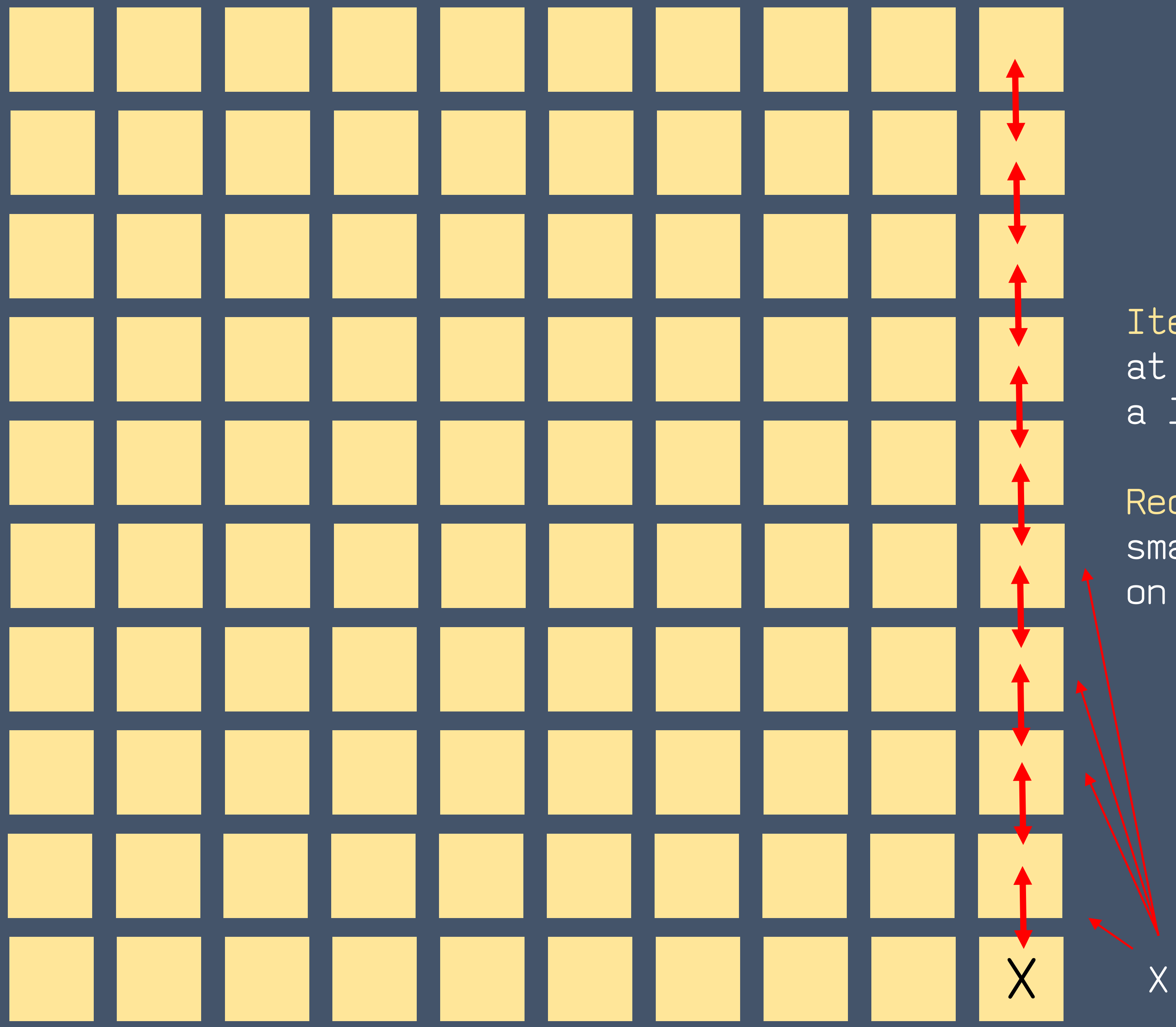
CMPSC 100 Computational Expression

Iteration

```
int i = 1;
do {
  System.out.print(i);
  i++;
} while (i <= 10);
```

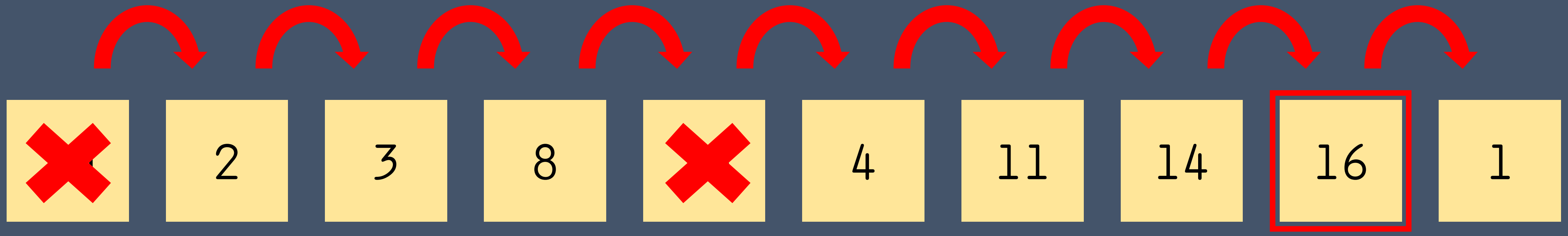
Recursion

```
public void castBallot(String name) {
   Candidate candidate = searchCandidates(name);
   if (candidate != null) {
     candidate.addVote();
     int index = this.form.indexOf(candidate);
     this.form.set(index,candidate);
   } else {
     System.out.println("WRITE-IN: " + name);
     candidate = new Candidate(name);
     addCandidate(candidate);
     castBallot(candidate.name);
   }
 }
```



Iteration tackles the whole task all at once, one instruction at a time in a list.

