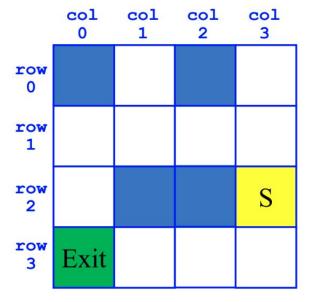


## SearchForTheExit

Initialize a Queue to hold Squares as we search
Mark starting square as visited
Enqueue starting square on Queue
While Queue is not empty
Dequeue square sq from Queue
Mark sq as visited
If sq is the Exit, we're done!
For each of square's unvisited neighbors (S, W, N, E):
Set neighbor's previous to sq
Enqueue neighbor to Queue



## SearchForTheExit

Initialize a Stack to hold Squares as we search
Mark starting square as visited
Push starting square on Stack
While Stack is not empty
Pop square sq from Stack
Mark sq as visited
If sq is the Exit, we're done!
For each of square's unvisited neighbors (S, W, N, E):
Set neighbor's previous to sq
Push neighbor to Stack

## **Abstract Data Types (In Java, Interfaces)**

## **Data Structures (In Java, implementing classes)**

```
class Sort1 {

public static boolean isSorted1(int[] arr) {
   for(int i = 0; i < arr.length - 1; i += 1) {
      if(arr[i] > arr[i + 1]) { return false; }
   }
   return true;
}

# isSorted1 in Python
def is_sorted1(lst):
   for i in range(0, len(lst) - 1):
      if lst[i] > lst[i + 1]: return False
   return True
```

```
class Sort2 {

public static boolean isSorted2(int[] arr) {
   for(int i = 0; i < arr.length; i += 1) {
     for(int j = i + 1; j < arr.length; j += 1) {
        if(arr[i] > arr[j]) { return false; }
     }
   }
   return true;
}
```

