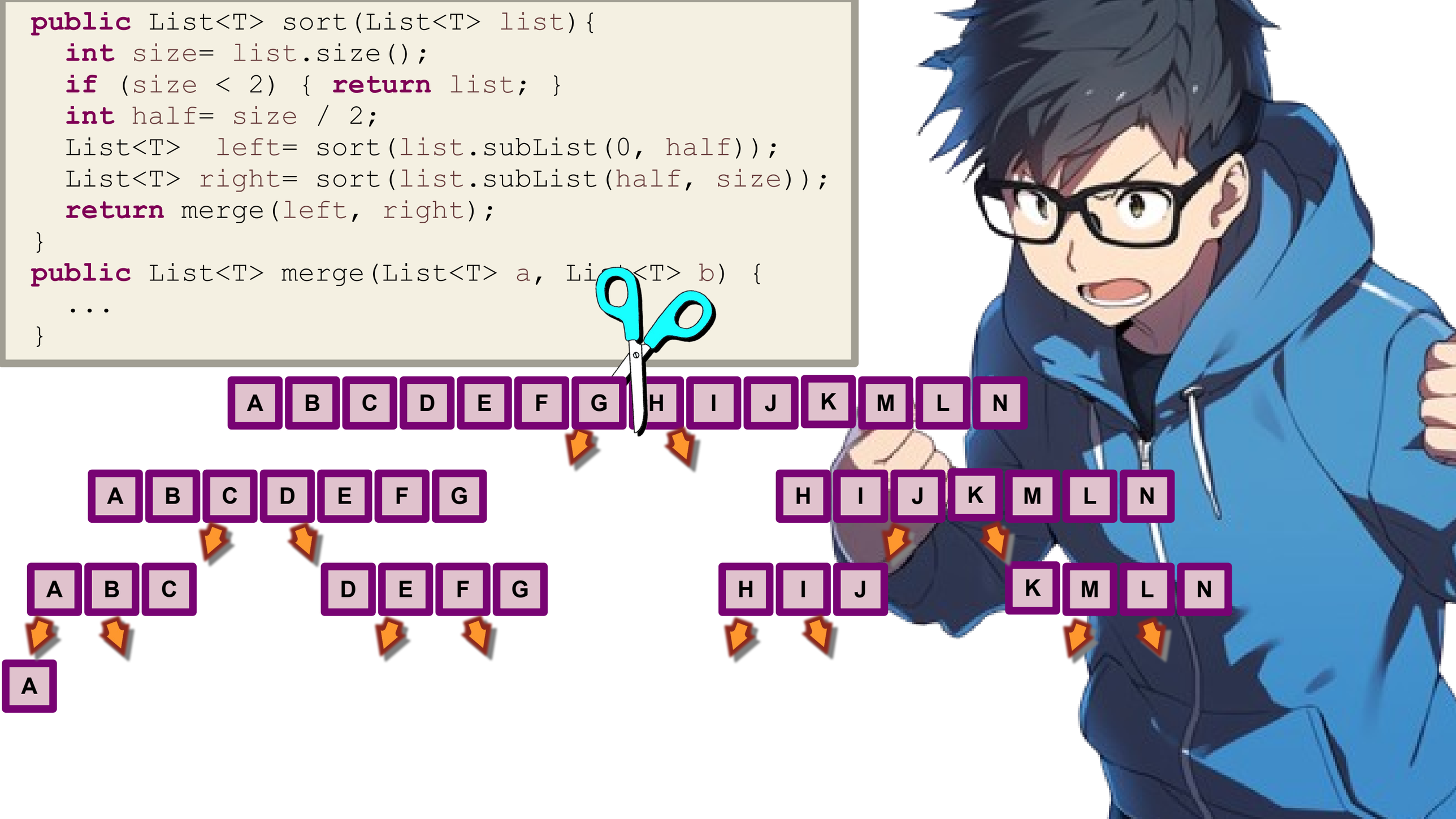
```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(list.subList(0, half));
  List<T> right= sort(list.subList(half, size));
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```



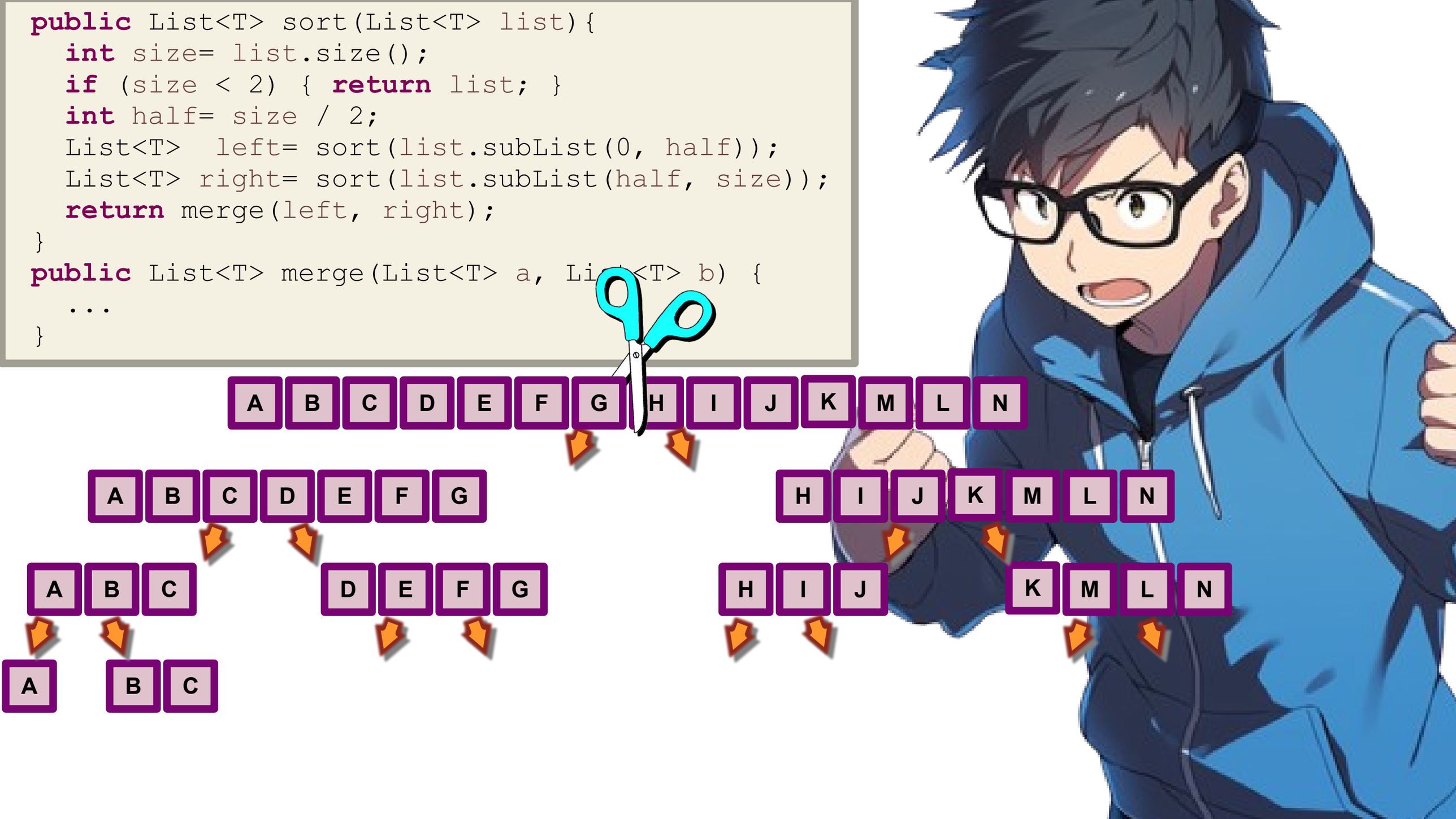
```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(list.subList(0, half));
  List<T> right= sort(list.subList(half, size));
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```



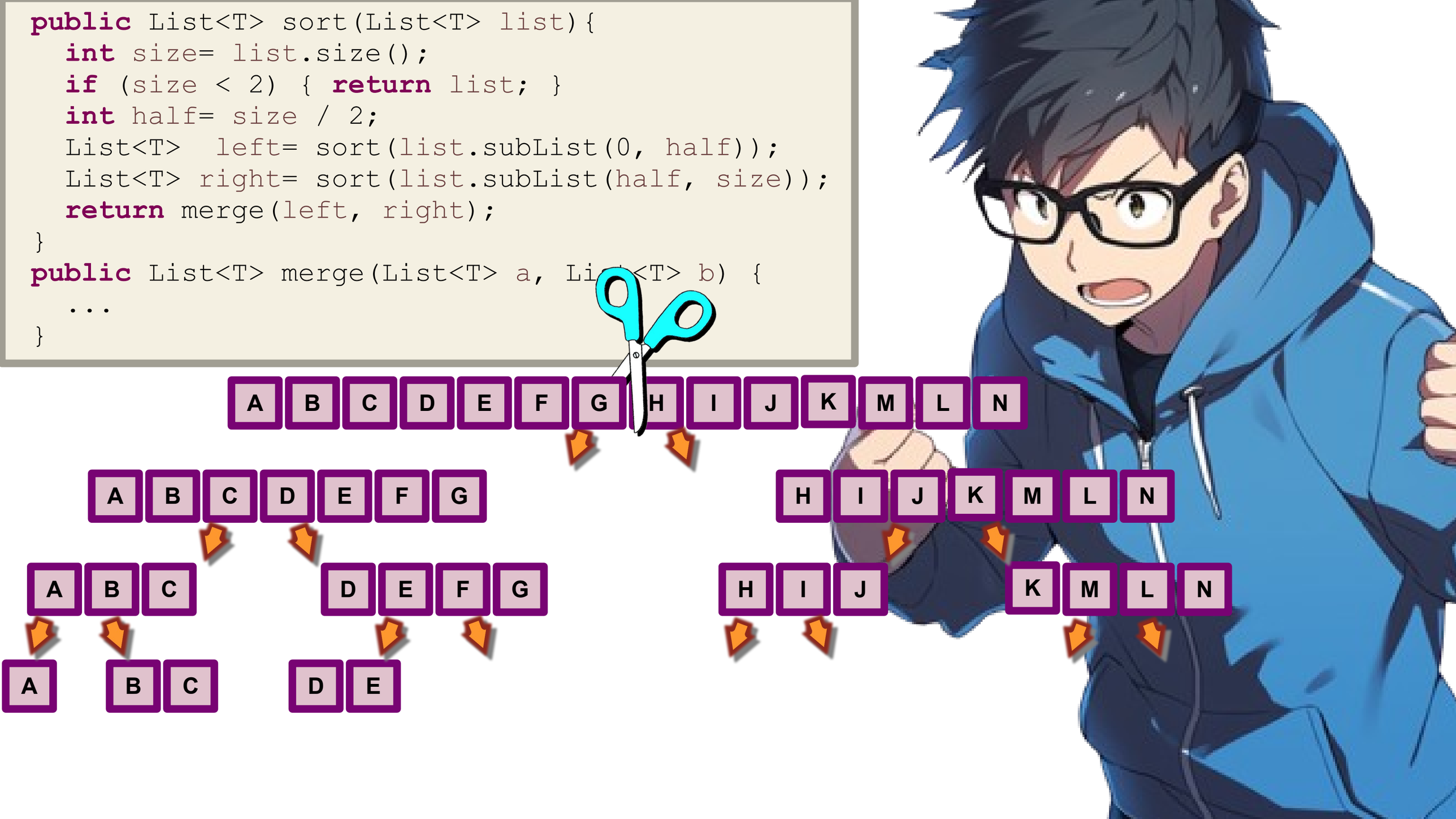
```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(list.subList(0, half));
  List<T> right= sort(list.subList(half, size));
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```



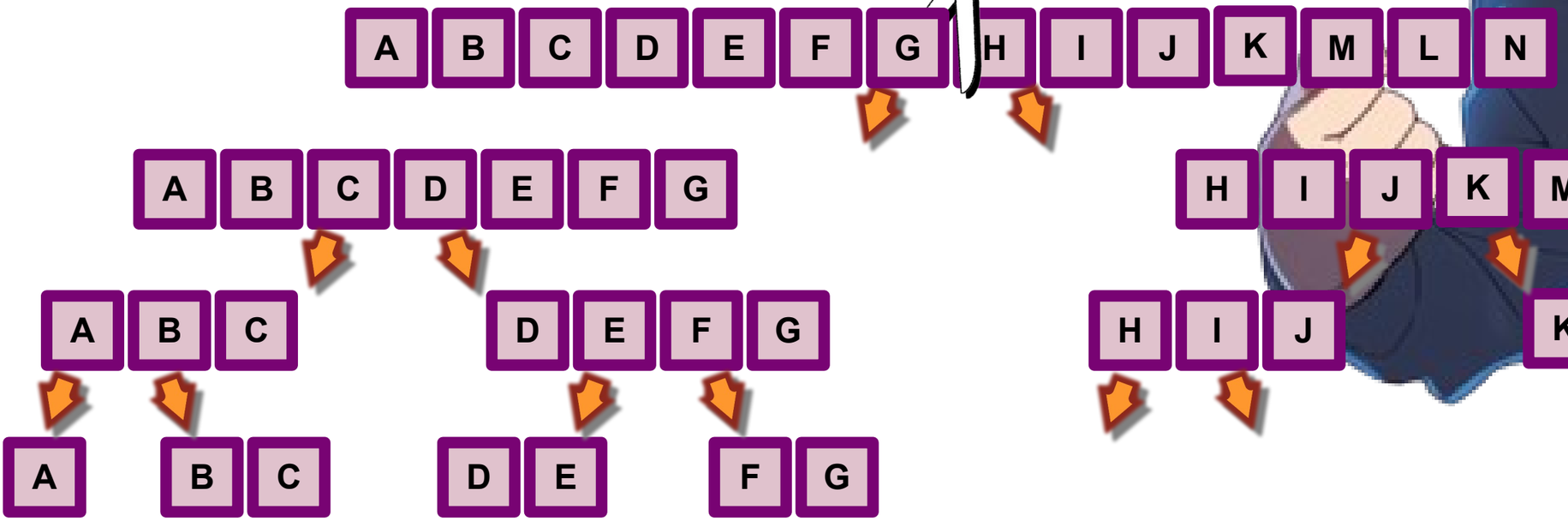
```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(list.subList(0, half));
  List<T> right= sort(list.subList(half, size));
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```

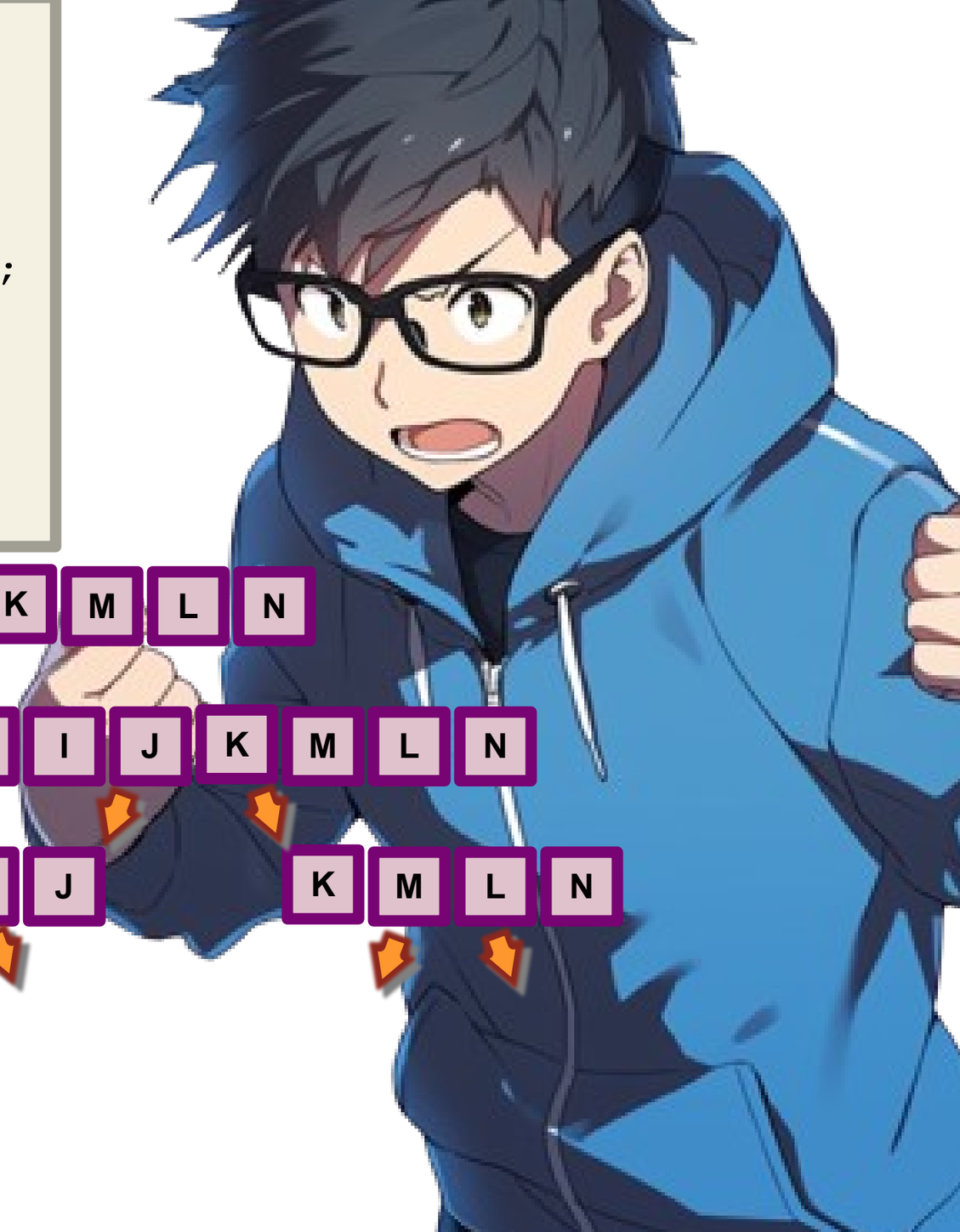


```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(list.subList(0, half));
  List<T> right= sort(list.subList(half, size));
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```

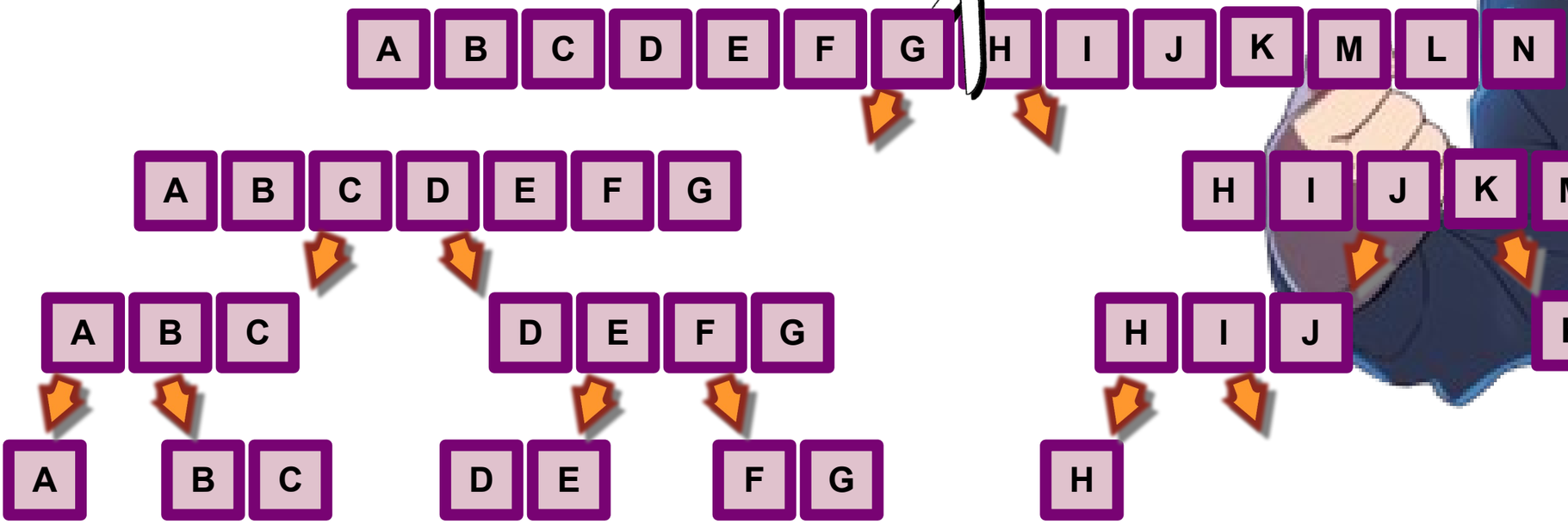


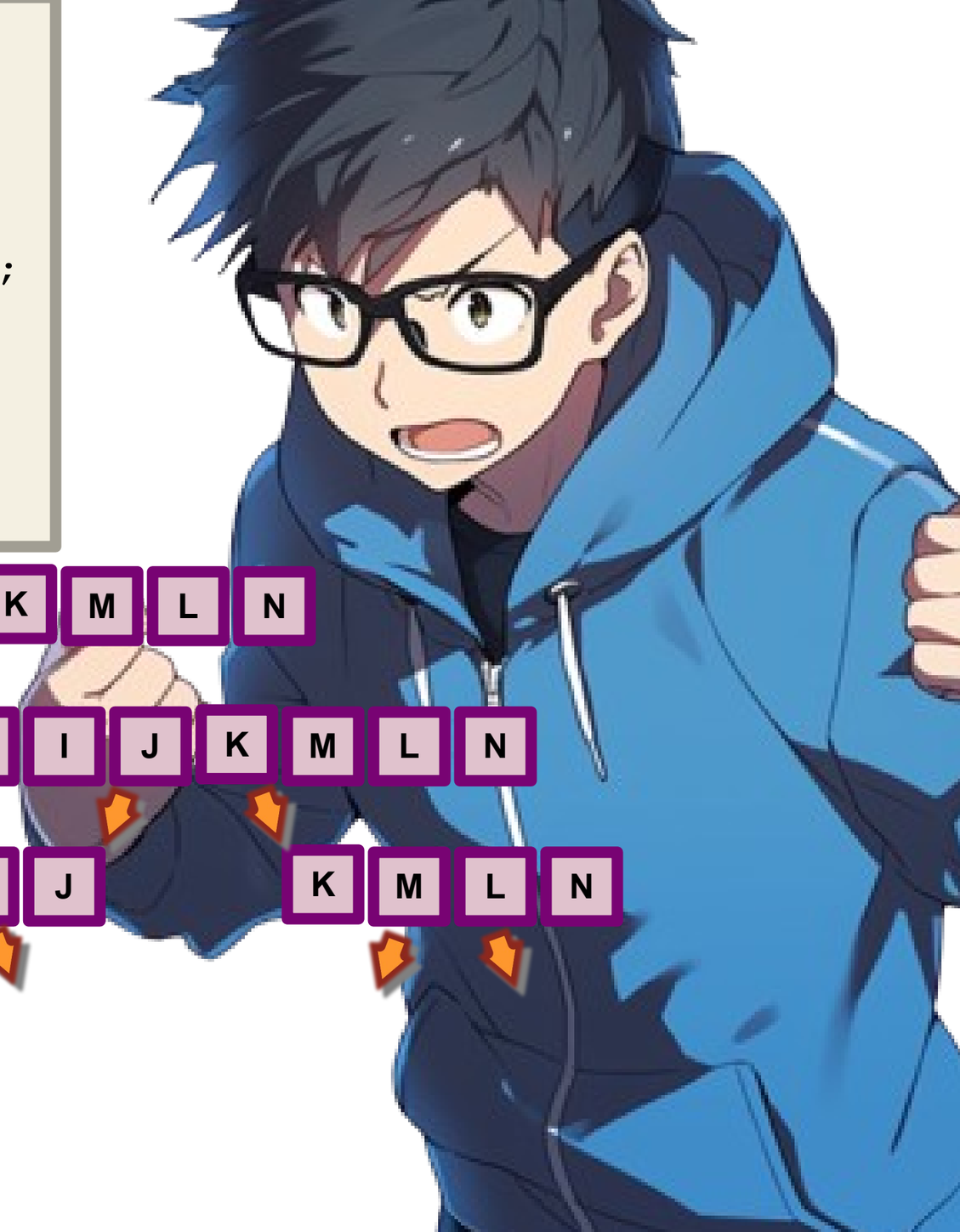
```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(list.subList(0, half));
  List<T> right= sort(list.subList(half, size));
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```



