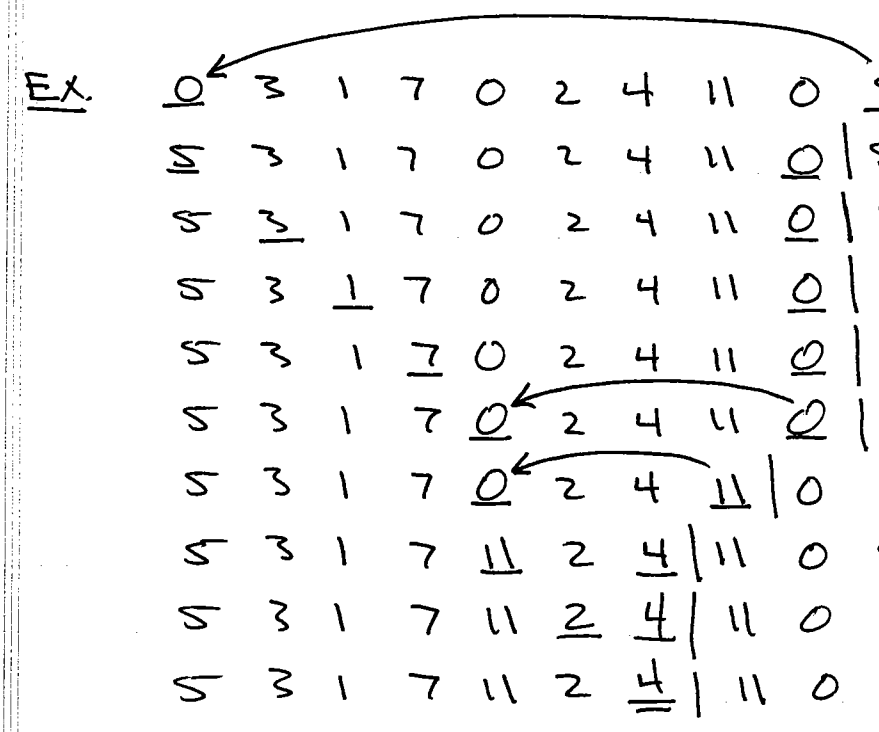
NOTE THAT FEWER COPY OPERATIONS ARE PERFORMED IN THIS ALGORITHM, BUT MUCH MORE MEMORY IS USED (i.e. THE NEW LIST.)

INPUT: $n \geq 1$, $a_1, \ldots, a_n$
OUTPUT: NEW LIST $b_1, \ldots, b_l$, l = LENGTH

1.) SET l TO n
2.) SET i TO 1
3.) SET j TO 1
4.) WHILE $i \leq n$
5.) IF $a_i \neq 0$
6.) SET b_j TO a_i
7.) SET j TO $j+1$
8.) ELSE
9.) SET l TO $(l-1)$
10.) SET i TO $i+1$
11.) END LOOP
12.) STOP

CONVERGING POINTERS ALGORITHM

MOVE YOUR LEFT FINGER ALONG LIST FROM LEFT TO RIGHT, RIGHT FINGER FROM RIGHT TO LEFT. LEFT FINGER SKIPS OVER NON ZERO VALUES, AND WHEN IT REACHES A ZERO, WE COPY WHATEVER IS AT THE RIGHT POSITION TO THE LEFT POSITION.



EX.											LENGTH
	0	3	1	7	0	2	4	11	0	5	10
	5	3	1	7	0	2	4	11	0	5	9
	5	3	1	7	0	2	4	11	0	5	
	5	3	1	7	0	2	4	11	0	5	
	5	3	1	7	0	2	4	11	0	5	
	5	3	1	7	0	2	4	11	0	5	
	5	3	1	7	0	2	4	11	0	5	8
	5	3	1	7	11	2	4	11	0	5	7
	5	3	1	7	11	2	4	11	0	5	
	5	3	1	7	11	2	4	11	0	5	

INPUT: $n \geq 1$, $a_1, \ldots, a_n$

OUTPUT: MODIFIES list.

1.) SET l TO n
2.) SET i TO 1
3.) SET j TO n
4.) WHILE $i < j$
5.) IF $a_i \neq 0$
6.) SET i TO $i+1$
7.) ELSE
8.) SET l TO $(l-1)$
9.) SET a_i TO a_j
10.) SET j TO $(j-1)$
11.) END loop
12.) IF $a_i = 0$
13.) SET l TO $(l-1)$
14.) STOP

OBSERVE THAT THIS ALGORITHM IS SPACE EFFICIENT IN THAT IT STORES THE RESULT IN THE ORIGINAL LIST, AND IT DOES MUCH LESS COPYING THAN SHUFFLE LEFT.

WE WILL RIGOROUSLY QUANTIFY THESE GAINS IN EFFICIENCY, TO SHOW THAT THEY REALLY EXIST.

MEASURING EFFICIENCY

WE'LL COME BACK TO THE DATA CLEAN-UP ALGORITHMS, FIRST RECALL SEQUENTIAL SEARCH

INPUT: $n \geq 1$, $a_1, \ldots, a_n$, TARGET
OUTPUT: THE FIRST INDEX i FOR WHICH a_i = TARGET

```
1.) SET i TO 1
2.) SET FOUND TO FALSE
3.) REPEAT UNTIL i > n OR FOUND
4.)       IF a_i = TARGET
5.)             SET FOUND TO TRUE
6.)       ELSE
7.)             SET i TO i+1
8.) END LOOP
9.) IF NOT FOUND
10.)      SET i TO 0
11.) PRINT i
12.) STOP
```

WE WILL TAKE AS OUR CENTRAL UNIT OF WORK THE COMPARISON OF ONE NUMERICAL VALUE WITH ANOTHER, THUS WE'LL COUNT THE NUMBER OF TIMES STEP 4 IS EXECUTED.

