Counting Sort Proof of Correctness

Josh Cohen

November 17, 2020

We define counting sort as the following pseudocode:

```
function countingSort<A>(a: A[], k : int, key : A -> int) : A[] {
    b = new int[k]; //initialized to 0
    for(int i = 0; i < a.Length; i++) {
        b[key(a[i])] += 1;
    }
    b[0] -= 1; //for 1 indexing
    for(int i = 1; i < k; i++) {
        b[i] += b[i-1];
    }
    c = new A[a.Length];
    for(int i = a.Length - 1; i >= 0; i--) {
        c[b[key(a[i])]] = a[i];
        b[key(a[i])] -= 1;
    }
    return c;
}
```

As expected for counting sort, we require that, for all x in a, $0 \leq key(x) \leq k$.
For talking about slices of arrays, we will use the notation $a[x..y]$, consisting of $a[x], a[x+1], ..., a[y-1]$.
We now define the following functions:

1. $numLt(a, x) :=$ the number of elements in array a with key smaller than x.
2. $numEq(a, x) :=$ the number of elements in array a with key equal to x.
3. $numLeq(a, x) := numLt(a, x) + numEq(a, x)$.
4. $position(x, i, a) := numLt(x, a) + numEq(x, a[0..i + 1]) - 1$.
   This is meant to represent the position of an element in a sorted, stable array consisting of elements of a. To see this, consider $position(key(a[i]), i, a)$. How many elements occur to the left of $a[i]$ in the sorted, stable array? All of the elements with smaller keys must occur before $a[i]$, as well as all of the elements with equal keys in the array $a[0..i + 1]$. We subtract 1 because the array is 0-indexed.
5. $filter(a, f)$ filters the array a by predicate f - it keeps all elements in a that satisfy f, in order.

Now we begin the proof of correctness.

Lemma 1. *After the first loop, for all i, $b[i] = numEq(a, i)$.*

Proof. The proof is easy; for each element x in a with $key(x) = i$, we increment $b[i]$ exactly once. □

Lemma 2. *After the second loop, $b[i] = numLeq(a, i) - 1$.*

Proof. We prove the claim by induction on i. When $i = 0$, since all keys are nonnegative, $numLt(a, 0) = 0$, so $numLeq(a, i) = numEq(a, i)$. Since we subtract 1 from $b[0]$, the claim holds.
Now assume the claim is true for $i - 1$. Then $b[i] = b[i] + b[i - 1] = numEq(a, i) + numLeq(a, i - 1) - 1$. Since $numLeq(a, i - 1) = numLt(a, i)$, the claim holds. □

Now we need to handle the third loop. We will let c_j denote the portion of c that has been filled in when $i = j$. We will use the following loop invariants:

1. $a[(i+1)..a.Length]$ and c_i are permutations of each other.
2. For all $0 \le j \le k$, $b[j] = position(j, i, a)$.
3. For all j such that $c_i[j]$ exists (ie, position j has been filled in), there exists a k such that $i < k < a.Length$ and $c_i[j] = a[k]$ and $j = position(key(a[k]), k, a)$.
4. For all integers x, $filter(a[i+1..a.Length], y \to y = x) = filter(c_i, y \to y = x)$ (ie, the portion of a considered so far and the portion of c completed so far are stable with respect to each other).

Lemma 3. *All invariants hold before the third loop.*

Proof. Since $i = a.Length - 1$, $a[i+1..a.Length]$ is empty, as is c_i. Thus, the only nontrivial invariant is invariant 2.
Since $i = a.Length - 1$, $position(j, i, a) = numLt(j, a) + numEq(j, a[0..a.Length]) - 1 = numLt(j, a) + numEq(j, a) - 1 = numEq(j, a) - 1$. This is true by Lemma 2. □

Now, we will prove that each invariant is preserved during the loop by first proving the following lemmas:

Lemma 4. *position is injective in the following sense: for any i, j, if $position(key(a[i]), i, a) = position(key(a[j]), j, a)$, then $i = j$.*