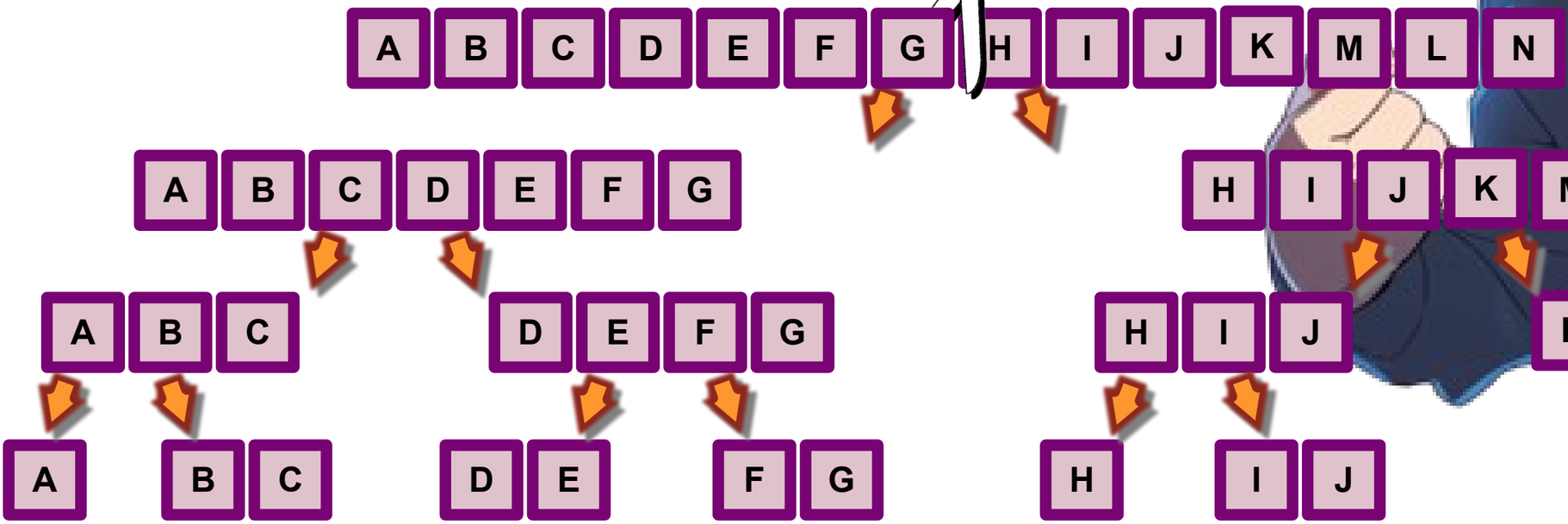


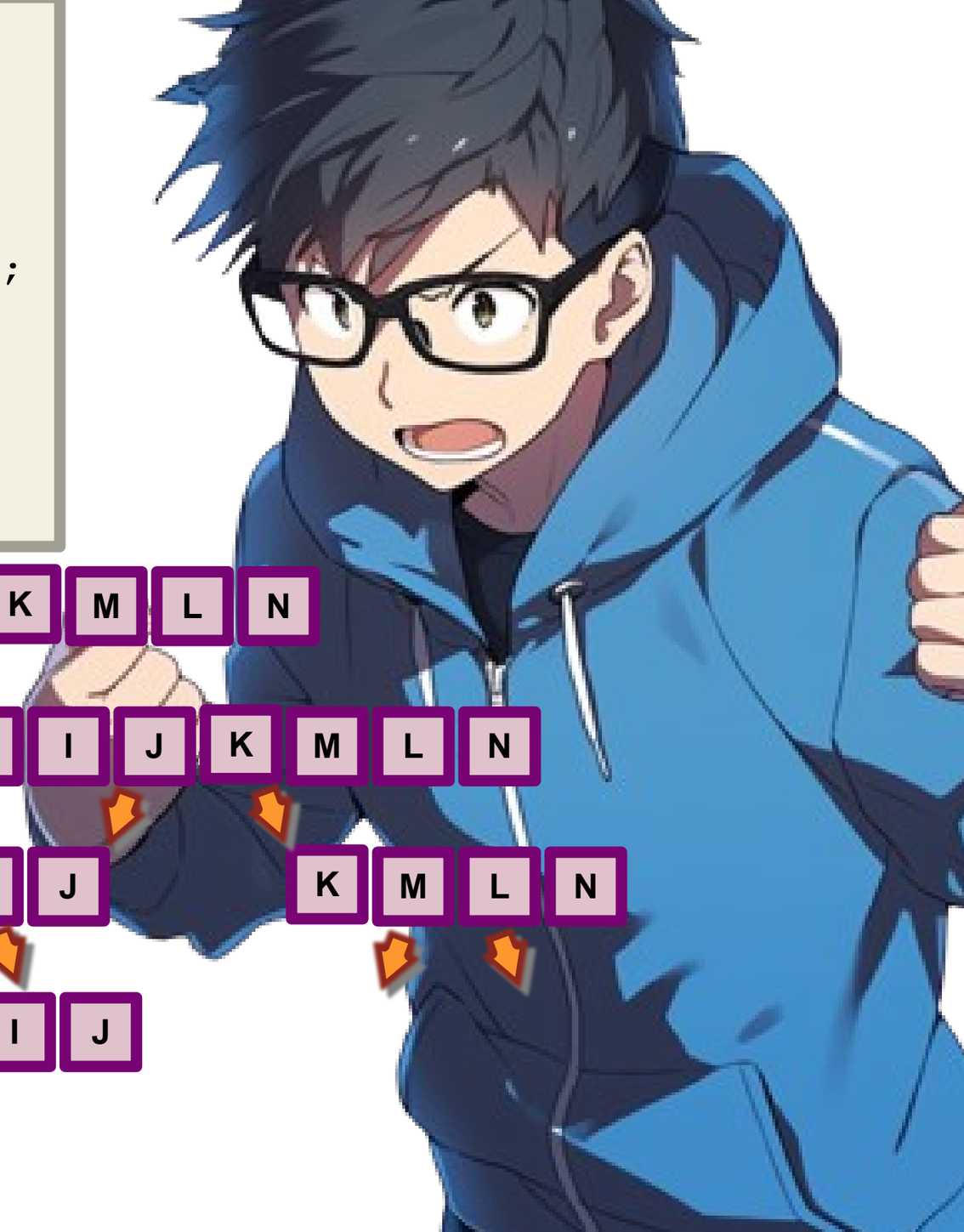
```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(list.subList(0, half));
  List<T> right= sort(list.subList(half, size));
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```



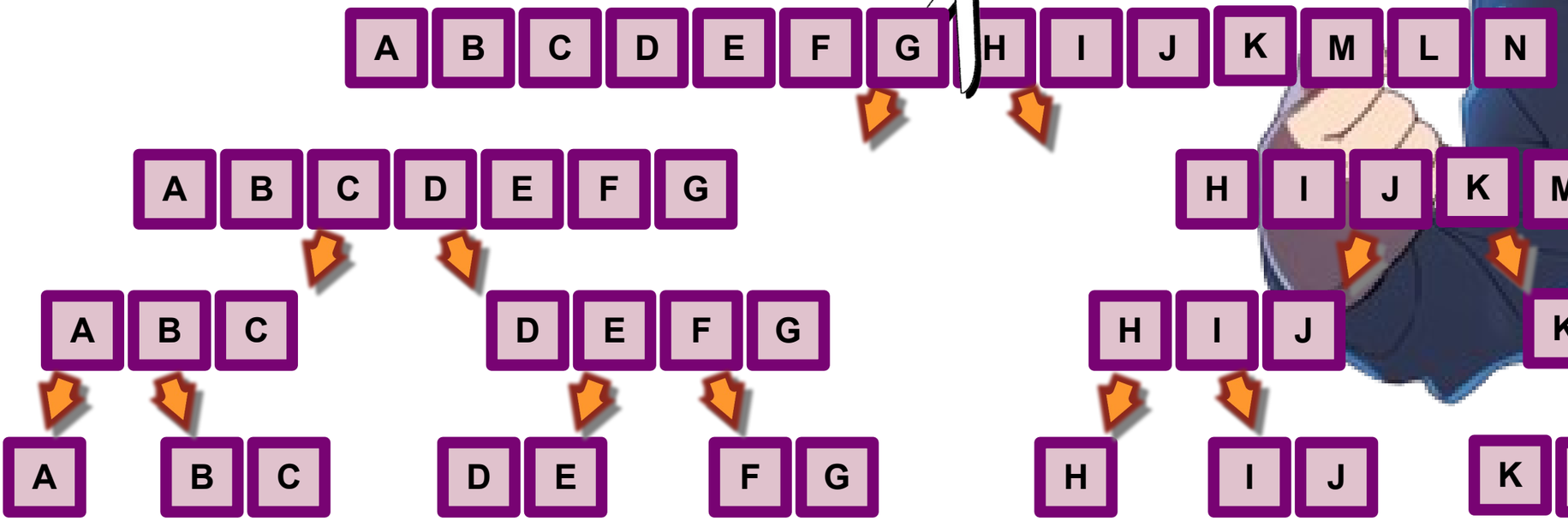


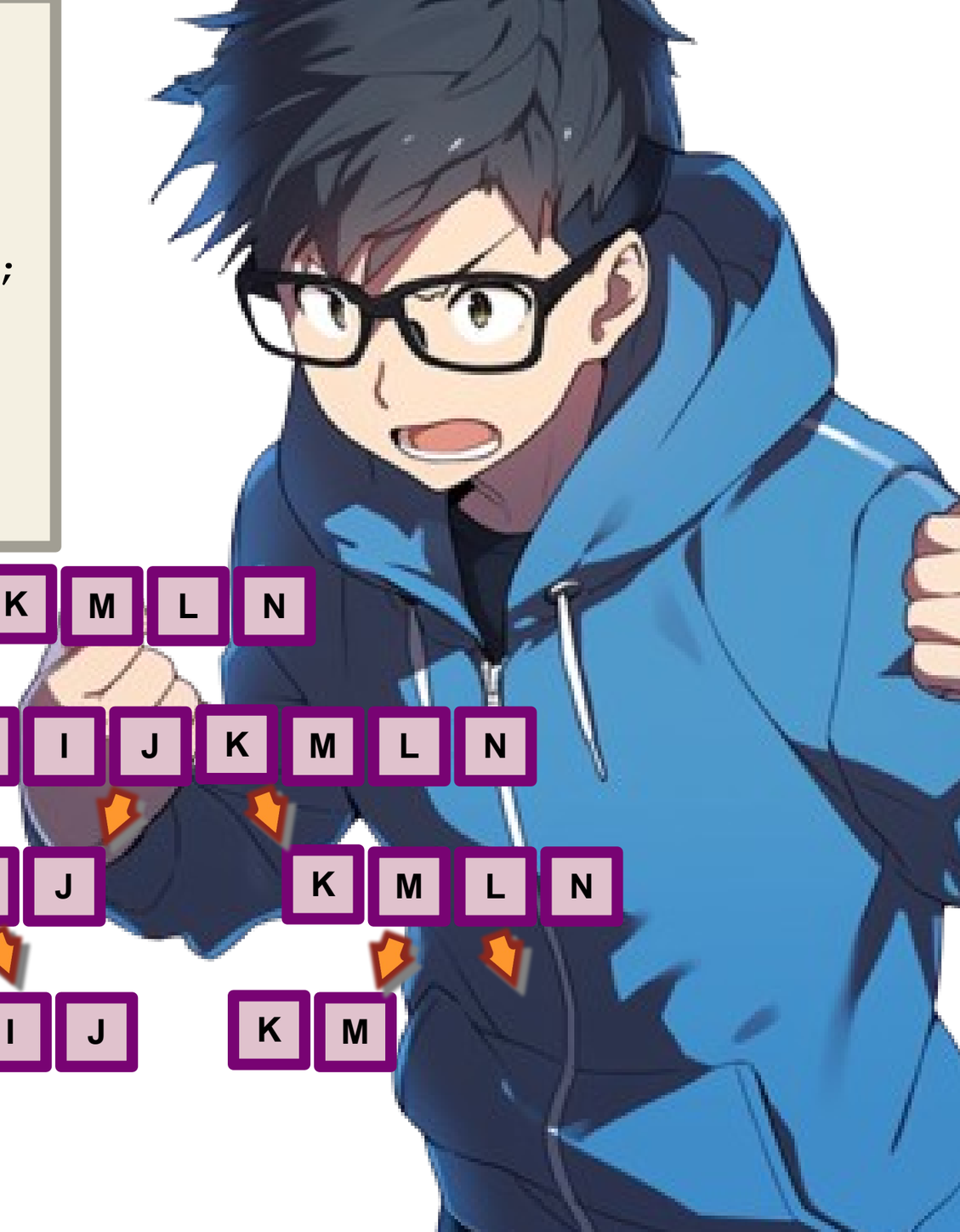
```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(list.subList(0, half));
  List<T> right= sort(list.subList(half, size));
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```



