
Hint: find minimum and maximum numbers, say $a$ and $b$. Define $g = (b - a)/(n - 1)$. Clearly, the maximum gap has $A[j] - A[i] \geq g$. Split the numbers in the array into intervals of the form $[a + kg, a + (k + 1)g)$. The maximum gap cannot be within one of these intervals. This method is inspired by counting sort because you map $A[m]$ into the interval number $\lfloor (A[m] - a)/g \rfloor$, and we can sort them according to these interval numbers.

Answer. We can compute the minimum and maximum of entries in $A$, $Amin$ and $Amax$. If $Amin = Amax$ then the largest gap is 0. Otherwise we can observe that the largest gap is at least $(Amax - Amin)/(n - 1)$. We can split interval $[Amin, Amax]$ into $n$ intervals of equal length, for each entry in $A$ we can finds its interval $i = (A[j] - Amin)/(Amax - Amin) \times n - 1)$. For each interval we can find the count of entries of $A$, the maximum such entry and the minimum. Then we can check, for every two consecutive non-empty intervals (counts larger than 0), the minimum of the latter minus the maximum of the former.

This is correct because inside an interval the differences are below the interval length, hence, below the largest gap. The running time is linear because we do constant work with each entry of $A$ and with each interval.
Problem 2. Radix sort. In a railroad yard, we have three tracks labeled $B$, $C$ and $D$, which are connected by a switch to track $A$:

A railroad car can move from $A$ to $B$, $C$ or $D$ by rolling down on a gentle slope. We can release and engage the brakes in a car by remote control, and we can operate the switch to direct the car to $B$, $C$ or $D$.

![Diagram of railroad tracks A, B, C, and D with a switch]

a. Suppose that we have some number of cars on tracks $B$, $C$ and $D$, and that we have one locomotive on track $A$. How can we move all of the cars to $A$ using the one locomotive? The locomotive can push or pull cars, and you can connect cars to each other or disconnect them.

b. Now suppose that we have a train consisting of a locomotive followed by a string of cars on track $A$, with the train facing away from the switch (that is, the locomotive is farther from the switch than the cars are). The cars have destinations $1, 2, \ldots, 9$ in some random order. (Note that many cars may have the same destination.) We want to rearrange the cars in the train so that cars to 1 are in front, followed by cars to 2, then to 3, and so forth. Describe a method for sorting the cars in this fashion using tracks $A$, $B$, $C$ and $D$ efficiently; that is, you want the total number of times that cars cross the switch to be as small as you can make it.

Hint: it suffices to roll cars from $A$ to the other tracks and return them to $A$, repeating this roll-return process a suitable number of times, if you do it correctly.

Answer. We apply radix sort, in this case, we represent 9 destinations as 9 two-letter combinations, BB, BC, BD, CB, CC, CD, DB, DC, DD. We first send cars down to tracks $B$, $C$, $D$ according to the second letters, and then the locomotive goes to from track $A$ to track $B$, collects the cars, back to $A$ and then to $C$, collects the cars and again with track $D$.

Next we send cards down again, this time according to first letters. For example, in track $B$ the cars BD will come first (from the end of the train), BC next and finally, BB. Thus after locomotive collects the cars as before, BB cars will be closes to the locomotive, DD will be the farthest and in general, the order will be correct.

One can see that each car rolls down the switch twice, and then they move back and forth several times when they are collected by the locomotive (those that moved initially to track $A$ have to make B-A-C-D-A movement during the collection, those that moved to track $C$ make C-D-A movement and those that move to track $D$ make D-A movement, but this is really the single collection effort by the locomotive).
Problem 3. Radix sort. You are given an array of characters $C[N]$ and an array $A[n+1]$ such that $0 = A[0] \leq A[1] < \cdots \leq A[n+1] = N$. String $s_1$ is the sequence of characters in the array sector $(C, A[i], A[i+1])$. We define lexicographic order of strings:

- if $\lambda$ is an empty string and $s$ any other string then $\lambda < s$;
- if $c, d$ are characters, $s, t$ are strings and $c < d$ then $cs < ct$;
- if $c$ is a character, $s, t$ are strings and $s < t$ then $cs < ct$.

Your task is to sort the strings, e.g., initialize

```plaintext
for (i = 0; i < n; i++)
    S[i] = i;
```

and then to permute $S[n]$ in such a way that $s_{S[i]} \leq s_{S[i+1]}$ for $i = 0, 1, \ldots, n-2$. You should use $O(N)$ time.

Hint: start by sorting strings by their length. Let $L$ be the maximum length. You should use Radix sort, but when you are performing census and distribution according to $k$-th characters of the strings you cannot inspect strings that do not have that characters, i.e. that are shorter than $k$.

Answer. Radix sort runs rounds of counting sort described in the textbook as DistributionCounting on page 253, in which “digits” of the values being sorted are used, from the least significant to most.

In case of strings described here, string $i$ has length $A[i+1] - A[i]$, so its $k$-th character/digit is $C[A[i]+k]$ if $A[i]+k < A[i+1]$ and “empty” otherwise, and when we compare characters, “empty” is smaller than a proper character (because empty strings is before other strings). Thus we could use array with a position for each possible value of a character, plus a extra value of “empty”, e.g.

```plaintext
int DE[257], *D;
D = DE+1;
```

and we can count the number of “empty” characters using counter $D[-1]$. The only problem with that is that we unnecessarily, and possibly, very frequently, check the values of “empty” characters. If the maximum length of a string is $L$, and a string has length $a$, then we do not have to look at this string during the first $L-a$ rounds. This idea can be elaborated as follows:

```plaintext
for (L = i = 0; i < n; i++)
    if (A[i+1]-A[i] > L)
        L = A[i+1]-A[i];
    int *DL = (int*) malloc((n+L+1)*sizeof(int));
    D += L+1;
```

Now we can do the first distribution by length, so strings of length $a$ are counted using $DL[a]$ and after the distribution they are at positions from $S[DL[a]]$ to $S[DL[a+1] - 1]$. Then we perform rounds that use $k$-th positions in strings

```plaintext
for (k = L; k >= 0; k--)
    ....
```

After such round, strings of length $k$ or more will be properly sorted according to the
characters with number \( k \) and more, and will be at positions from \( S[DL[k]] \) to \( S[n-1] \). In the next round we will count and distribute according to character number \( k-1 \), and thus we will also look at strings of length \( k-1 \), which are at positions from \( S[DL[k-1]] \) to \( S[DL[k]-1] \), exactly as they should be: according to characters from \( k \) onwards they are empty strings, so they should be before other strings. Thus in the next round we count and distribute strings starting from position \( S[DL[k-1]] \), and in general, in round \( k \) we count and distribute starting from \( S[DL[k]] \).

**Problem 7** from page 271. To check if all elements in a list are distinct, we can initialize empty dictionary, and then for each list element we first check if it is already present — if yes, we discovered a duplication — and if not, we insert this element. If we do not discover a duplication, we know that all elements are distinct.