Title: "coursework1_nkatz01_dsa"
Author: "NK"

1) Missing number problem for arrays with a fixed length

1.
8 times.

2.
51 times. 9*5+6

In general, assuming the array is sorted as in the question, assuming i=numberSearchedInTheArray and assuming there is 1 value missing in the array:

if i is found before the missing number, then the number of inner loops will be ((n-1)*(i-1)+i); if i is found after the missing number, the formula will be ((n-1)*(i-1)+(i-1)).

2)Missing number problem for arrays with an arbitrary length

1.
n iterations.

2.

((n-1)*(i-1)+i)

3.

((n-1)*(i-1)+(i-1))

3) Binary increment operation

```
int i = A.length - 1;

	while (i >= 0) {//starts counting from end of array backwards
		if ((A[i] != 1)) {//as soon as it finds a cell !=1, it overrides it with
1.
		A[i] = 1;
		break;
		}
	i--;// decreases counter
	}
	i++; //i increases in order to move rewind 1 cell back.

	while (i <= A.length - 1) {
	A[i] = 0; //overrides all the cells it had passed until the end of the array
in order to increase binary no only by 1
	i++;
```

```
	}
```

4) Processing a binary matrix

```
int i = 0;
int curj = 0;
int curR = 1;
int curR2 = 0;
int curj2 = 0;
boolean foundOnSameRow = false;
boolean foundOnSameCol = false;
boolean foundOnSameRowAndCol = false;

	for (i = 0; i < n * n; i++) { // tests each cell for first 1 up to end of A. n
being number of rows and cols in array.
			if (curj == n) { // when end of row is reached
				curj = 0;// restart column
				curR++;// increase row
			}
		curj = (n - (n - (curj + 1)));// considering col starts at 0, increase
col and also use for record keeping for
		// when/if 1 is found. (col no will be real col as opposed to java's -1
index
		// number)
		if (A[curR - 1][curj - 1] == 1)// when first 1 is found
			{
				curR2 = curR;// initialize a second row variable
				curj2 = curj;// ini a second col variable
				for (int l = i + 1; l < n * n; l++) {// do a second loop
to search for second 1, continuing from the index after
					// which first 1 was found
					if (curj2 == n) {// resets col and row increase for
second loop
						curj2 = 0;
						curR2++;
					}
					curj2 = (n - (n - (curj2 + 1)));// record col
					if (A[curR2 - 1][curj2 - 1] == 1) { // if both
						if (curR2 == curR && curj2 == curj)// if a 1
found on same row and also on same col
							foundOnSameRowAndCol = true;
						else if (curj2 == curj)// if only found on
same col
							foundOnSameCol = true;
						else if (curR2 == curR)// if only found on
same row
							foundOnSameRow = true;
						else
						;
					} // end of first if
				} // end of middle for
			} // end of second if
	} // end of outer for
```

```
if (foundOnSameRowAndCol || (foundOnSameCol && foundOnSameRow)) // if found on same
row and col (of first find) OR
	// if found separete instances of both
	System.out.println("foundOnSameRowAndCol==true ");
else if (foundOnSameRow)
	System.out.println("foundOnSameRow==true ");
else if (foundOnSameCol)
	System.out.println("foundOnSameCol==true ");
else
	System.out.println("notfoundNotOnSameColNorOnSameRow ");
```

5) A constant time algorithm

1.

1. Choose any two neighboring elements and compare them to see which value is bigger
2. Return that value.

2.

Because if the array is not pairwise distinct, you may pick up two neighboring elements that have the same value. For this reason, in part 1. of this question, they have to be neighbors, else you may pick two elements which are the same in which case, this would make the algorithm differ depending on the which elements you happened to come by.