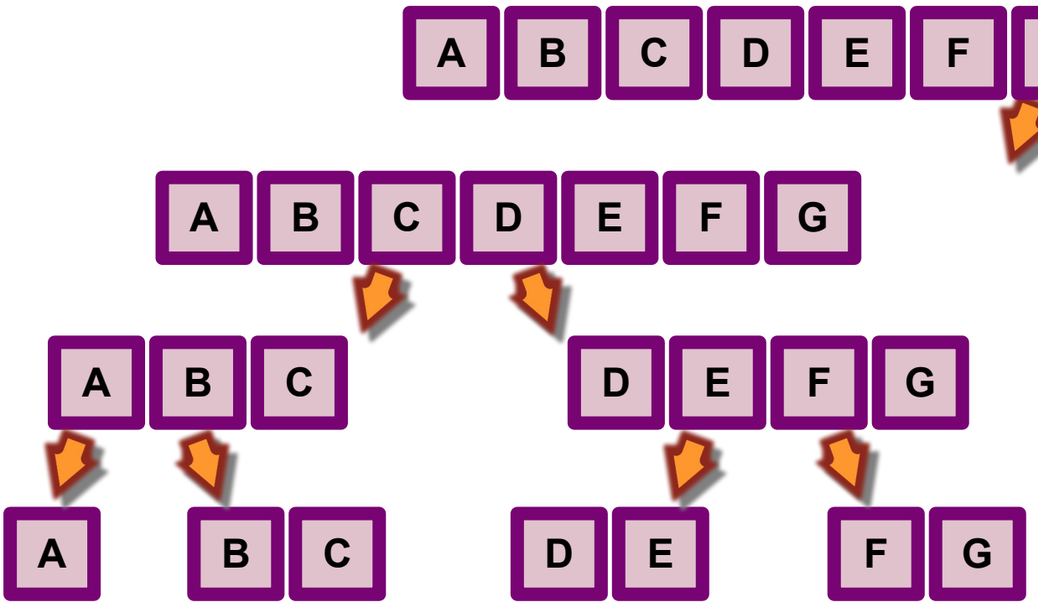


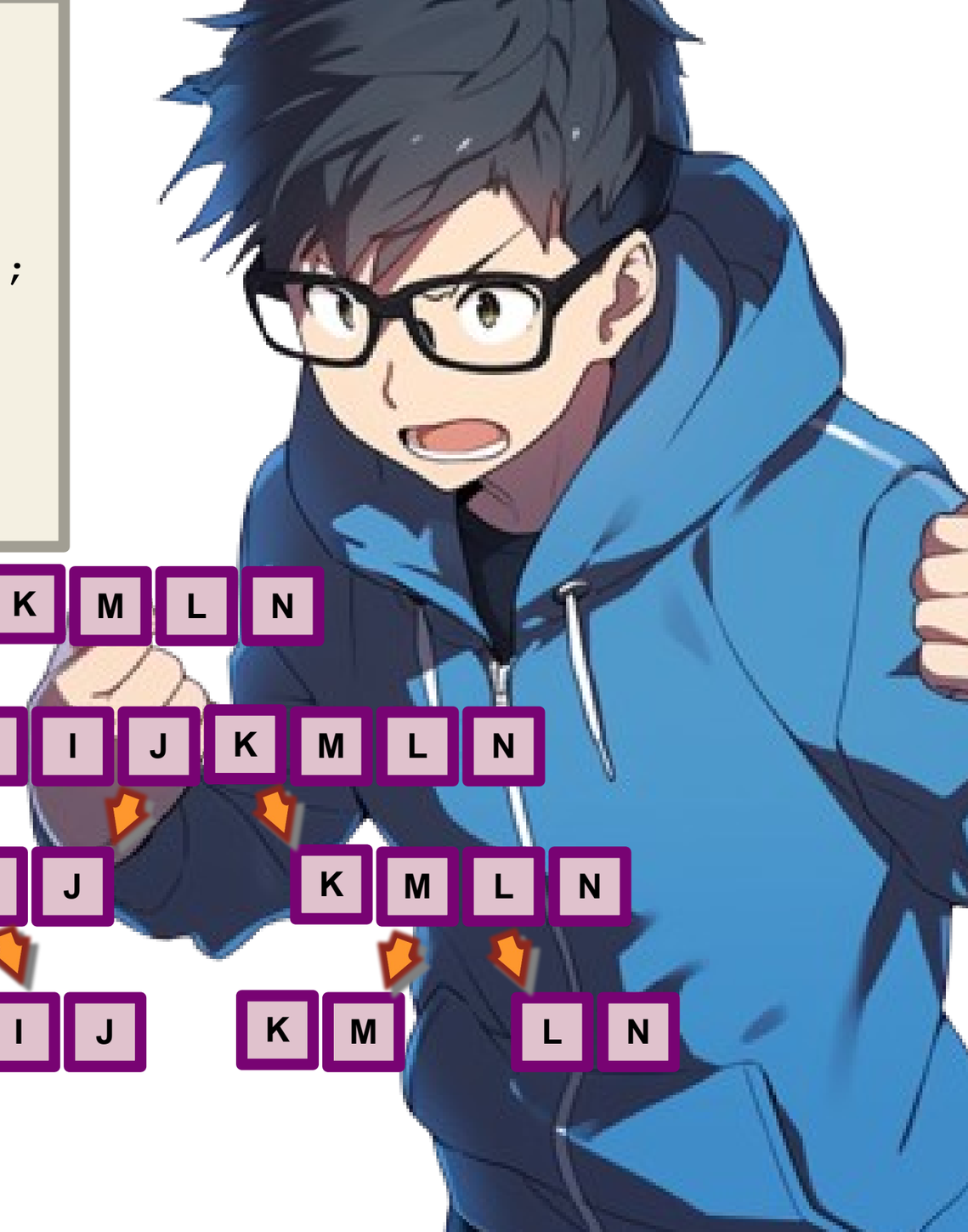
```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(list.subList(0, half));
  List<T> right= sort(list.subList(half, size));
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```



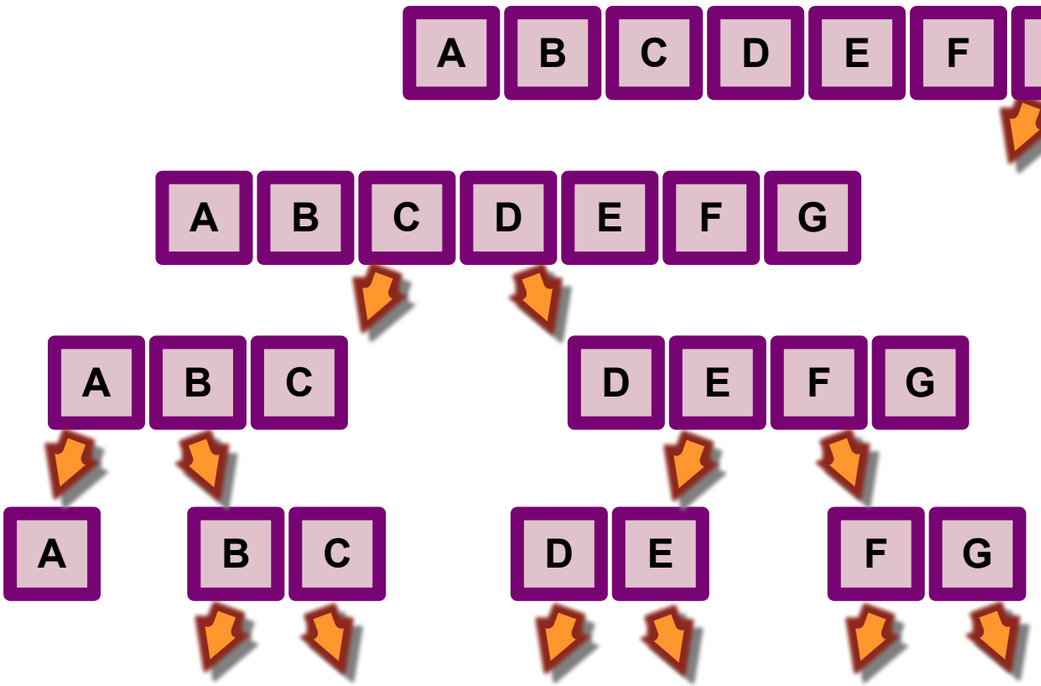


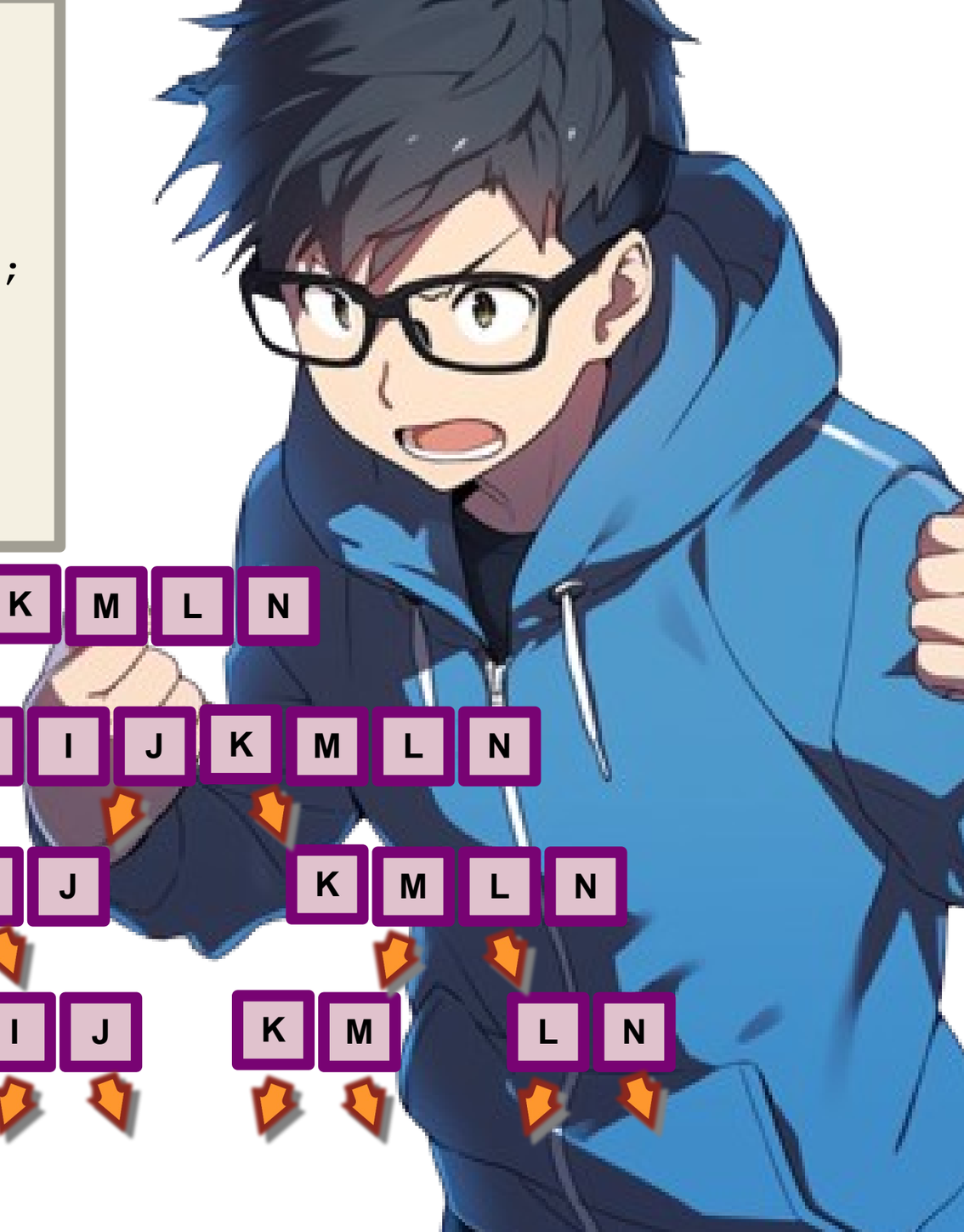
```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(list.subList(0, half));
  List<T> right= sort(list.subList(half, size));
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```



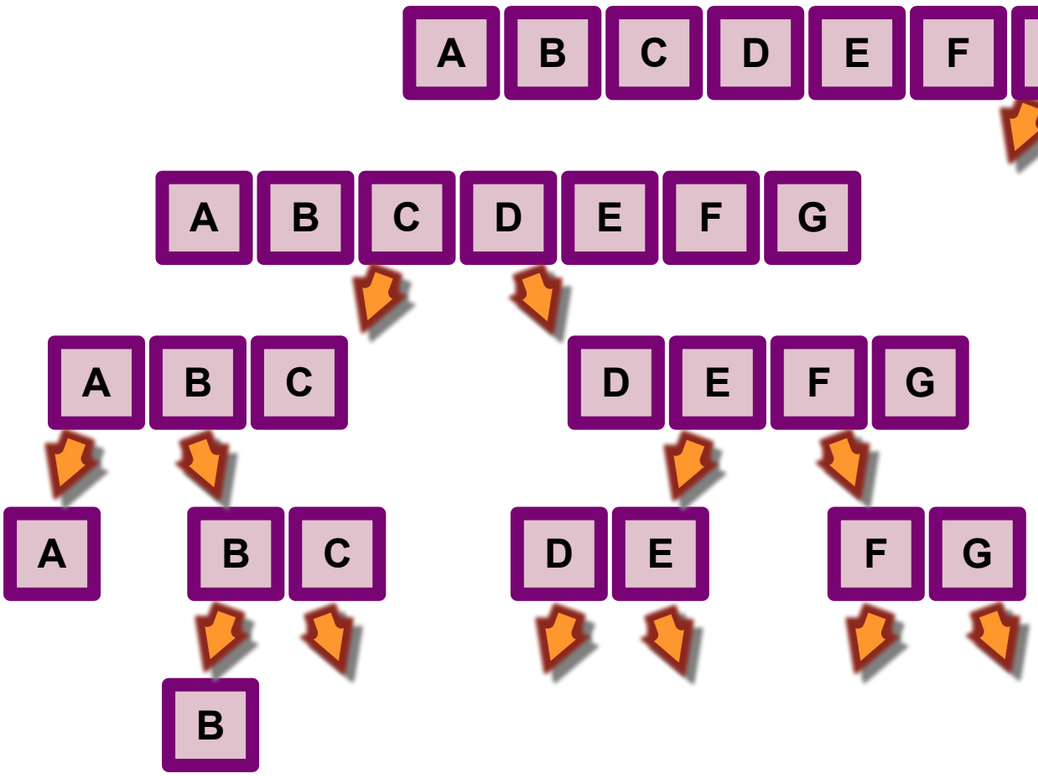


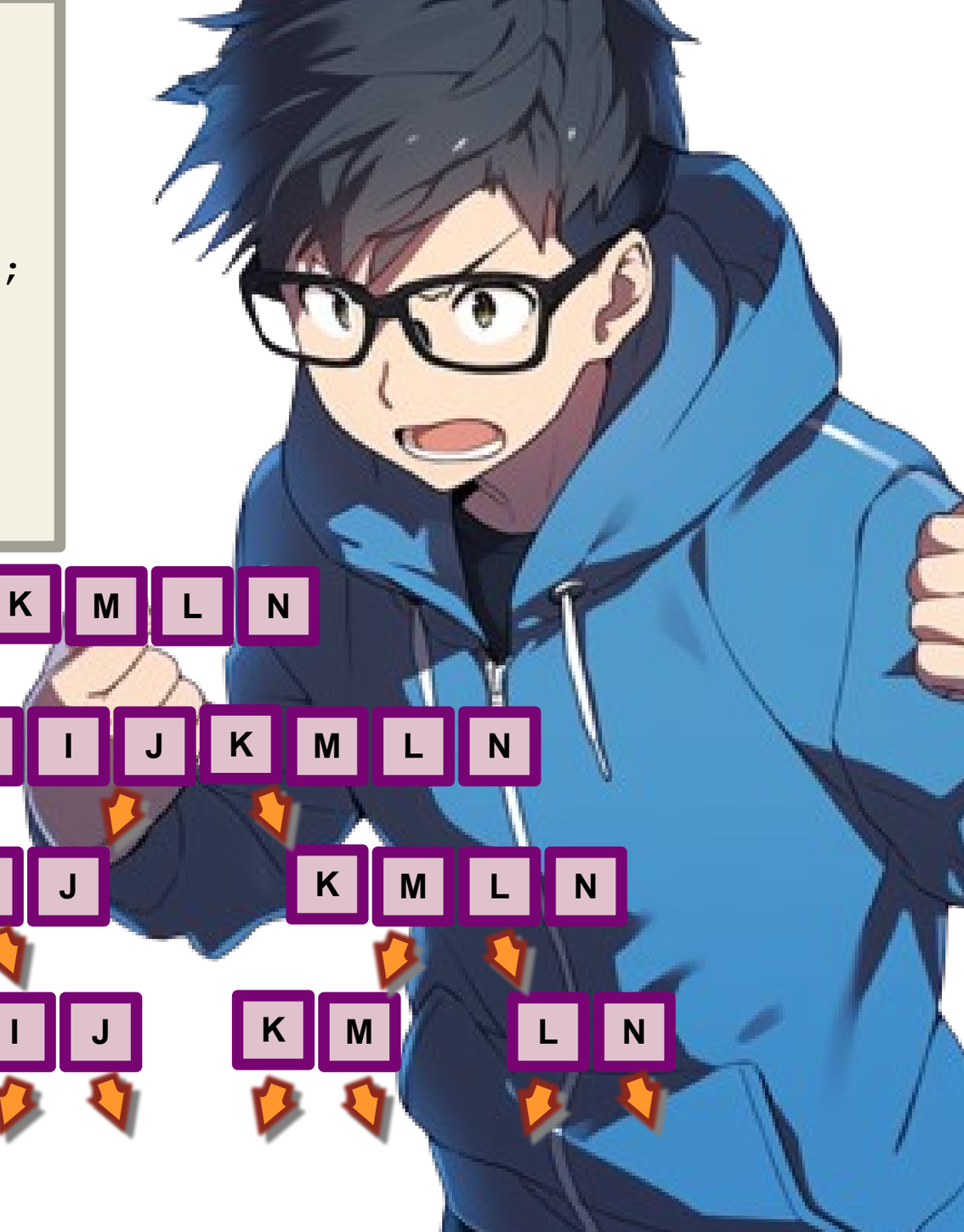
```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(list.subList(0, half));
  List<T> right= sort(list.subList(half, size));
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```



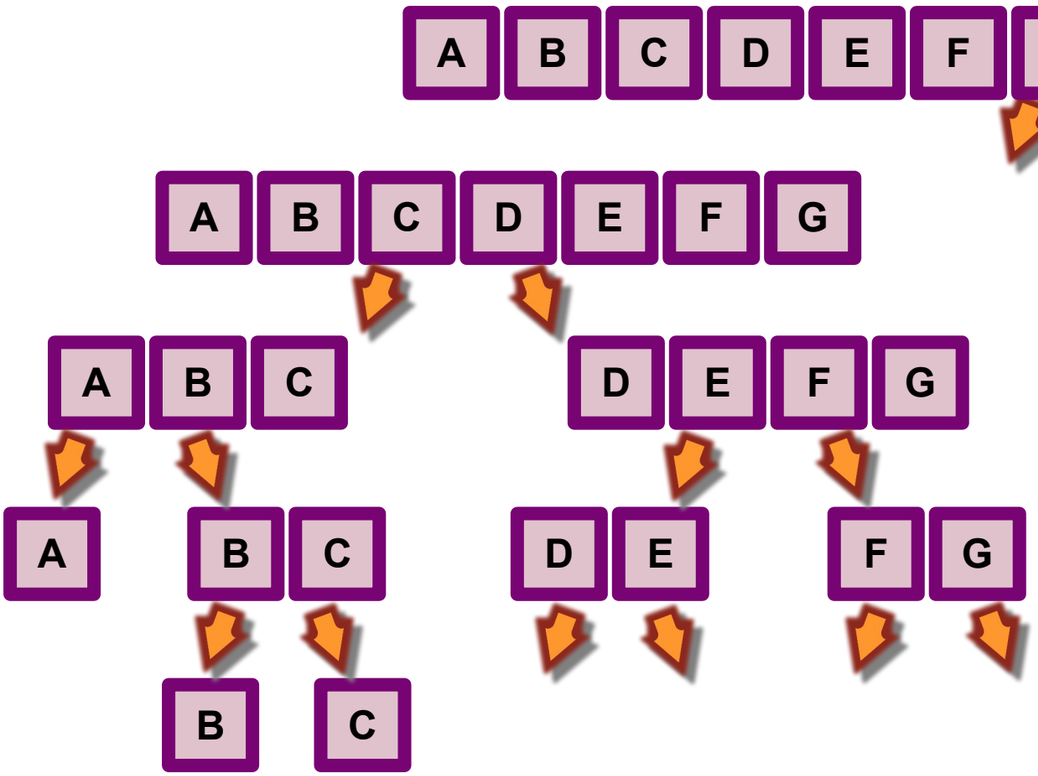


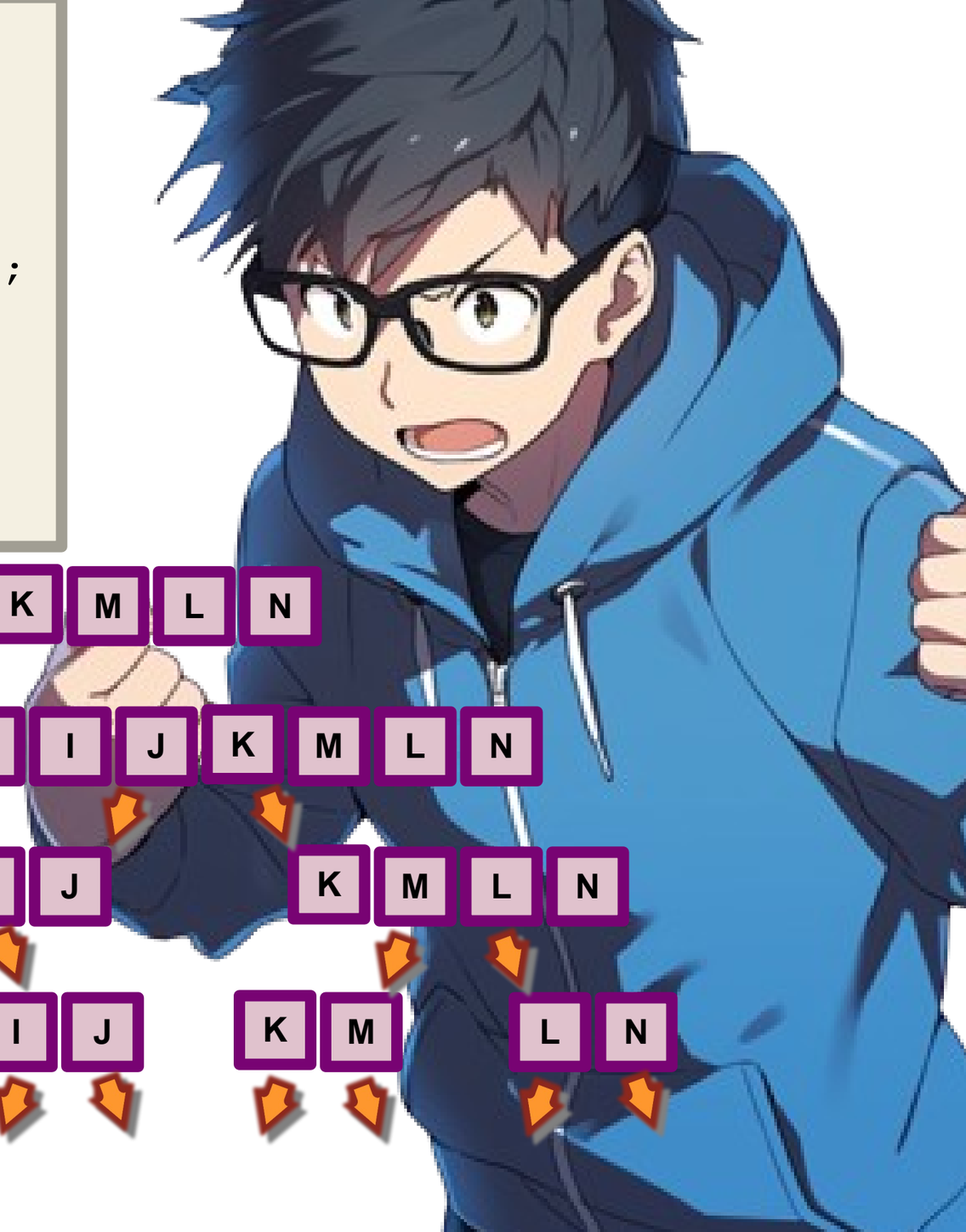
```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(list.subList(0, half));
  List<T> right= sort(list.subList(half, size));
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```



