Consider an array A whose elements are all either "black" or "white". The following algorithm rearranges the elements of A so that every black appears before every white.

```
1: j = 0
2: i = 0
3: while i < len(A) do
4:     if A[i] == "black" then
5:         A[i], A[j] = A[j], A[i]
6:         j = j + 1
7:     end if
8:     i = i + 1
9: end while
```

1. Write down preconditions and postconditions for this algorithm, based on the description above.

Ans: Preconditions: A is an array, all of whose elements are either "black" or "white".

[Shouldn't we also say that A is not empty, i.e., $len(A) \geq 1$? The loop does not execute at all if $len(A) = 0$, so the code works in this case because in the final array, it is vacuously true that every black precedes every white. Since the algorithm works for $len(A) = 0$, we should not arbitrarily place a restriction that $len(A) > 0$ in the precondition.]

Postconditions: The elements of A have been rearranged so that every "black" appears at a smaller index than every "white".
□

2. Prove that the algorithm is partially correct– this will involve finding a suitable loop invariant.

Ans: There are two aspects to finding a good loop invariant:

a) It must allow us to derive the postcondition at the end of the loop. So we can start from the postcondition and try to think about the ways in which the loop is making progress toward this goal.

b) It must describe the operation of the loop. So we can trace through the loop and observe how the values of the variables change with respect to each other.

Trace through two extreme cases of input: [b,b,w,w] and [w,w,b,b], to get a feeling for how the algorithm works.

Let's investigate the states of things partway through the loop and see if we can use them to come up with loop invariant. values in $A[0..i-1]$ have been rearranged so that every black appears before every white– this is just like the postcondition except just for an initial segment of the array. If we proved this as our loop invariant, it would give us the postcondition for free by the end of the loop– as long as we could show that $i = len(A)$ at the end.

This is easy to do if we add the condition that $i \leq len(A)$ as part of the loop invariant: then, at the end of the loop, we can conclude $i \leq len(A)$ (from the loop invariant) and $i \geq len(A)$ (because the loop condition is false when the loop stops), and this is equivalent to saying $i = len(A)$.

However, trying to prove this directly won't quite work, because it gives us no information about j that we could use to reason about what happens during one iteration. This means we need to add more parts to the loop invariant– we need to state more precisely everything we know about the variables i and j and how they relate to the values in $A[0..i-1]$.

Regarding values of i and j, it seems plausible that $0 \leq j \leq i \leq len(A)$. Regarding values in $A[0..i-1]$, we have two cases: either $j = i$, or $j < i$ and $A[0..j-1]$ = "black", $A[j..i-1]$ = "white".

Now, let's put this altogether:
Loop invariant (LI): $0 \leq j \leq i \leq len(A)$ and values in $A[0..i-1]$ have been rearranged so that $A[0..j-1]$ = "black", and either $j = i$ or $A[j..i-1]$ = "white".

Partial Correctness:

[Assume that the precondition holds, and the algorithm executes. When the loop stops, we know that $i \geq len(A)$ (negation of the loop condition) and that LI holds (by definition of a loop invariant). Then $i = len(A)$ (because $i \leq len(A)$ from LI and $i \geq len(A)$ from the loop condition). Then the values in $A[0..i-1] = A[0..len(A)-1] = A$ have been rearranged so that $A[0..j-1] =$ "black", and either $j = i = len(A)$ (in which case every value in A is black) or $A[j..i-1] = A[j..len(A)-1] =$ "white". In other words, the values have been rearranged so that every "black" appears before every "white", which is what we wanted.]

Let's prove the loop invariant by induction on the number of iterations:

Base Case: $j_0 = 0$, $i_0 = 0$ (from the initialization) and hence $0 \leq j_0 \leq i_0 \leq len(A)$ and there are no values in $A[0..i_0 - 1]$ (empty sublist) so the rest of LI is vacuously true.

Induction Step: Assume $k >= 0$ and LI holds after k^{th} iteration (IH). Assume the loop executes one more iteration. From the loop condition, this means $i_k < len(A)$. Either $A[i_k] =$ "black" or $A[i_k] =$ "white".

Case 1: If $A[i_k] =$ "black", then either $j_k = i_k$ or $j_k < i_k$.

Subcase A: If $j_k = i_k$, then $A[0..j_k - 1] = A[0..i_k - 1] =$ "black", and $A[i_k] =$ "black". The algorithm swaps $A[i_k]$ and $A[j_k]$ (which has no effect) and sets $j_{k+1} = j_k + 1$, $i_{k+1} = i_k + 1$. Then, $0 \leq j_{k+1} \leq i_{k+1} \leq len(A)$ because $j_k = i_k < len(A)$ implies $j_{k+1} = i_{k+1} = i_k + 1 \leq len(A)$. Also, $A[0..i_{k+1} - 1] = A[0..i_k] =$ "black", and $j_{k+1} = i_{k+1}$. Hence, LI holds.

Subcase B: If $j_k < i_k$, then $A[0..j_k - 1] =$ "black" and $A[j_k..i_k - 1] =$ "white". The algorithm swaps $A[i_k]$ and $A[j_k]$– after this, $A[j_k] =$ "black" and $A[i_k] =$ "white" and sets $j_{k+1} = j_k + 1$. Then, $A[0..j_{k+1} - 1] = A[0..j_k] =$ "black", and since $i_{k+1} = i_k + 1$ and $j_k < i_k$, $j_{k+1} < i_{k+1}$ and $A[j_{k+1}..i_{k+1} - 1] = A[j_k + 1..i_k] =$ "white". Hence, LI holds. In both subcases, LI holds.

Case 2: If $A[i_k] =$ "white", then the algorithm simply sets $i_{k+1} = i_k + 1$ and j_{k+1} stays equal to j_k. Then $0 \leq j_{k+1} \leq i_{k+1} \leq len(A)$ and $A[0..j_{k+1} - 1] = A[0..j_k - 1] =$ "black", and $j_{k+1} < i_{k+1}$ and $A[j_{k+1}..i_{k+1} - 1] = A[j_k..i_k] =$ "white". Hence, LI holds.

By induction, LI holds at the end of every iteration.

□

3. Prove that the algorithm terminates.

Ans:

Intuitively: i starts at 0 and increases by 1 every time through the loop, and the loop stops as soon as $i \geq len(A)$, which will happen after exactly $len(A)$ many iterations.

Formally: The expression $E = len(A) - i$ is a natural number (by the loop invariant, $i \leq len(A)$) and strictly decreases from one iteration to the next (because of the line $i = i+1$). Hence, the sequence of values $E_0, E_1, E_2, \cdots$ is a strictly decreasing sequence of natural numbers, which must be finite. Therefore, the loop terminates. □