Proof. Suppose not. We consider 2 cases:

1. If $key(a[i]) = key(a[j])$, let $k = key(a[i])$. Then

$$\begin{aligned} position(key(a[i]), i, a) &= numLt(k, a) + numEq(k, a[0..i+1] - 1 \\ position(key(a[j]), j, a) &= numLt(k, a) + numEq(k, a[0..j+1]) \end{aligned}$$

   WLOG, suppose $i < j$. Then $numEq(k, a[0..i+1]) + numEq(k, a[i+1..j+1]) = numEq(k, a[0..j+1])$. Since $i < j$ and $a[j] = k$, $numEq(k, a[i+1..j+1]) > 0$. This contradicts the fact that the positions were equal.

2. If $key(a[i]) \ne key(a[j])$, WLOG assume $key(a[i]) < key(a[j])$. We have the following bounds straight from the definition of *position*:

$$\begin{aligned} position(key(a[i]), i, a) &\le numLeq(key(a[i]), a) - 1 \\ numLt(key(a[j]), a) &\le position(key(a[j]), j, a) \end{aligned}$$

   Note that we can write $numLt(key(a[j]), a) = numLeq(key(a[j]) - 1, a)$ (since we are working with integers). Now $key(a[i]) \le key(a[j]) - 1$, so

$$\begin{aligned} position(key(a[i]), i, a) &\le numLeq(key(a[i]), a) - 1 \\ &\le numLeq(key(a[j] - 1), a) - 1 \\ &= numLt(key(a[j]), a) - 1 \\ &< numLt(key(a[j]), a) \\ &\le position(key(a[j]), j, a) \end{aligned}$$

   This again contradicts the position equality.

□

Lemma 5. *At the beginning of each iteration of the loop, $c[b[(key(a[i])]]$ has not yet been filled in.*

Proof. Suppose it had, then by invariant 3, there is some k such that $i < k < a.Length$, $c_i[b[key(a[i])]] = a[k]$, and $b[key(a[i])] = position(key(a[k]), k, a)$. But by invariant 2, $b[key(a[i])] = position(key(a[i]), i, a)$. By Lemma 4, then, $i = k$, a contradiction. □

Lemma 6. *At the beginning of each iteration of the loop, for all $0 \leq j < b[key(a[i])]$, if $c_i[j]$ exists (ie, position j has been filled), then $key(c_i[j]) \neq key(a[i])$. In other words, we fill in all the values with $key = key(a[i])$ from right to left.*

Proof. Suppose not, so there is some $j < b[key(a[i])]$ where $key(c_i[j]) = key(a[i])$. By invariant 3, there is some k with $i < k < a.Length$, $c_i[j] = a[k]$, and $j = position(key(a[k]), k, a)$. By assumption, $key(a[k]) = key(a[i])$. By invariant 2, $b[key(a[i])] = position(key(a[i]), i, a)$. Thus, we get that $position(key(a[i]), k, a) < position(key(a[i]), i, a)$. Since $i < k$, this is a contradiction (ie, we cannot decrase the number of equal elements in the array by extending the range we are considering). □

Now, we can prove that each invariant is preserved.

Lemma 7. *Each invariant is preserved by the body of the third loop.*

Proof. Let b_{old} represent b at the beginning of the current iteration of the loop, and b_{new} represent b after the body of the loop.
By Lemma 5, c_{i-1} is the same as c_i, except that we fill in $c_{i-1}[b_{old}[key(a[i])]]$ with $a[i]$. All other positions are the same. Also note that $a[i..a.Length]$ is just $a[i+1..a.Length]$ with $a[i]$ added at beginning. Finally, b_{new} is the same as b_{old} except that the only change to b is that $b_new[key(a[i])] = b_old(key(a[i])] - 1$.

1. By above, we add $a[i]$ to both sides, so they are still permutations of each other.

2. We consider 2 cases:

   (a) If $j = key(a[i])$, then $b_{new}[j] = b_{old}[j] - 1 - position(j, i, a) - 1$. But $position(j, i-1, a) = position(j, ia) - 1$, since $key(a[i])$ appears at position $a[i]$.