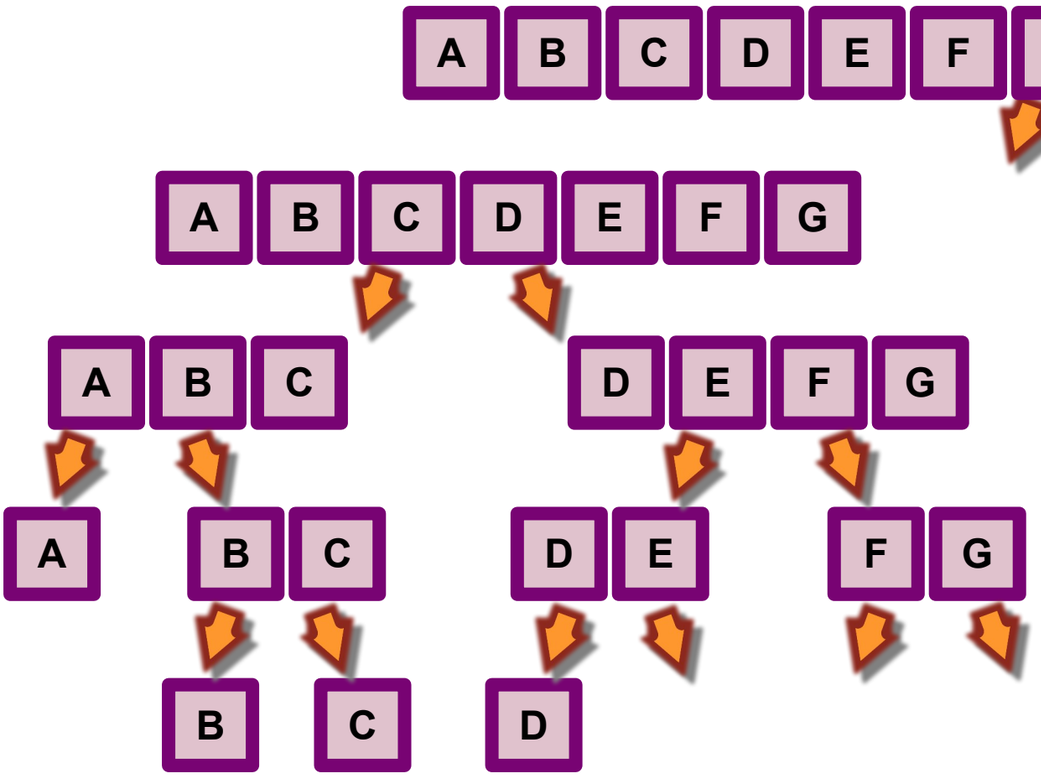


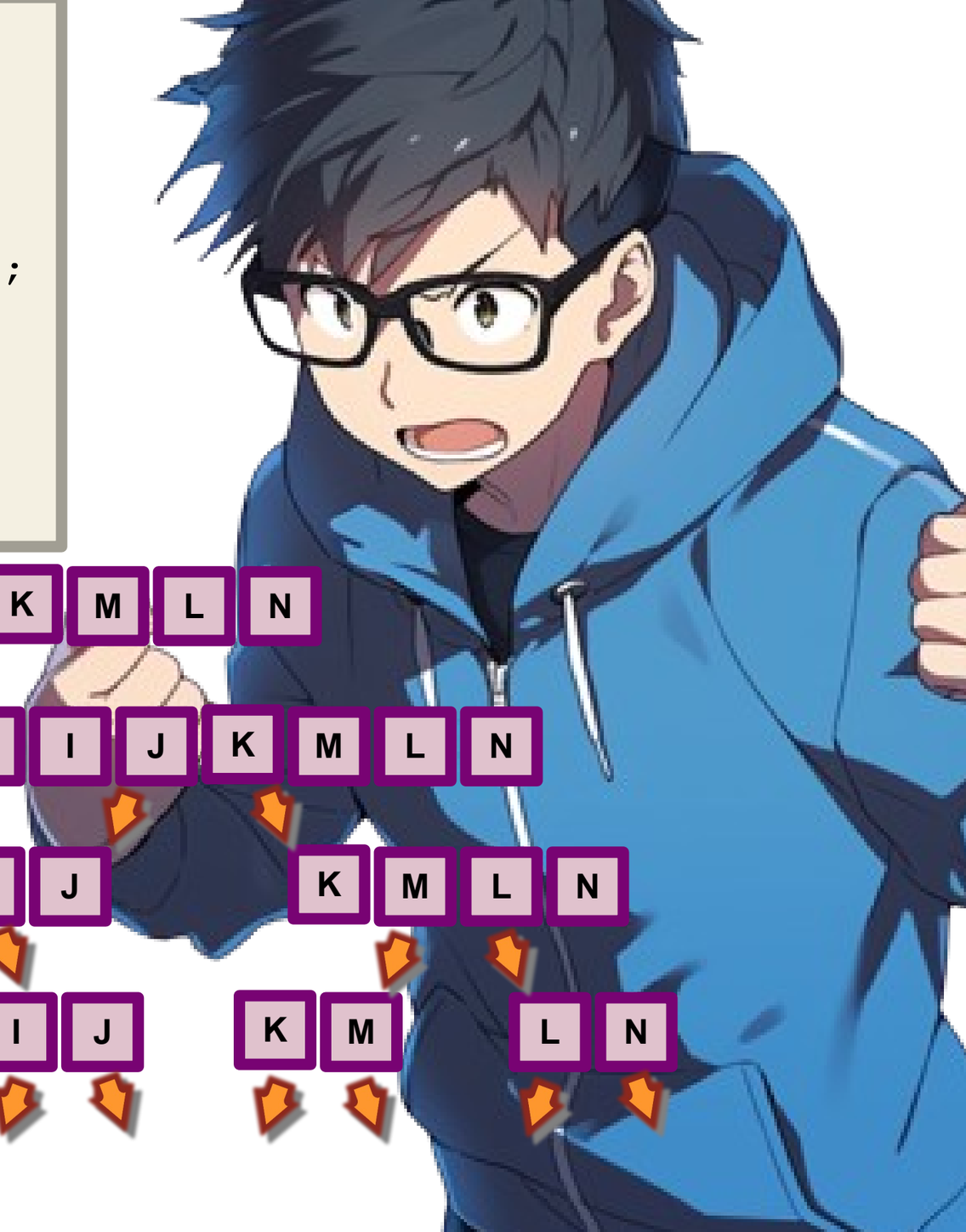
```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(list.subList(0, half));
  List<T> right= sort(list.subList(half, size));
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```



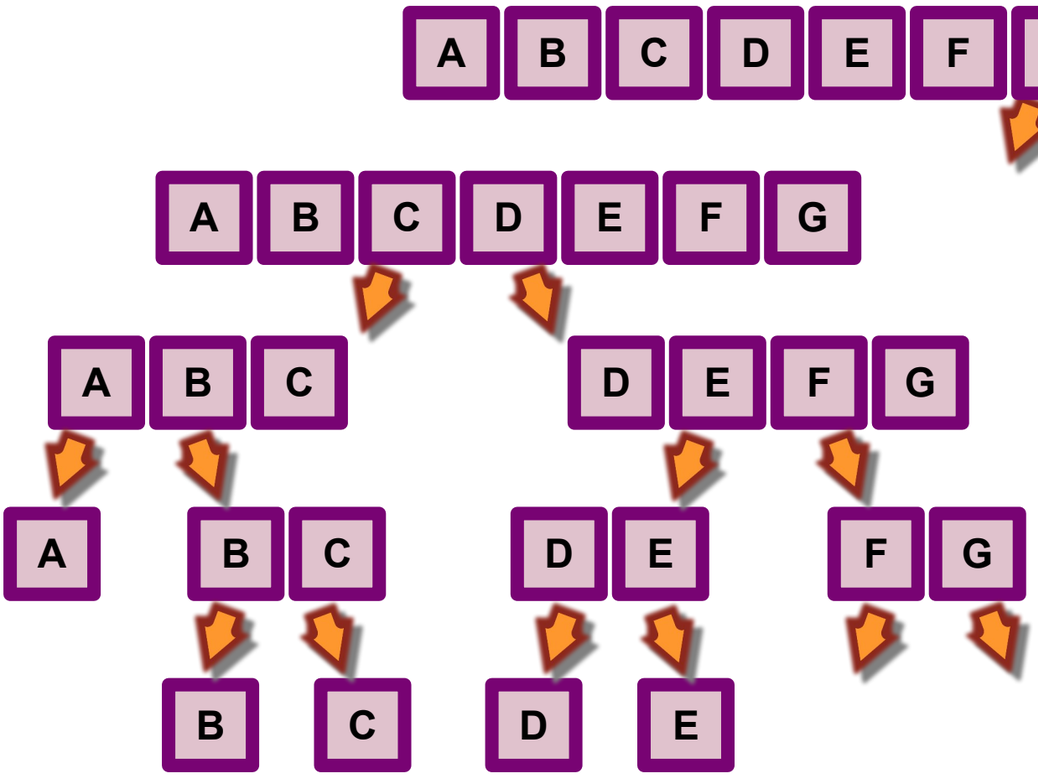


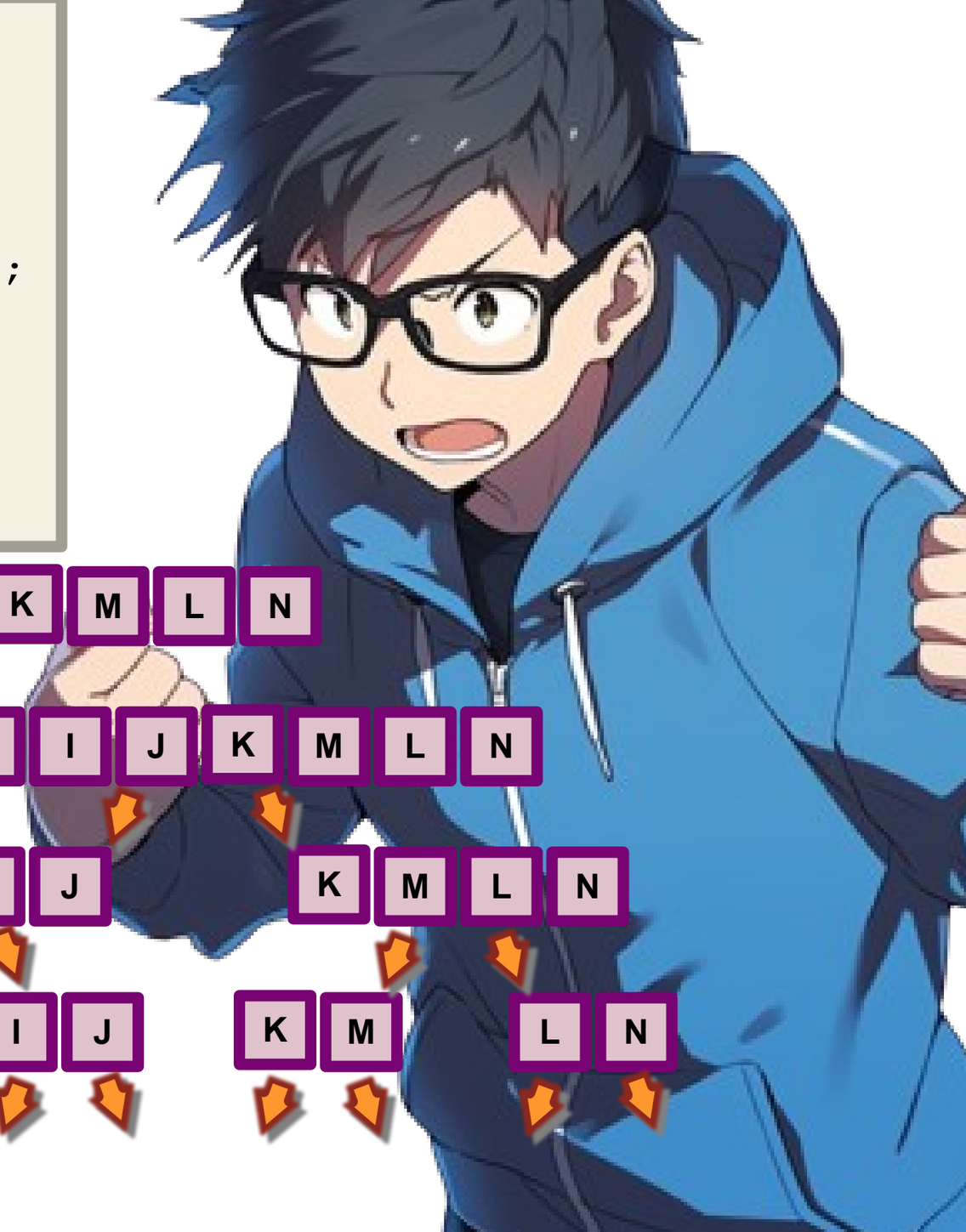
```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(list.subList(0, half));
  List<T> right= sort(list.subList(half, size));
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```



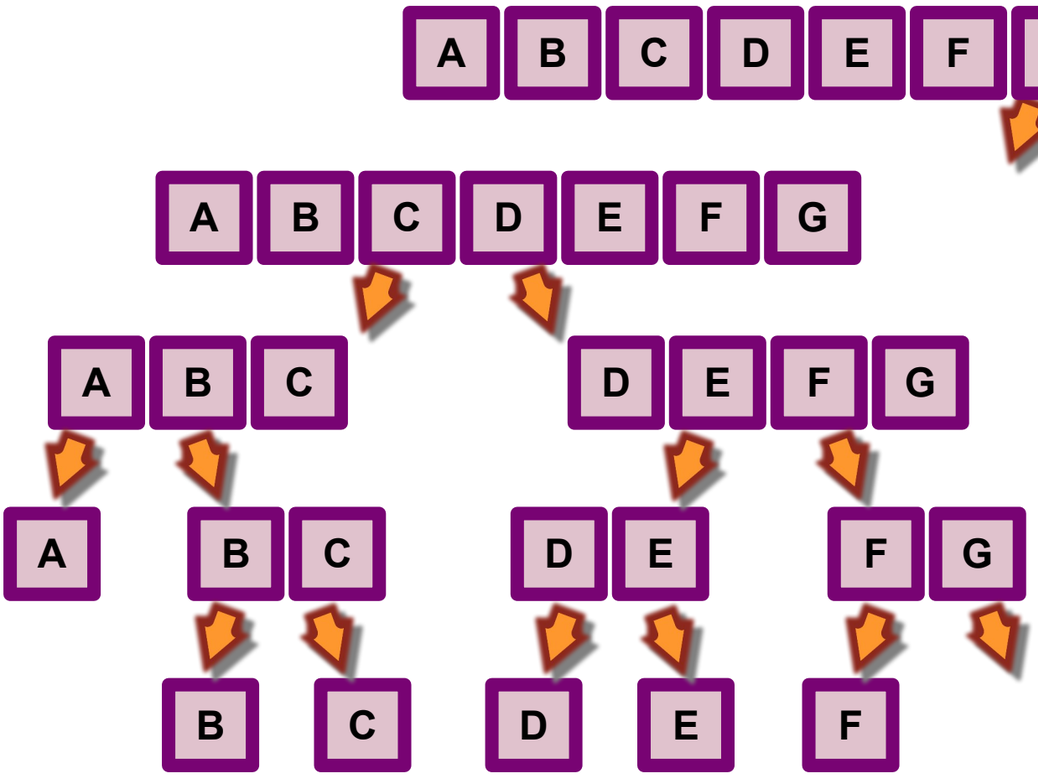


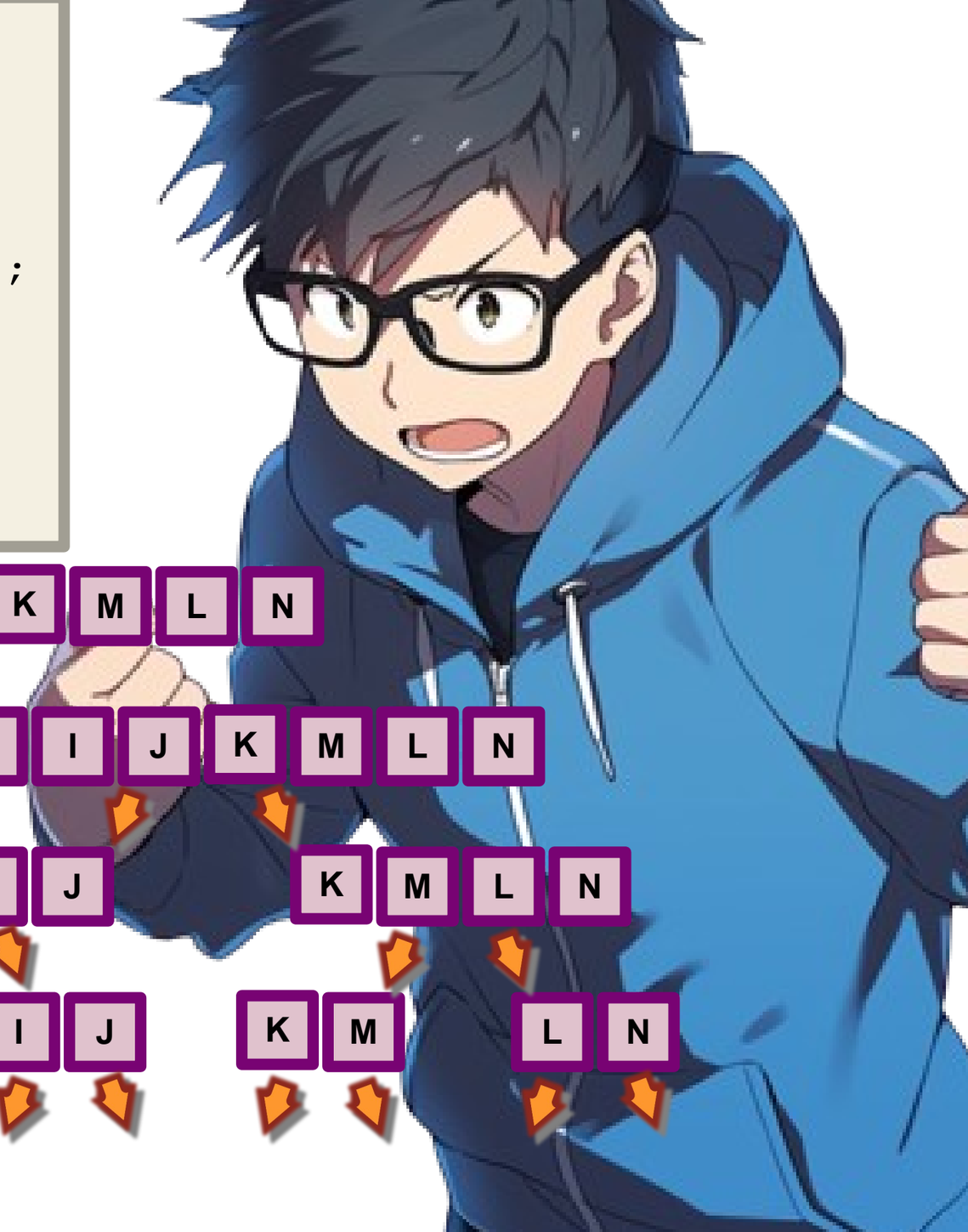
```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(list.subList(0, half));
  List<T> right= sort(list.subList(half, size));
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```



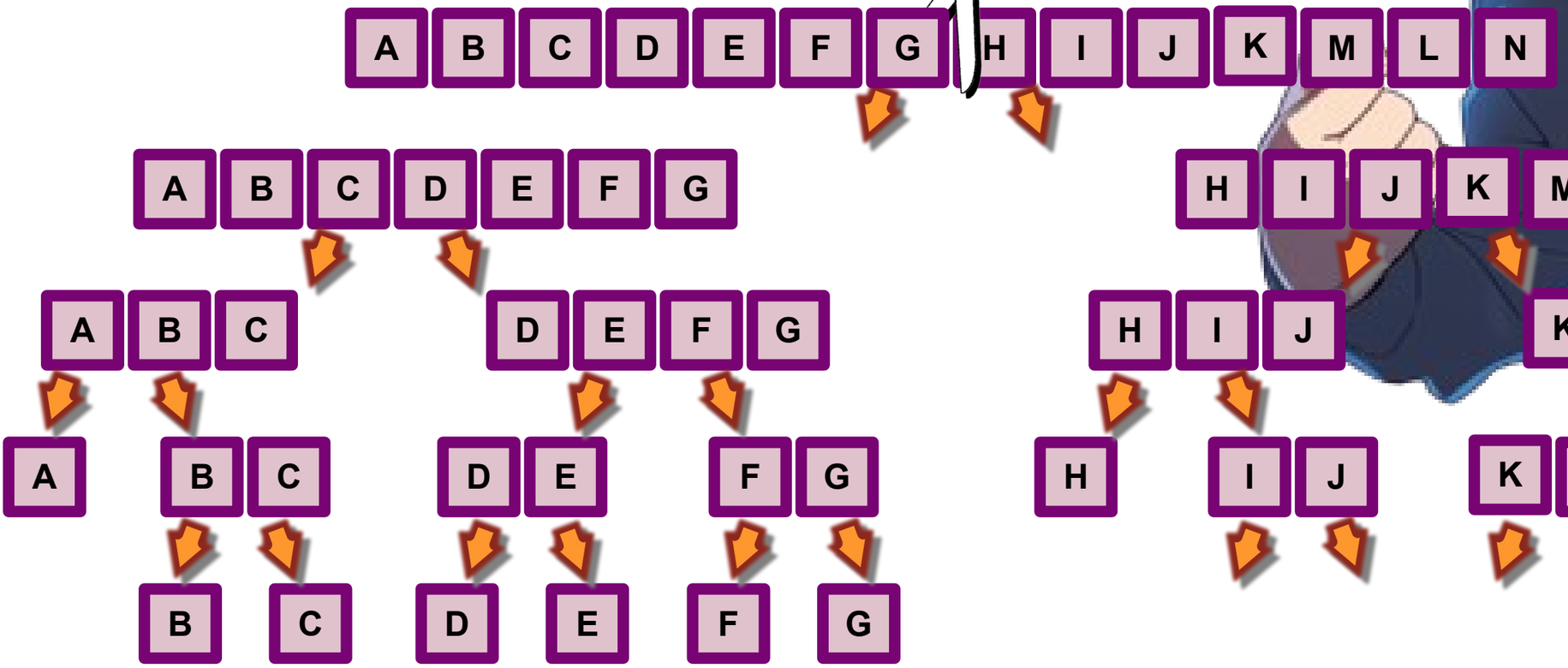