THE OTHER INSTRUCTIONS WILL BE CONSIDERED PERIPHERAL TASKS AND WILL NOT BE COUNTED, WHY?

NOTE STEPS 6, 8, 12 DON'T REALLY DO ANYTHING. STEPS 1, 2, 9, 11 ARE EXECUTED JUST ONCE, AND 5, 10 ARE EXECUTED AT MOST ONCE SO THEIR CONTRIBUTIONS WILL BE MINIMAL.

STEPS 3, 7 ARE EXECUTED (APPROXIMATELY) THE SAME NUMBER OF TIMES AS 4, SO PERHAPS WE SHOULD COUNT STEP 4 AND MULTIPLY BY 3 (OR SOME OTHER CONSTANT FACTOR.)

COMPARISON OF NUMERICAL VALUES MAY BE CONSIDERED MORE WORK THAN INCREMENTING i TO $i+1$. WE'LL SEE THAT ULTIMATELY IT DOESN'T MATTER WHAT FACTOR WE USE HERE.

NOTE:
WHAT MATTERS MOST IS NOT THE ACTUAL NUMBER OF OPERATIONS PERFORMED, BUT THE WAY THAT NUMBER SCALES WITH N - THE SIZE OF THE INPUT.
— MORE ON THIS LATER —

THUS WE COUNT STEP 4 IN WORST CASE, BEST CASE, AVERAGE CASE FOR LISTS OF A FIXED LENGTH N.

THE BEST CASE CLEARLY OCCURS WHEN TARGET IS THE FIRST ELEMENT IN THE LIST, IN WHICH CASE 4 IS EXECUTED ONCE.

THE WORST CASE OCCURS WHEN TARGET IS THE LAST ELEMENT IN THE LIST (OR NOT IN THE LIST). IN THIS CASE TARGET IS COMPARED WITH EVERY ELEMENT IN THE LIST — STEP 4 IS EXECUTED N TIMES.

FOR THE AVERAGE CASE ANALYSIS WE ASSUME (FOR SIMPLICITY) THAT TARGET IS IN THE LIST. AND IS EQUALLY LIKELY TO BE IN ANY POSITION.

SO ASSUME WE HAVE AN n ELEMENT LIST $a_1, a_2, \ldots, a_n$ AND A TARGET T WHICH OCCURS SOMEWHERE IN THE LIST.

NOTE THAT IF $T = a_1$, THEN 1 COMPARISON IS REQUIRED, IF $T = a_2$ THE 2 COMPARISONS, AND IN GENERAL IF $T = a_i$ THEN i COMPARISONS ARE PERFORMED BY SEQUENTIAL SEARCH.

THUS THE AVERAGE NUMBER OF COMPARISONS IS

$$\frac{1+2+3+\cdots+n}{n} = \frac{\frac{n(n+1)}{2}}{n} = \frac{n+1}{2}.$$

EXCERCISE:
FIND THE AVERAGE NUMBER OF COMPARISONS WHEN THE POSSIBILITY THAT T IS NOT IN THE LIST IS ALLOWED. ASSUME THAT T IS EQUALLY LIKELY TO BE IN THE LIST AS NOT.

ANSWER: $\frac{3n+1}{4}$

THE NUMBER OF COMPARISONS PERFORMED BY SEQUENTIAL SEARCH ON AN n ELEMENT LIST CONTAINING THE TARGET IS

BEST CASE	WORST CASE	AVERAGE CASE
1	n	$\frac{n+1}{2}$

WE IGNORED PERIPHERAL TASKS (STEPS 3, 7) IN OUR ANALYSIS. HAD WE INCLUDED THEM, WE WOULD HAVE OBTAINED AN OPERATIONS (IN WORST CASE) FOR SOME FIXED CONSTANT a (DEPENDING ON THE RELATIVE VALUES OF DIFFERENT TYPES OF OPERATIONS,) (ACTUALLY $an+b$)

CONSIDER ANOTHER SIMPLE PROBLEM WITH A SIMPLE SOLUTION:

GIVEN A TABLE OF NUMBERS WITH n ROWS AND n COLUMNS (i.e. A SQUARE MATRIX)

$$\begin{matrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{matrix} ,$$

PRINT OUT ALL VALUES IN THE TABLE.

1.) SET i TO 1 ~~AND~~ (j TO 1)
2.) WHILE $i \leq n$
3.) WHILE $j \leq n$
4.) PRINT a_{ij}
5.) SET j TO j+1
6.) END loop
7.) SET i TO i+1
8.) END loop
9.) STOP.

LET THE UNIT OF WORK BE PRINTING A NUMERICAL VALUE. IGNORING PERIPHERAL INSTRUCTIONS (LIKE INCREMENTING i AND j) WE COUNT n^2 OPERATIONS. (NOTE BEST CASE, WORST CASE, AND AVERAGE CASE ARE ALL THE SAME.)

IF WE DO COUNT PERIPHERAL INSTRUCTIONS WE COUNT AGAIN bn^2 OPERATIONS FOR SOME FIXED CONSTANT b.

OBSERVE THAT ANY ALGORITHM WHICH DOES bn^2 OPERATIONS IS WORSE THAN ANY WHICH DOES an OPERATIONS, NO MATTER WHAT a AND b ARE!