   (b) If $j \neq key(a[i])$, then $position(j, i, a) = position(j, i-1, a)$, since the number of equal keys in the given range did not change; we just removed an unequal element.

   Thus, invariant 2 is preserved.

3. For all $j \neq key(a[i])$, the claim follows from the invariant. If $j = b_{old}[key(a[i])]$, we have $i - 1 < i < a.Length$, $c_{i-1}[j] = a[i]$, and by invariant 2, $j = position(key(a[i]), i, a)$, proving the claim.

4. If $x \neq key(a[i])$, then, since we added $a[i]$ to both sides, we did not change anything with respect to stability, so the claim continues to hold.
   If $x = key(a[i])$, then by Lemma 6, there were no elements with equal keys before index $b_old[key(a[i])]$, so we added $a[i]$ to the beginning of the filtered list. Thus, if we consider the list of elements with $key = key(a[i])$ in c_i, we have added $a[i]$ before all of the others. Likewise, we added $a[i]$ to the beginning of $a[i+1..a.Length]$, so the claim holds.

□

Therefore, we know that, once the third loop exits, the following are true (define $c_{ret} := c_{-1}$ - the array c when the loop exits):

1. a and c_{ret} are permutations.

2. a and c_{ret} are stable with respect to each other.

3. For every index j where $c_{ret}[j]$ exists, there exists a k such that $0 \leq k < a.Length$ and $c_{ret}[j] = a[k]$ and $j = position(key(a[k]), k, a)$.

The first condition implies that $|a| = |c_{ret}|$ and thus, all indices of c_{ret} are filled in (since we created c such that $c.Length = a.Length$). Thus, the third condition is really equivalent to:

- For every $0 \leq j < c.Length$, there exists a k such that $0 \leq k < a.Length$ and $c_{ret}[j] = a[k]$ and $j = position(key(a[k]), k, a)$.

All that remains is to prove that the above condition implies sortedness. We do so in the following lemma:

Theorem 8. *c_{ret} is sorted.*

Proof. We prove this in two parts: first, we claim that, for every $0 \leq j < c_{ret}.Length$, $numLt(key(c_{ret}[j]), c_{ret}) \leq numLeq(key(c_{ret}[j]), c_{ret}) - 1$.
To prove this, we consider the k such that $0 \leq k < a.Length$, $c_{ret}[j] = a[k]$, and $j = position(key(a[k]), k, a)$.
Let $x = key(a[k]) = key(c_{ret}[j])$. Again, we know that:

$$\begin{aligned} numLt(x, a) \leq position(x, k, a) &\leq numLeq(x, a) - 1 \\ numLt(x, a) \leq j &\leq numLeq(x, a) - 1 \end{aligned}$$

It is clear by definition, that $numLt$ and $numLeq$ are preserved over permutations. Thus,

$$numLt(x, c_{ret}) \leq j \leq numLeq(x, c_{ret}) - 1$$

which is what we wanted to show.
Now we prove that the above condition implies sortedness. We want to prove that, for all $i \leq j$, $key(c_{ret}[i]) \leq key(c_{ret}[j])$.
Suppose not, so $key(c_{ret}[j]) < key(c_{ret}[i])$.
Then, using a similar idea as in Lemma 4,

$$\begin{aligned} j &\leq numLeq(key(c_{ret}[j]), c_{ret}) - 1 \\ &< numLeq(key(c_{ret}[j]), c_{ret}) \\ &\leq numLeq(key(c_{ret}[i] - 1), c_{ret}) \\ &= numLt(key(c_{ret}[i]), c_{ret}) \\ &\leq i \end{aligned}$$

So $j < i$, a contradiction. Thus, c_{ret} is sorted. $\square$

We have shown, therefore, that the returned array is a permutation of the input, is sorted, and is stable with respect to the input.