```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(list.subList(0, half));
  List<T> right= sort(list.subList(half, size));
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```

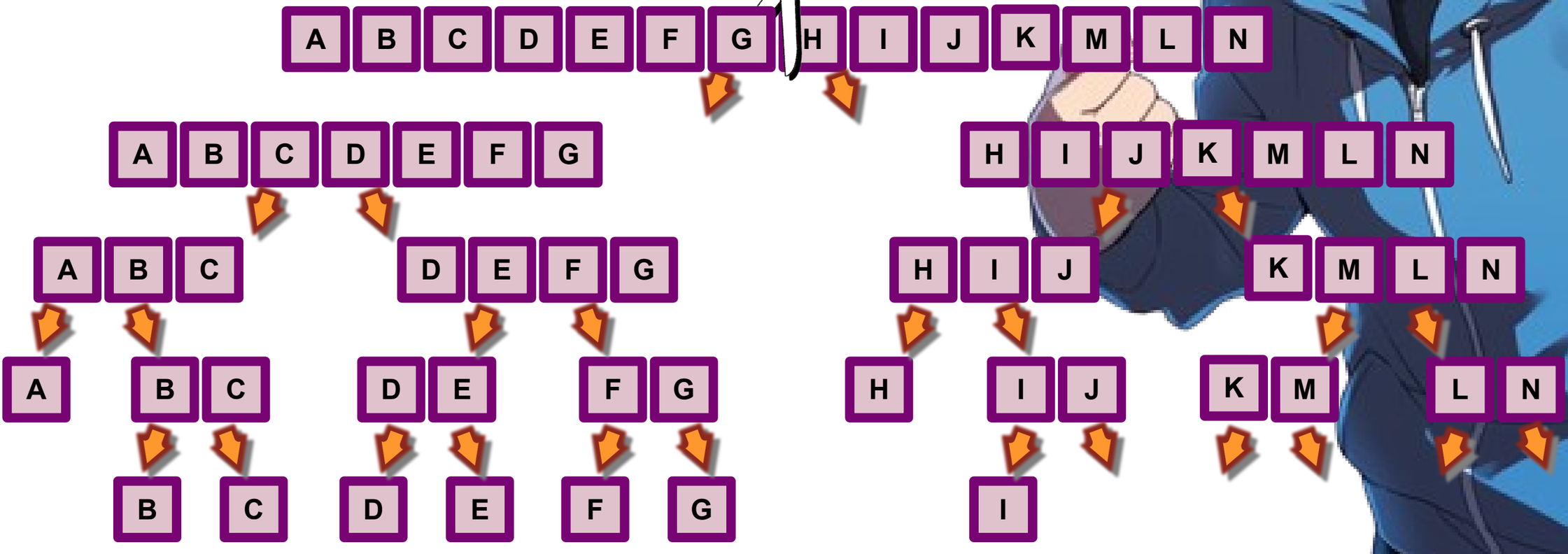


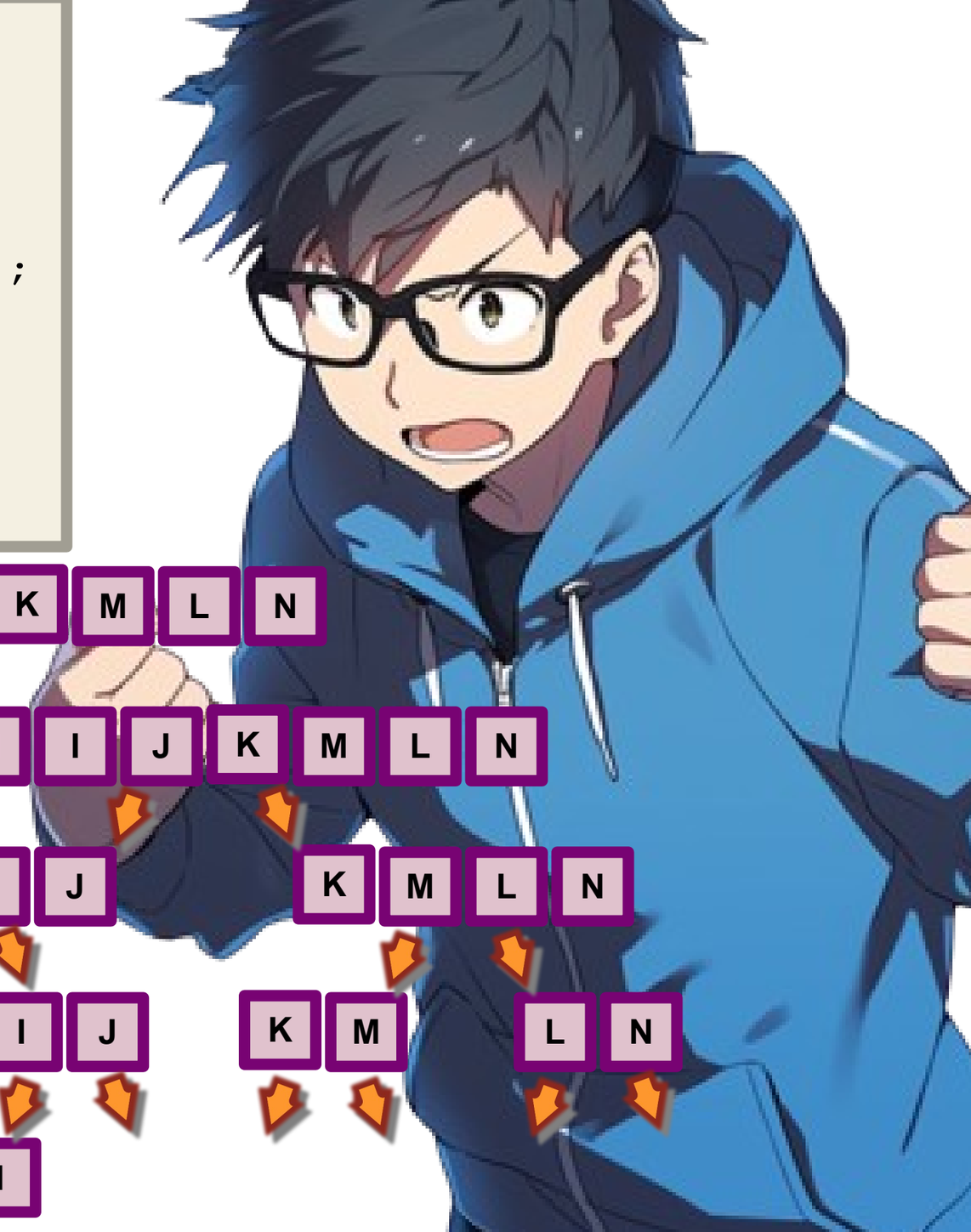


```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(list.subList(0, half));
  List<T> right= sort(list.subList(half, size));
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```

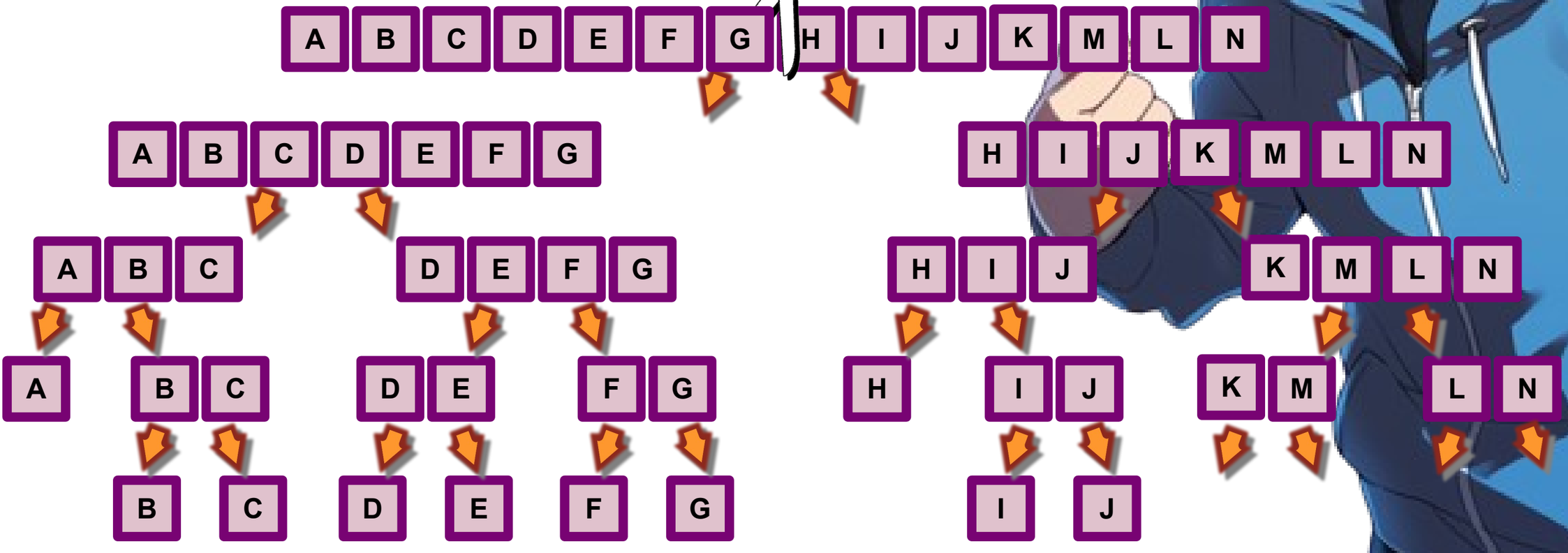


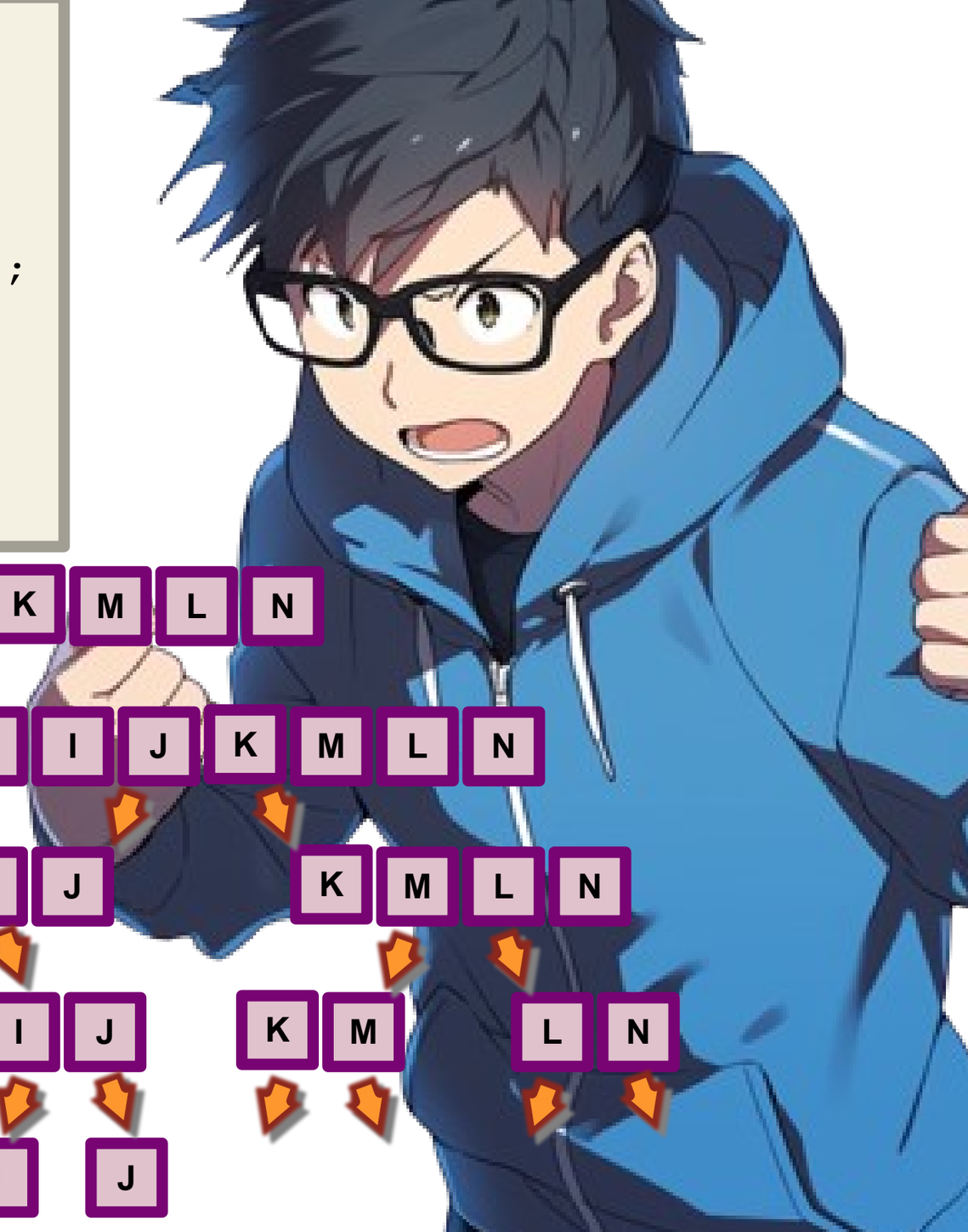
```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(list.subList(0, half));
  List<T> right= sort(list.subList(half, size));
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```



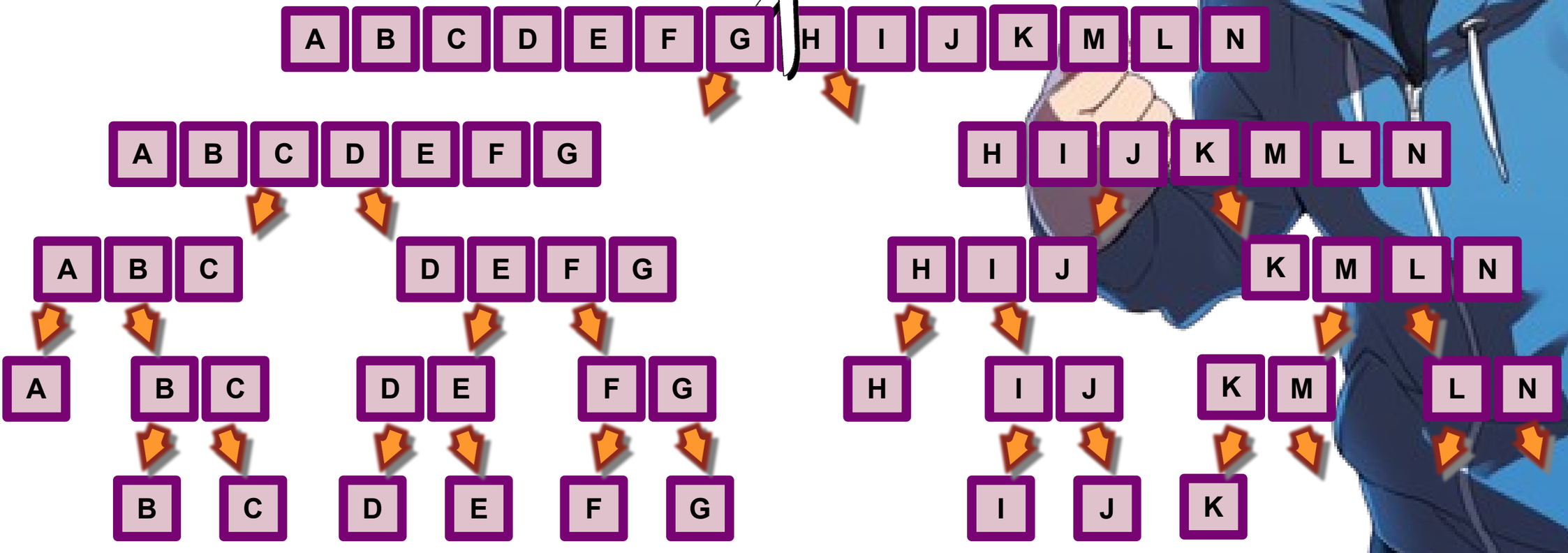


```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(list.subList(0, half));
  List<T> right= sort(list.subList(half, size));
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```

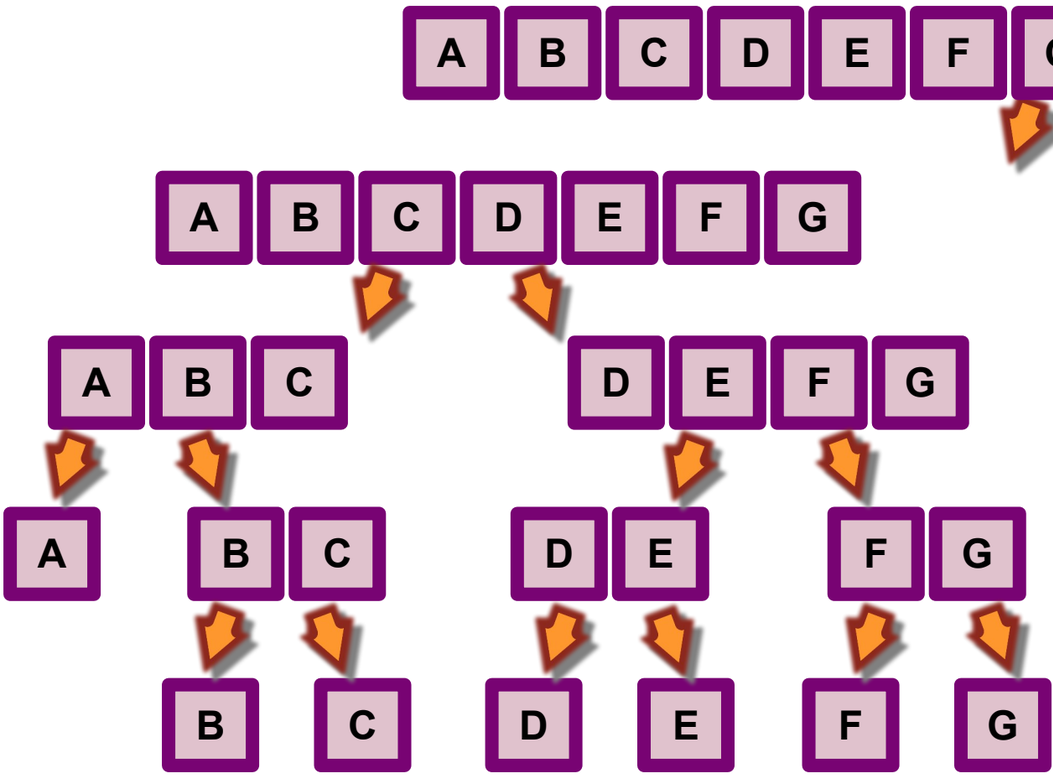


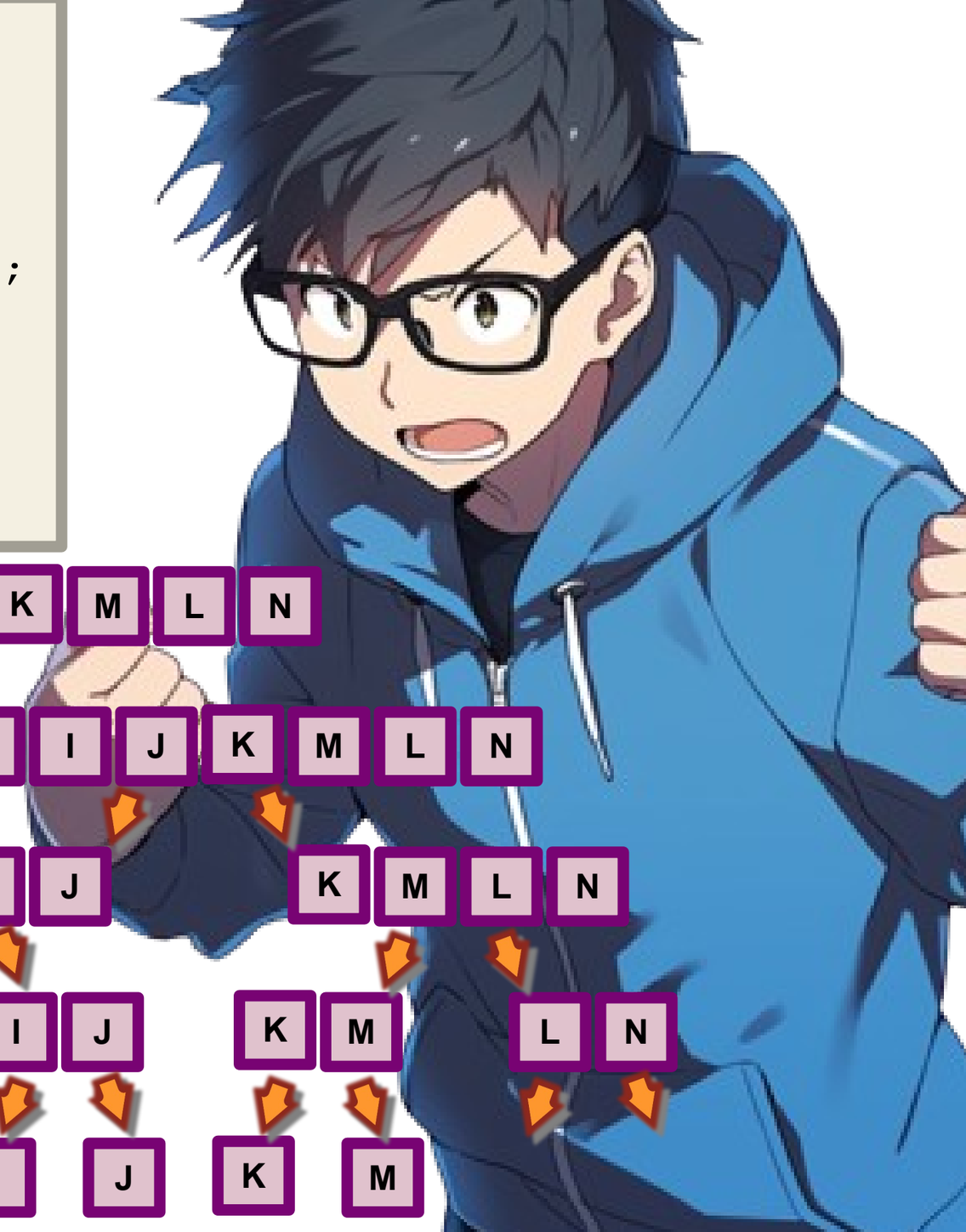