Recursion breaks the problem down into smaller pieces and concentrates only on solving them one call at a time.



Iterative problem: Let's look over all the numbers and keep track of the maximums we find along the way.

Recursive problem: Considering we have N numbers, let's look at the next one to see if it's larger; considering now we have N-1 numbers left and let's look at the next one to see if it's larger; considering we have N-2 numbers…

```
git pull download master

cd to 11-november/search
```

Information

- The RandomList.java file generates a list full of random numbers.
- In order to demonstrate our iterative and recursive solutions, we're going to examine:
  - Iterative "linear" search
  - Recursive "linear" search
- We will examine the code to see how each works.

```
public int linearSearch(int number) {
   for(int i = 0; i < this.list.length; i++){
     if (this.list[i] == number){
       return i;
     }
   }
   return -1;
 }
```

SEARCH.JAVA

```java
public int recursiveSearch(int number, int index) {
   if (index < 0 || this.list[index] == number) {
     return index;
   }
   return recursiveSearch(number, index - 1);
 }
```

SEARCH.JAVA

```java
public int rOL(int n, int i) {
  return (i < 0 || this.list[i] == n) ? i : rOL(n, i -  1);
}
```

SEARCH.JAVA

```java
do {
        …
        linear = search.linearSearch(number);
        recurse = search.recursiveSearch(number, search.list.length - 1);
        recurseOL = search.rOL(number, search.list.length - 1);
        …
} while (linear < 0 && recurse < 0);
```

RUNSEARCH.JAVA (EXCERPTED)

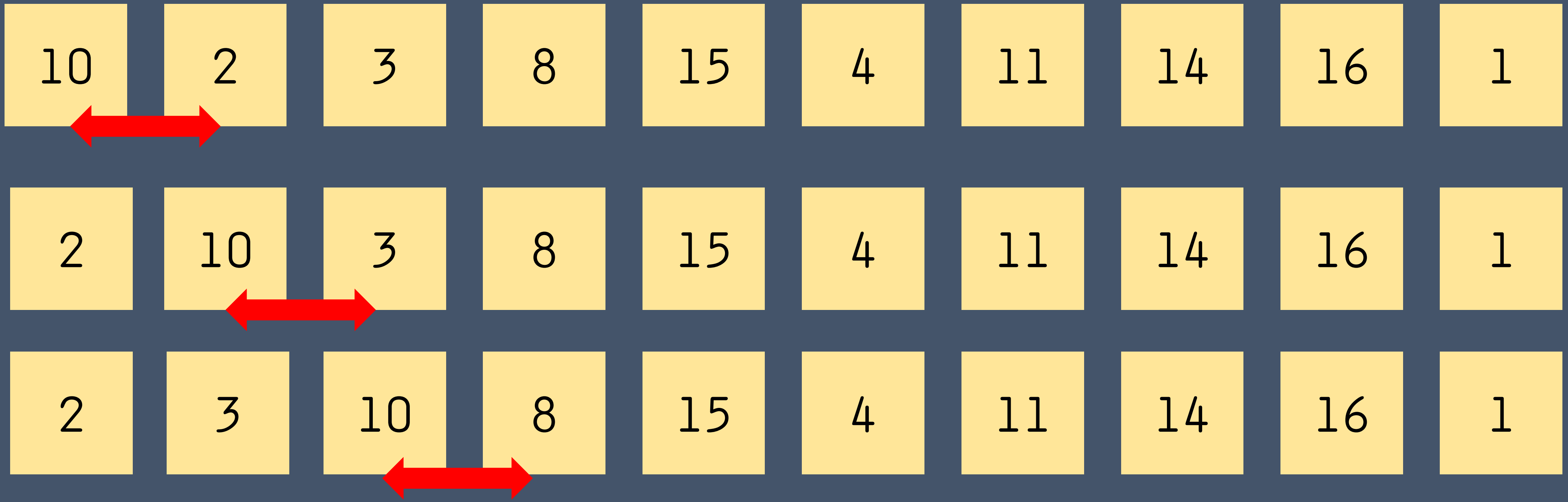
Let’s test it out to be sure:

```
gradle -q --console plain run
```

cd to `11-november/sort`

Iterative problem: Let's look through all of the numbers and swap where we need to.

Recursive problem: Considering we have N numbers, let's look at the next one to see if it's larger, then let's consider we have N-1 numbers left and let's look at the next one to see if it's larger, then let's consider we have N-2 numbers…



```
public int[] iterativeSort(int[] array) {
    for (int i = 0; i < array.length; i++) {
      for (int j = i + 1; j < array.length; j++) {
        if (array[i] > array[j]) {
          int temp = array[i];
          array[i] = array[j];
          array[j] = temp;
        }
      }
    printProgress(array); // “Tracing” code
    }
    return array;
  }
```

We use this in the exercise just to display what’s going on in the arraya

SORT.JAVA

```java
public int[] recursiveSort(int[] array, int index) {
    if(index <= 0) {
      return array;
    }
    for (int i = 0; i < array.length - 1; i++){
      if (array[i] > array[i + 1]) {
        int temp = array[i];
        array[i] = array[i + 1];
        array[i + 1] = temp;
      }
    }
    printProgress(); // “Tracing” code
    index--;
    return sort(array, index);
  }
```

We use this in the exercise just to display what’s going on in the arraya

SORT.JAVA

```
public static void main(String[] args) {
    Sort sort = new Sort();
    System.out.println("Iterative sort:");
    sort.iterativeSort(sort.list);
    sort = new Sort();
    System.out.println("Recursive sort:");
    sort.recursiveSort(sort.list, sort.list.length - 1);
  }
```

RUNSORT.JAVA (EXCERPTED)

Let’s test it out:

```
gradle run
```