```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(list.subList(0, half));
  List<T> right= sort(list.subList(half, size));
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```

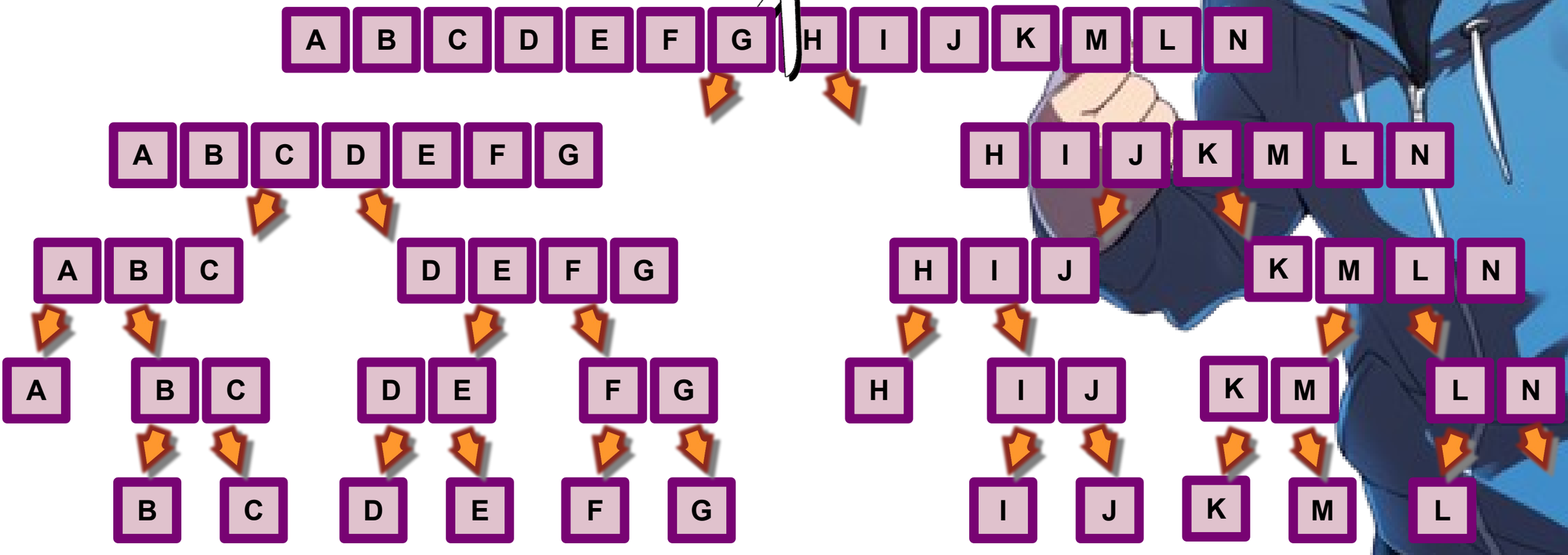


```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(list.subList(0, half));
  List<T> right= sort(list.subList(half, size));
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```

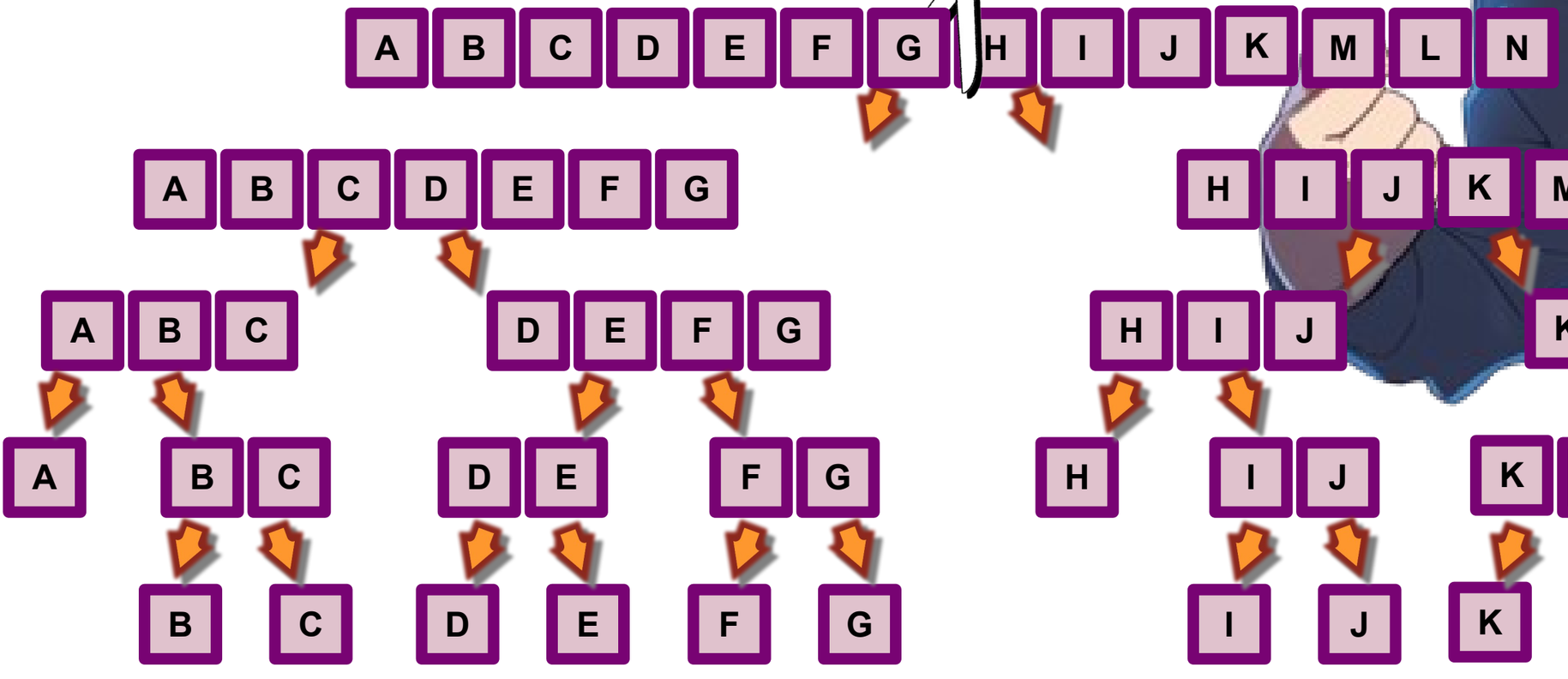


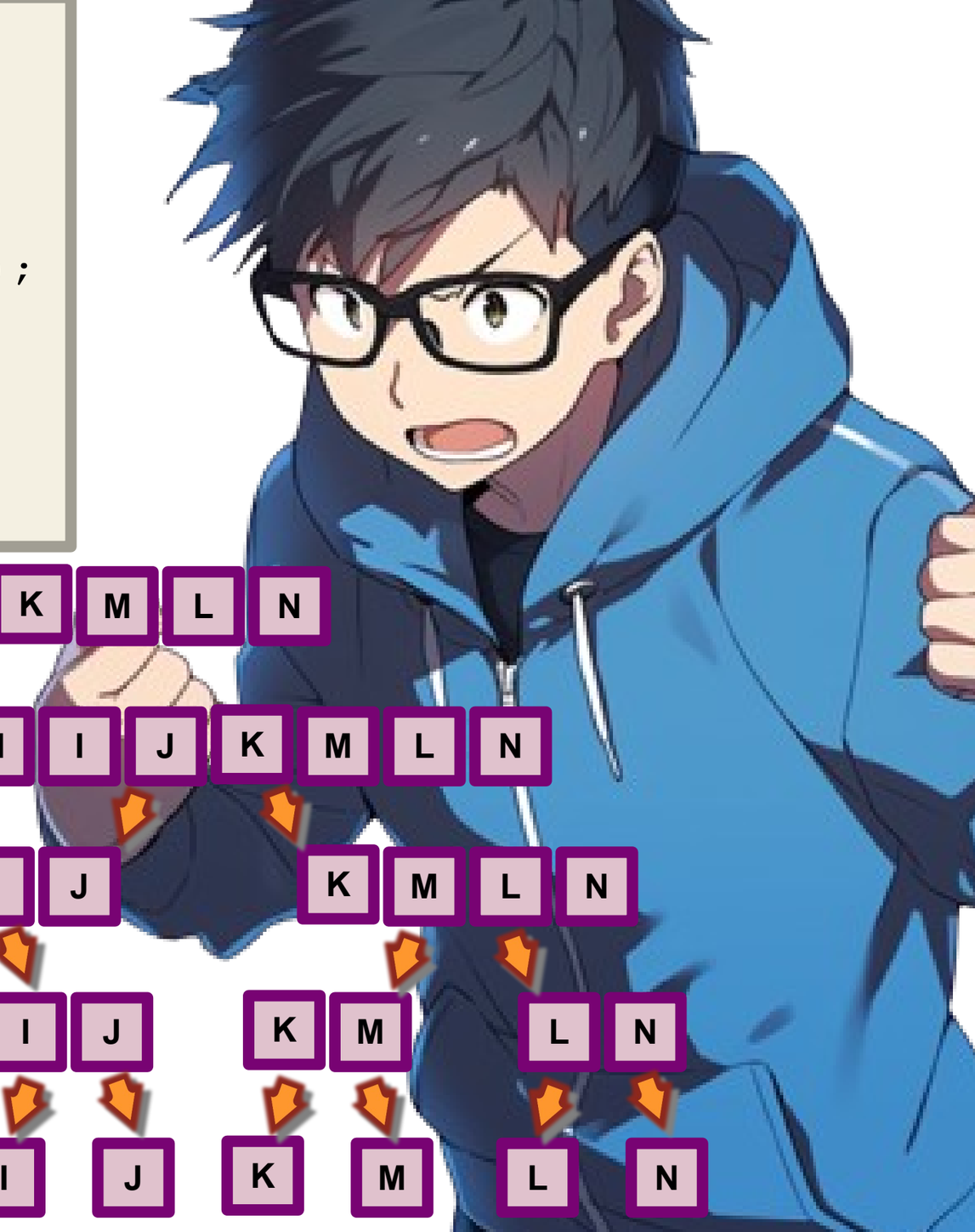


```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(list.subList(0, half));
  List<T> right= sort(list.subList(half, size));
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```

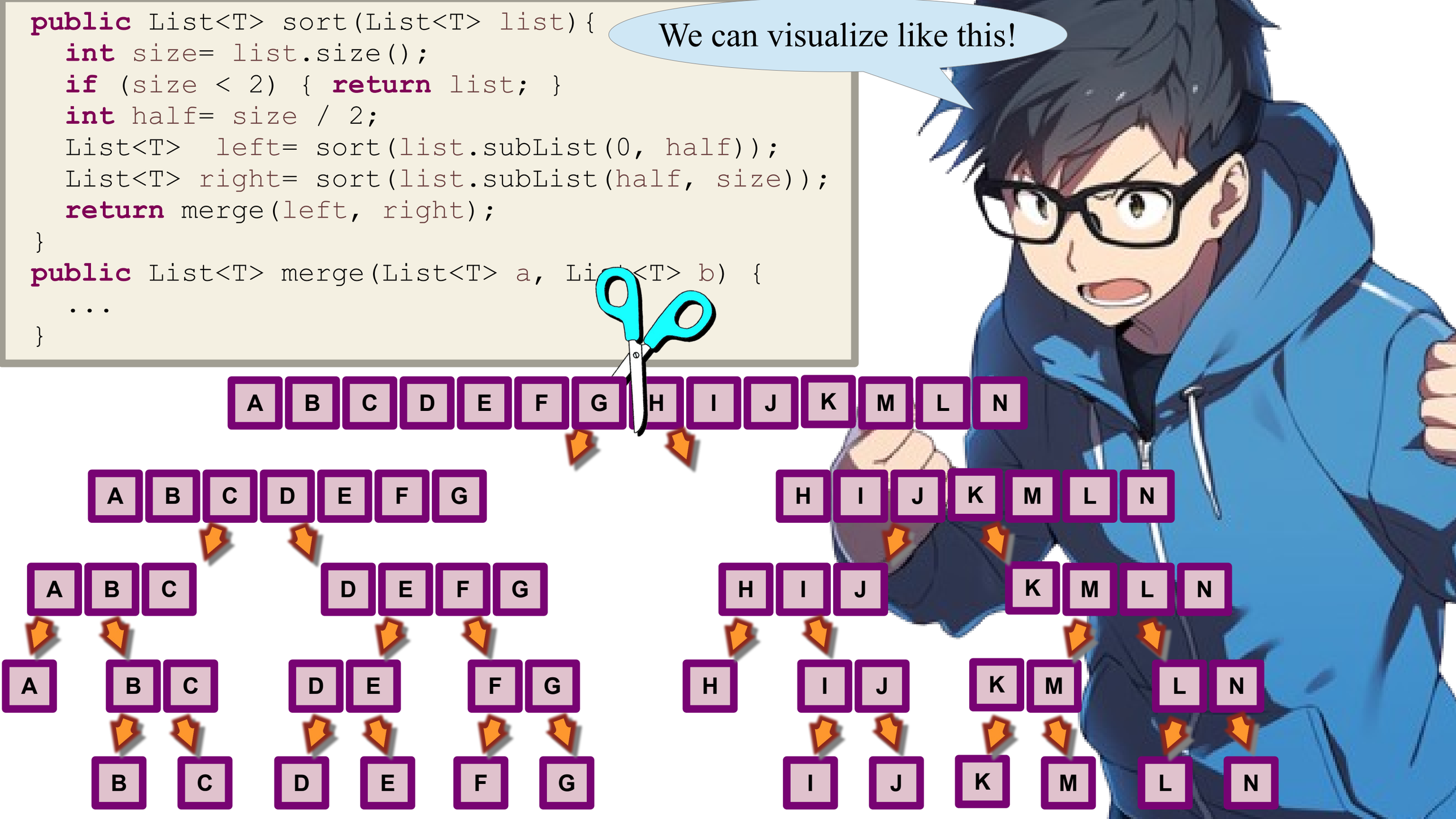


```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(list.subList(0, half));
  List<T> right= sort(list.subList(half, size));
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```

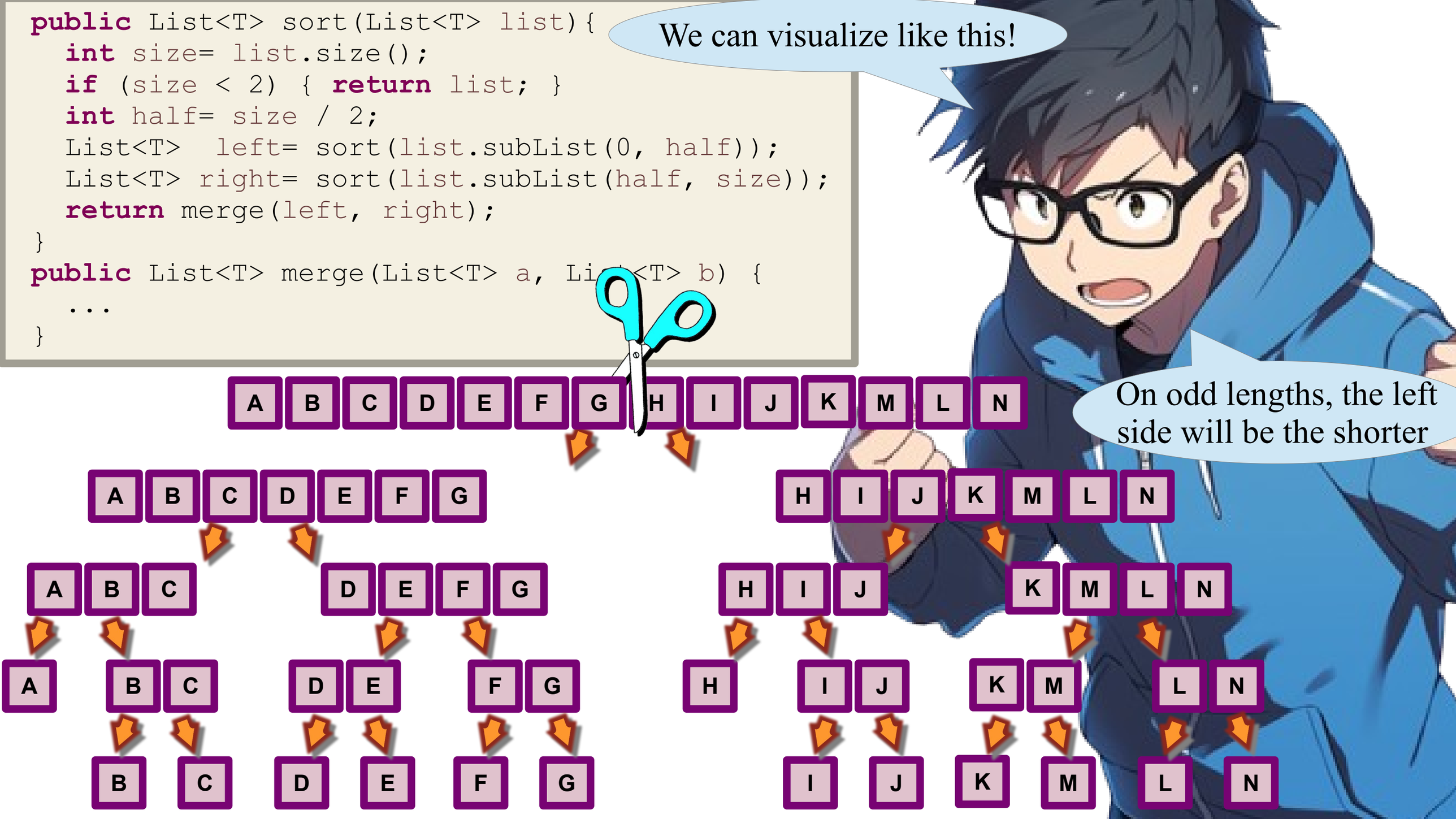




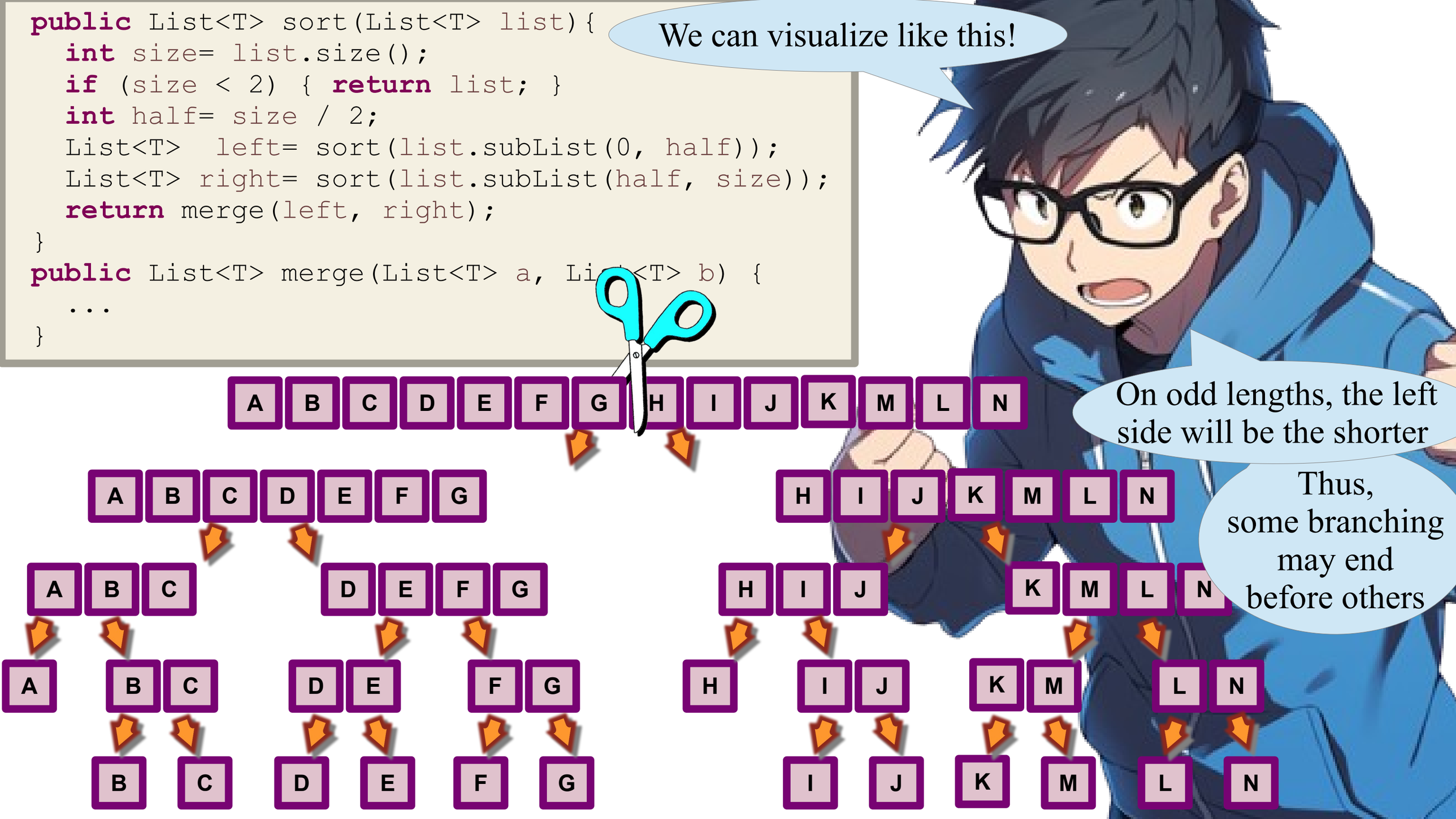
```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(list.subList(0, half));
  List<T> right= sort(list.subList(half, size));
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```



```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(list.subList(0, half));
  List<T> right= sort(list.subList(half, size));
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```



```java
public List<T> sort(List<T> list){
  int size= list.size();
  if (size < 2) { return list; }
  int half= size / 2;
  List<T>  left= sort(list.subList(0, half));
  List<T> right= sort(list.subList(half, size));
  return merge(left, right);
}
public List<T> merge(List<T> a, List<T> b) {
  ...
}
```




Sort cuts into pieces


Sort cuts into pieces
Merge glues two pieces together

```java
public List<T> merge(List<T> a, List<T> b) {



}
```


```java
public List<T> merge(List<T> a, List<T> b) {



}
```


```
public List<T> merge(List<T> a, List<T> b) {



}
```


```java
public List<T> merge(List<T> a, List<T> b) {
  List<T> res= new ArrayList<>(a.size() + b.size());

  return res;
}
```


```
public List<T> merge(List<T> a, List<T> b) {
  List<T> res= new ArrayList<>(a.size() + b.size());

  return res;
}
```


```
public List<T> merge(List<T> a, List<T> b) {
  List<T> res= new ArrayList<>(a.size() + b.size());




  return res;
}
```

```java
public List<T> merge(List<T> a, List<T> b) {
  List<T> res= new ArrayList<>(a.size() + b.size());
  int indexA= 0;
  int indexB= 0;
  while (                          ???                          ){
    T currA= a.get(indexA);
    T currB= b.get(indexB);









    }









  return res;
}
```


```java
public List<T> merge(List<T> a, List<T> b) {
  List<T> res= new ArrayList<>(a.size() + b.size());
  int indexA= 0;
  int indexB= 0;
  while (                          ???                          ){
    T currA= a.get(indexA);
    T currB= b.get(indexB);









    }









  return res;
}
```


```java
public List<T> merge(List<T> a, List<T> b) {
  List<T> res= new ArrayList<>(a.size() + b.size());
  int indexA= 0;
  int indexB= 0;
  while (                          ???                          ){
    T currA= a.get(indexA);
    T currB= b.get(indexB);

  return res;
}
```


```java
public List<T> merge(List<T> a, List<T> b) {
  List<T> res= new ArrayList<>(a.size() + b.size());
  int indexA= 0;
  int indexB= 0;
  while (                          ???                          ){
    T currA= a.get(indexA);
    T currB= b.get(indexB);

  return res;
}
```

We can use two cursors,
indexA and indexB

The current element in indexA is currA,
same for indexB and currB

```java
public List<T> merge(List<T> a, List<T> b) {
  List<T> res= new ArrayList<>(a.size() + b.size());
  int indexA= 0;
  int indexB= 0;
  while (                          ???                          ){
    T currA= a.get(indexA);
    T currB= b.get(indexB);
    if (currA.compareTo(currB) <= 0){
      res.add(currA);
      indexA++;
    }
    else {


    }
  }









  return res;
}
```


```java
public List<T> merge(List<T> a, List<T> b) {
  List<T> res= new ArrayList<>(a.size() + b.size());
  int indexA= 0;
  int indexB= 0;
  while (                          ???                          ){
    T currA= a.get(indexA);
    T currB= b.get(indexB);
    if (currA.compareTo(currB) <= 0){
      res.add(currA);
      indexA++;
    }
    else {


    }
  }









  return res;
}
```


```java
public List<T> merge(List<T> a, List<T> b) {
  List<T> res= new ArrayList<>(a.size() + b.size());
  int indexA= 0;
  int indexB= 0;
  while (                          ???                          ){
    T currA= a.get(indexA);
    T currB= b.get(indexB);
    if (currA.compareTo(currB) <= 0){
      res.add(currA);
      indexA++;
    }
    else {


  return res;
}
```


```java
public List<T> merge(List<T> a, List<T> b) {
  List<T> res= new ArrayList<>(a.size() + b.size());
  int indexA= 0;
  int indexB= 0;
  while (                          ???                          ){
    T currA= a.get(indexA);
    T currB= b.get(indexB);
    if (currA.compareTo(currB) <= 0){
      res.add(currA);
      indexA++;
    }
    else {


  return res;
}
```


```java
public List<T> merge(List<T> a, List<T> b) {
  List<T> res= new ArrayList<>(a.size() + b.size());
  int indexA= 0;
  int indexB= 0;
  while (                          ???                          ){
    T currA= a.get(indexA);
    T currB= b.get(indexB);
    if (currA.compareTo(currB) <= 0){
      res.add(currA);
      indexA++;
    }
    else {
      res.add(currB);
      indexB++;
    }
  }









  return res;
}
```


```java
public List<T> merge(List<T> a, List<T> b) {
  List<T> res= new ArrayList<>(a.size() + b.size());
  int indexA= 0;
  int indexB= 0;
  while (                          ???                          ){
    T currA= a.get(indexA);
    T currB= b.get(indexB);
    if (currA.compareTo(currB) <= 0){
      res.add(currA);
      indexA++;
    }
    else {
      res.add(currB);
      indexB++;
    }
  }


  return res;
}
```


```java
public List<T> merge(List<T> a, List<T> b) {
  List<T> res= new ArrayList<>(a.size() + b.size());
  int indexA= 0;
  int indexB= 0;
  while (                          ???                          ){
    T currA= a.get(indexA);
    T currB= b.get(indexB);
    if (currA.compareTo(currB) <= 0){
      res.add(currA);
      indexA++;
    }
    else {
      res.add(currB);
      indexB++;
    }
  }
}
```

If currB is the smaller,

we add it to the result
and we move indexB forward

```java
public List<T> merge(List<T> a, List<T> b) {
  List<T> res= new ArrayList<>(a.size() + b.size());
  int indexA= 0;
  int indexB= 0;
  while (                          ???                          ){
    T currA= a.get(indexA);
    T currB= b.get(indexB);
    if (currA.compareTo(currB) <= 0){
      res.add(currA);
      indexA++;
    }
    else {
      res.add(currB);
      indexB++;
    }
  }









  return res;
}
```


```java
public List<T> merge(List<T> a, List<T> b) {
  List<T> res= new ArrayList<>(a.size() + b.size());
  int indexA= 0;
  int indexB= 0;
  while (                    ???                    ){
    T currA= a.get(indexA);
    T currB= b.get(indexB);
    if (currA.compareTo(currB) <= 0){
      res.add(currA);
      indexA++;
    }
    else {
      res.add(currB);
      indexB++;
    }
  }

  return res;
}
```

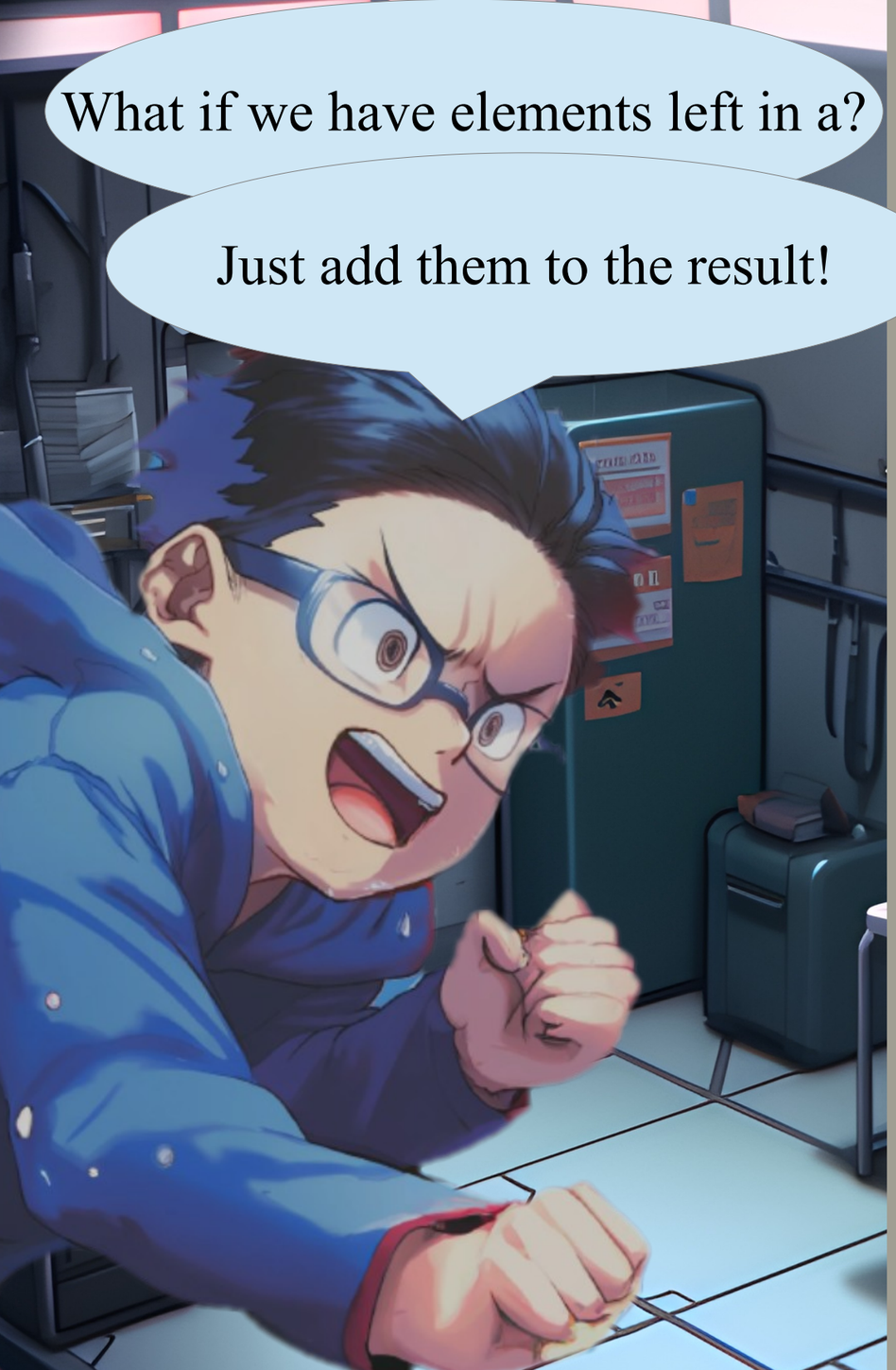
```java
public List<T> merge(List<T> a, List<T> b) {
  List<T> res= new ArrayList<>(a.size() + b.size());
  int indexA= 0;
  int indexB= 0;
  while (indexA < a.size() && indexB < b.size()){
    T currA= a.get(indexA);
    T currB= b.get(indexB);
    if (currA.compareTo(currB) <= 0){
      res.add(currA);
      indexA++;
    }
    else {
      res.add(currB);
      indexB++;
    }
  }

  return res;
}
```


```java
public List<T> merge(List<T> a, List<T> b) {
  List<T> res= new ArrayList<>(a.size() + b.size());
  int indexA= 0;
  int indexB= 0;
  while (indexA < a.size() && indexB < b.size()){
    T currA= a.get(indexA);
    T currB= b.get(indexB);
    if (currA.compareTo(currB) <= 0){
      res.add(currA);
      indexA++;
    }
    else {
      res.add(currB);
      indexB++;
    }
  }









  return res;
}
```

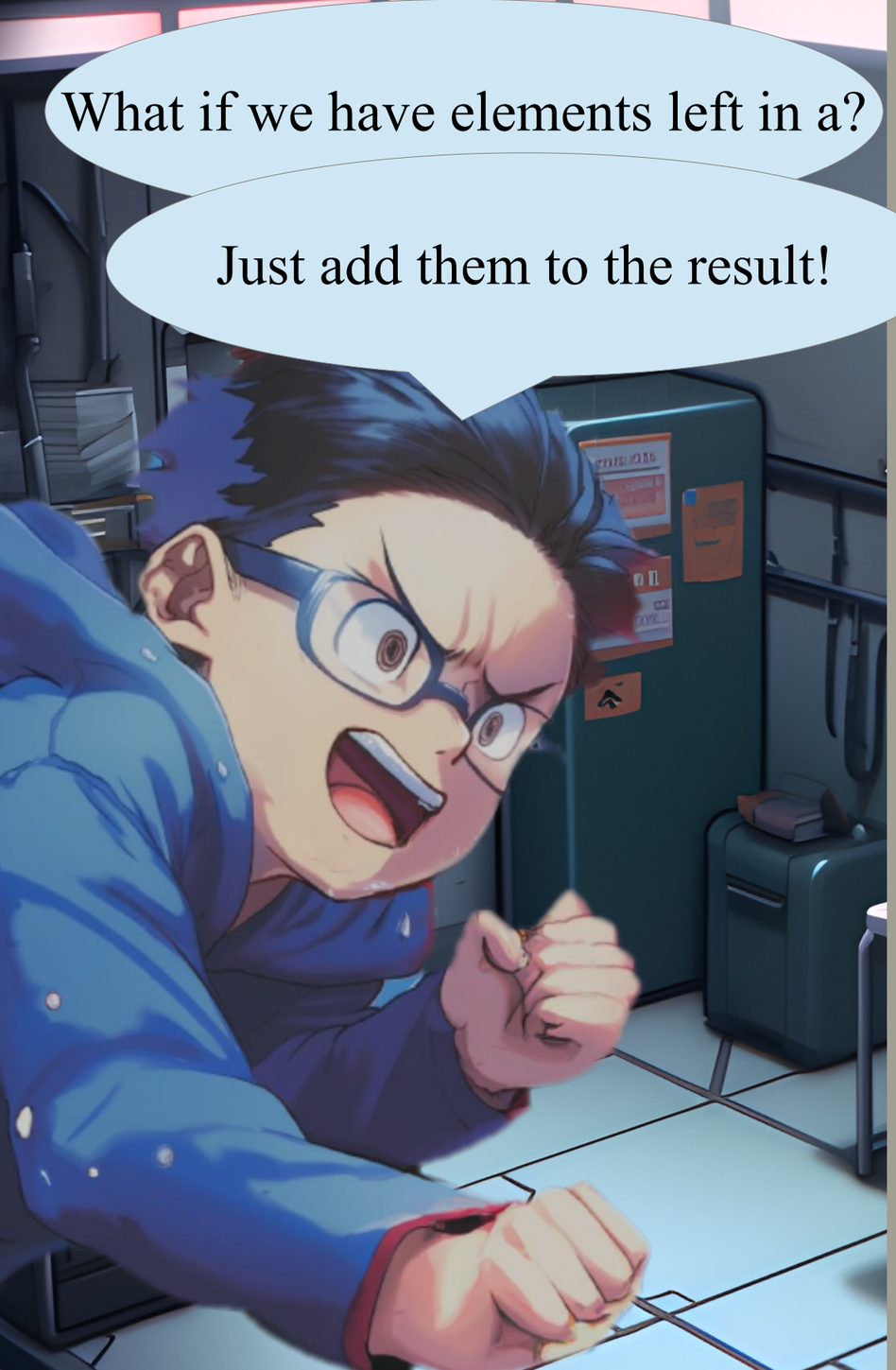

```java
public List<T> merge(List<T> a, List<T> b) {
  List<T> res= new ArrayList<>(a.size() + b.size());
  int indexA= 0;
  int indexB= 0;
  while (indexA < a.size() && indexB < b.size()){
    T currA= a.get(indexA);
    T currB= b.get(indexB);
    if (currA.compareTo(currB) <= 0){
      res.add(currA);
      indexA++;
    }
    else {
      res.add(currB);
      indexB++;
    }
  }

  return res;
}
```

```java
public List<T> merge(List<T> a, List<T> b) {
  List<T> res= new ArrayList<>(a.size() + b.size());
  int indexA= 0;
  int indexB= 0;
  while (indexA < a.size() && indexB < b.size()){
    T currA= a.get(indexA);
    T currB= b.get(indexB);
    if (currA.compareTo(currB) <= 0){
      res.add(currA);
      indexA++;
    }
    else {
      res.add(currB);
      indexB++;
    }
  }
  while (indexA < a.size()){
    res.add(a.get(indexA));
    indexA++;
  }



  return res;
}
```

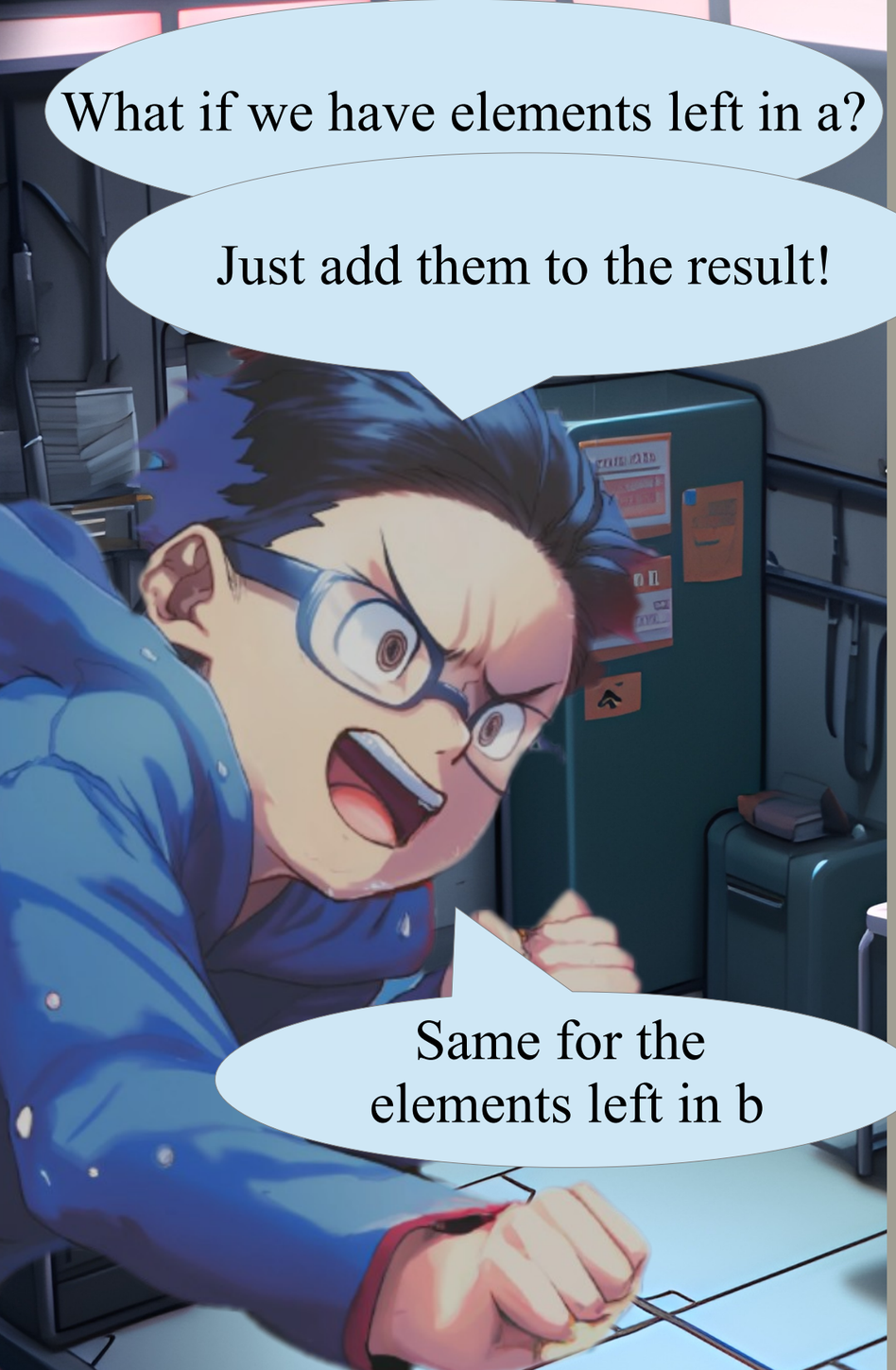

```java
public List<T> merge(List<T> a, List<T> b) {
  List<T> res= new ArrayList<>(a.size() + b.size());
  int indexA= 0;
  int indexB= 0;
  while (indexA < a.size() && indexB < b.size()){
    T currA= a.get(indexA);
    T currB= b.get(indexB);
    if (currA.compareTo(currB) <= 0){
      res.add(currA);
      indexA++;
    }
    else {
      res.add(currB);
      indexB++;
    }
  }
  while (indexA < a.size()){
    res.add(a.get(indexA));
    indexA++;
  }



  return res;
}
```



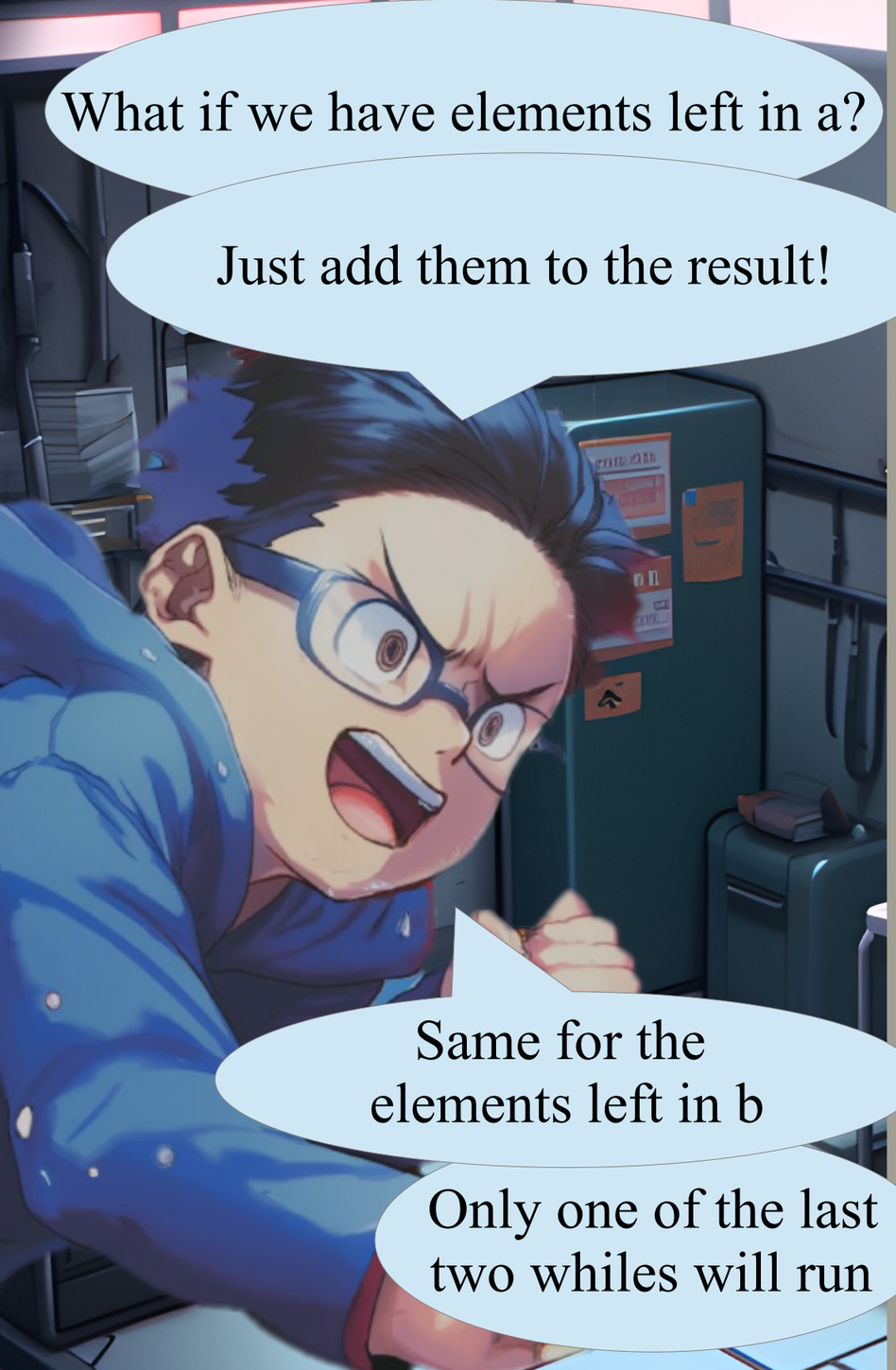

```java
public List<T> merge(List<T> a, List<T> b) {
  List<T> res= new ArrayList<>(a.size() + b.size());
  int indexA= 0;
  int indexB= 0;
  while (indexA < a.size() && indexB < b.size()){
    T currA= a.get(indexA);
    T currB= b.get(indexB);
    if (currA.compareTo(currB) <= 0){
      res.add(currA);
      indexA++;
    }
    else {
      res.add(currB);
      indexB++;
    }
  }
  while (indexA < a.size()){
    res.add(a.get(indexA));
    indexA++;
  }

  return res;
}
```




```java
public List<T> merge(List<T> a, List<T> b) {
  List<T> res= new ArrayList<>(a.size() + b.size());
  int indexA= 0;
  int indexB= 0;
  while (indexA < a.size() && indexB < b.size()){
    T currA= a.get(indexA);
    T currB= b.get(indexB);
    if (currA.compareTo(currB) <= 0){
      res.add(currA);
      indexA++;
    }
    else {
      res.add(currB);
      indexB++;
    }
  }
  while (indexA < a.size()){
    res.add(a.get(indexA));
    indexA++;
  }
  while (indexB < b.size()){
    res.add(b.get(indexB));
    indexB++;
  }
  return res;
}
```




```java
public List<T> merge(List<T> a, List<T> b) {
  List<T> res= new ArrayList<>(a.size() + b.size());
  int indexA= 0;
  int indexB= 0;
  while (indexA < a.size() && indexB < b.size()){
    T currA= a.get(indexA);
    T currB= b.get(indexB);
    if (currA.compareTo(currB) <= 0){
      res.add(currA);
      indexA++;
    }
    else {
      res.add(currB);
      indexB++;
    }
  }
  while (indexA < a.size()){
    res.add(a.get(indexA));
    indexA++;
  }
  while (indexB < b.size()){
    res.add(b.get(indexB));
    indexB++;
  }
  return res;
}
```




```java
public List<T> merge(List<T> a, List<T> b) {
  List<T> res= new ArrayList<>(a.size() + b.size());
  int indexA= 0;
  int indexB= 0;
  while (indexA < a.size() && indexB < b.size()){
    T currA= a.get(indexA);
    T currB= b.get(indexB);
    if (currA.compareTo(currB) <= 0){
      res.add(currA);
      indexA++;
    }
    else {
      res.add(currB);
      indexB++;
    }
  }
  while (indexA < a.size()){
    res.add(a.get(indexA));
    indexA++;
  }
  while (indexB < b.size()){
    res.add(b.get(indexB));
    indexB++;
  }
  return res;
}
```


The code looks complete
Ideally, I should compile it and test it extensively.


I could randomly generate some lists of various sizes and with different kinds of content
But since this is an UPU exam...


I can not compile it,
let alone run tests.
I can only submit
it as is!

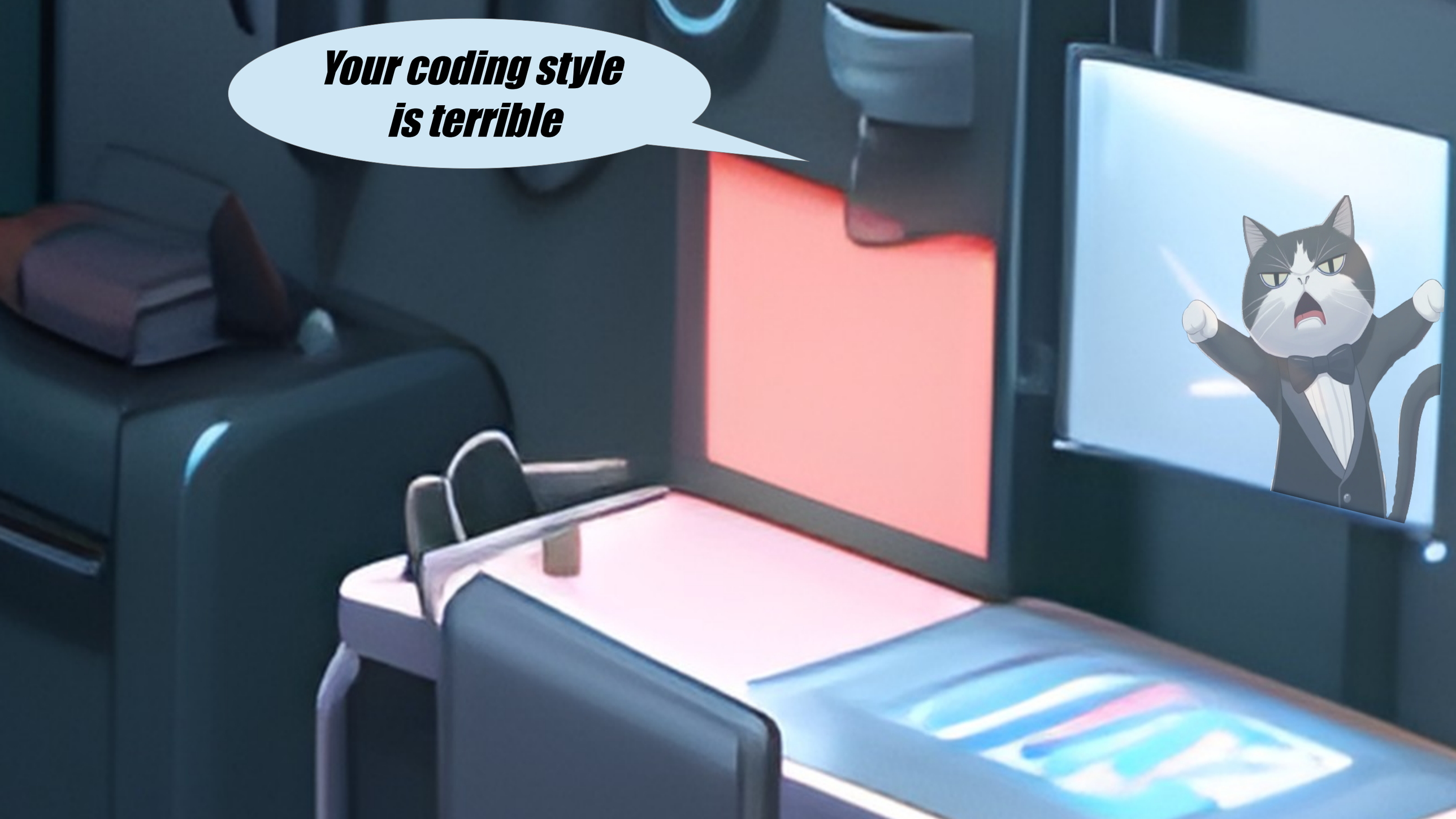

Submission in progress!

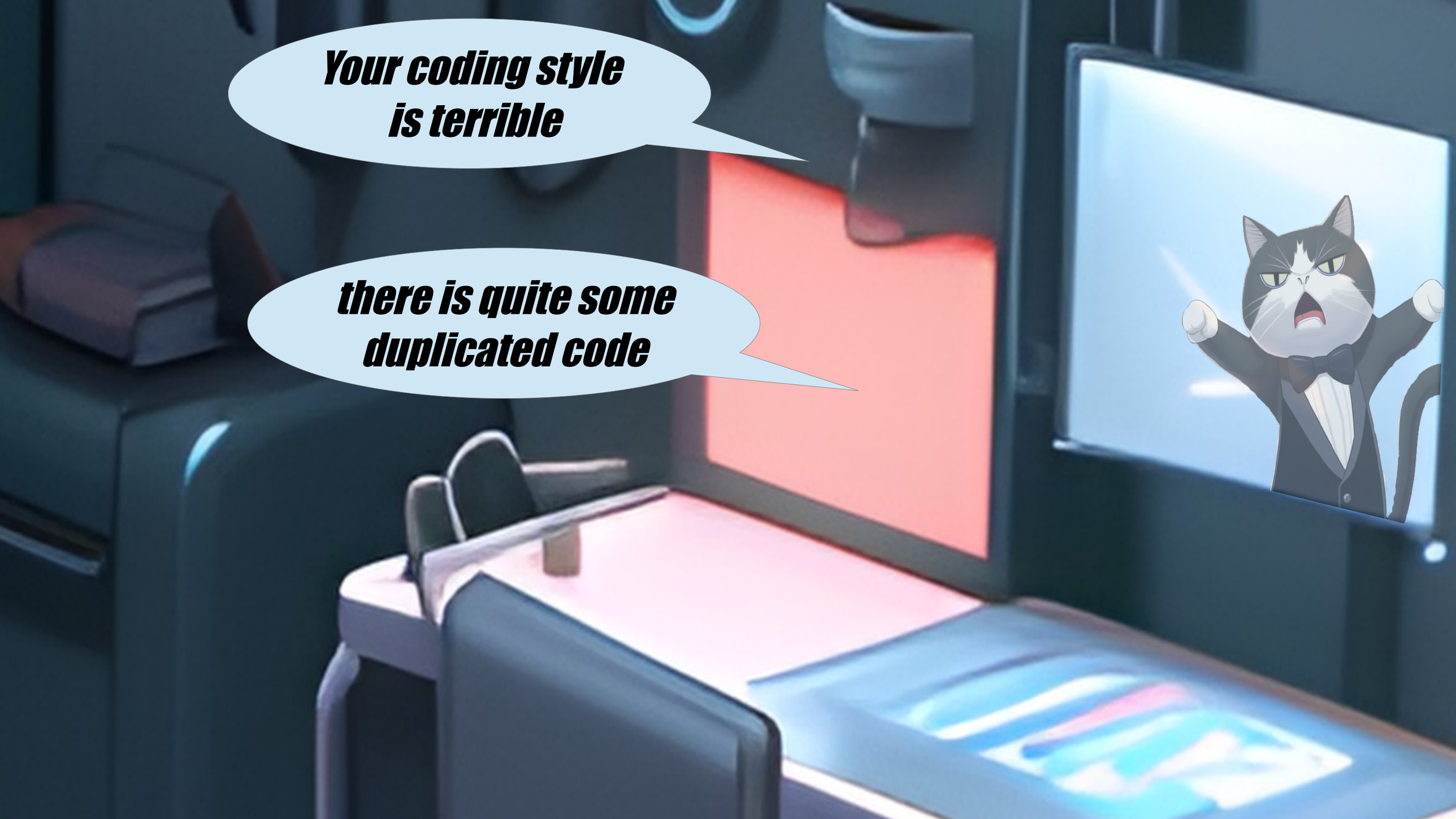

Submission in progress!
Oh, no!
Your submission

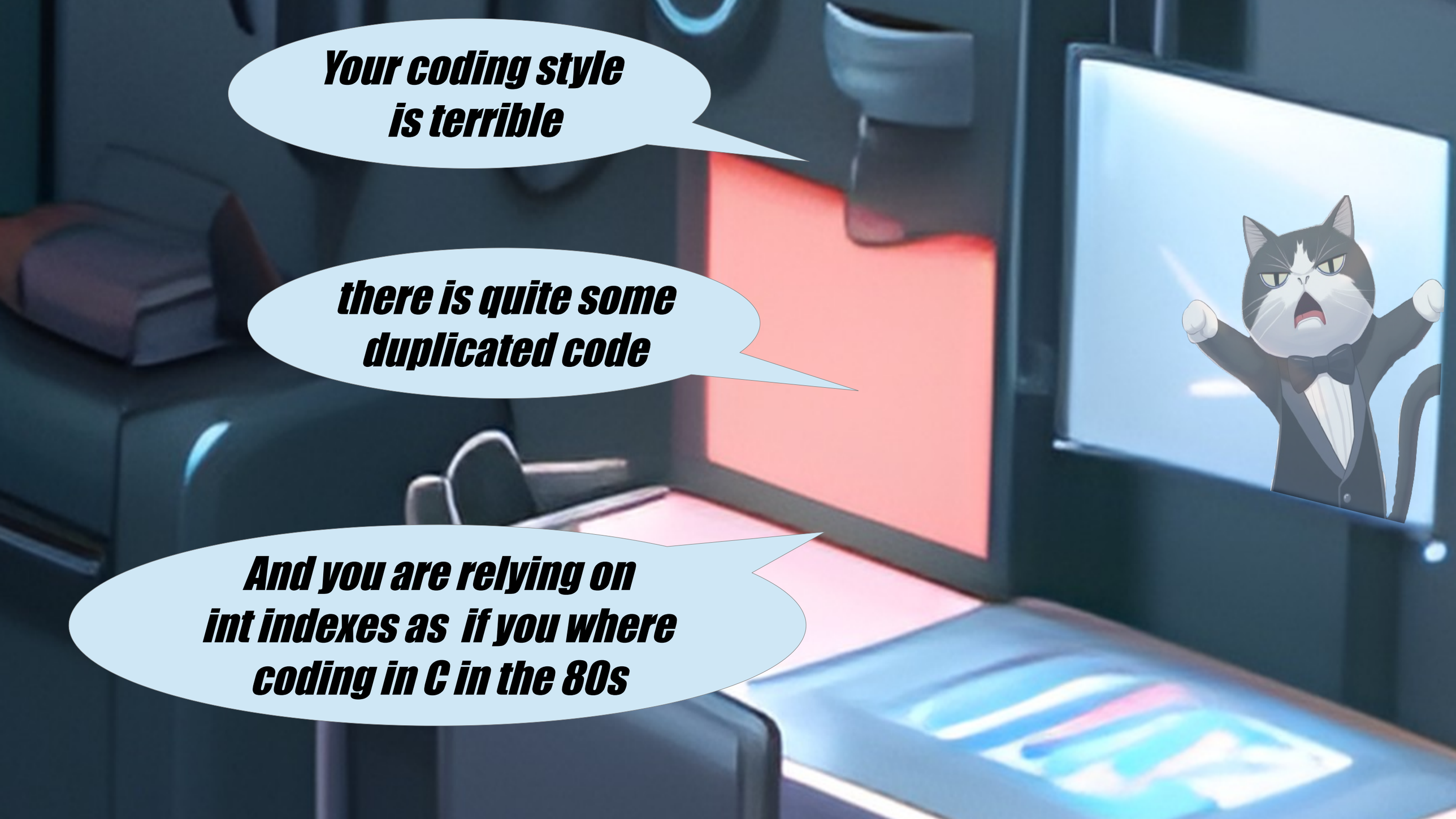

Submission in progress!
Oh, no! Your submission
It does not even met the minimum requirements to be marked!


To be clear:
it does compile, and
passes all our tests.


So,
why is it
refused then?


Your coding style is terrible


Your coding style is terrible
there is quite some duplicated code


Your coding style is terrible
there is quite some duplicated code
And you are relying on int indexes as if you where coding in C in the 80s


But not all is lost!


But not all is lost!
You can simply resubmit when you are ready


But not all is lost!
You can simply resubmit when you are ready
Every re submission caps your mark 10% down

At your first re submission any mark over 90% will be capped to 90%.

At your first re submission any mark over 90% will be capped to 90%.

At your second re submission It will be 80%

Pass grade is 50%, so between the two questions you can submit 5 more times

Pass grade is 50%, so between the two questions you can submit 5 more times

And still go to second year!


Just what I would expect from UPU


Just what I would expect from UPU
A first year student writing perfectly working code...


Just what I would expect from UPU
A first year student writing perfectly working code...
at the first attempt, does not even meet the minimum standards!


It is time to activate my power!