Solutions for Data Structures and Algorithms Spring 2023 — Problem Sets

By Dmitriy Okoneshnikov, B22-DSAI-04

February 21-22, 2023

Week 6. Problem set

1. Consider a modification of MERGE-SORT algorithm [Cormen, Section 2.3] that stops recursion when the size of subarray becomes less than or equal to k. For arrays of size $\leq k$, the modified algorithm performs BUBBLE-SORT. Answer the following questions about the modified algorithm:

   (a) What is the worst case time complexity in terms of n and k?
   Answer. $\Theta(\frac{n}{k} \log \frac{n}{k} + nk)$

   (b) What is the best case time complexity in terms of n and k?
   Answer. $\Theta(\frac{n}{k} \log \frac{n}{k} + n)$

   The answer should be given using Θ-notation. No justification is required.

2. Apply counting sort to the following input array where each column corresponds to one item with its numeric key and single-character satellite data:

6	0	6	3	0	6	1	3	2	0
S	I	A	Y	S	!	U	D	D	T

   You **must** demonstrate the final state of the auxiliary arrays used in the algorithm, as well as the output of the array

   Solution.

   First we populate the `count` array, its length is 6 + 1 (6 is max value of original array):

$$\{3, 1, 1, 2, 0, 0, 3\}$$

   Then we get the `accum` array, its length is equal to `count` array:

$$\{3, 4, 5, 7, 7, 7, 10\}$$

   Then we populate the output arrays. The final state is following:

   Final state of `count` (not changed): $\{3, 1, 1, 2, 0, 0, 3\}$
   Final state of `accum`: $\{0, 3, 4, 5, 7, 7, 7\}$
   Sorted numeric keys: $\{0, 0, 0, 1, 2, 3, 3, 6, 6, 6\}$
   Sorted satellite data: $\{I, S, T, U, D, Y, D, S, A, !\}$

3. Let A be an array of positive integers. Different integers may have different number of digits, but the total number of digits over all the integers in A is n. Propose an algorithm to sort the array in $\Theta(n)$ time, based on Radix-Sort and Counting-Sort. More precisely:

   (a) Briefly (in one paragraph) summarise the idea of the algorithm *in your own words*.

   Answer.

   Let us firstly sort the array using Counting-Sort based on amount of digits of a number. This gives us groups of numbers having the same amount of digits, such that, firstly we have numbers with 1 digit, then 2 digits, and so on. Finally, let us sort every group using Radix-Sort. Therefore, we get our sorted array.

   (b) Provide complete pseudocode of the algorithm. You may use the pseudocode from [Cormen, Chapter 8] to help with a starting point of your pseudocode.

   Answer.

```
Sort(A, N) // N is size of A
    let B[1:N] be a new array
    B := Counting-Sort-Length(A, N) // B is now sorted by length of a number
    prev := 0 // index of start of group
    for i := 2 to N:
        if Get-Number-Of-Digits{B[i - 1]} < Get-Number-Of-Digits{B[i]}:
            B := Radix-Sort(B, N, Get-Number-Of-Digits{B[prev]}, prev, i)
            prev := i
    B := Radix-Sort(B, Get-Number-Of-Digits{B[prev]}, prev, N)
    return B

Counting-Sort-Length(A, N)
    M := Get-Number-Of-Digits{max{A}} + 1 // length of the maximal number in arr
    let count[1:M] be a new array
    let accum[1:M] be a new array
    let output[1:N] be a new array
    for i := 1 to M:
        count[i] := 0
        accum[i] := 0
    for i := 1 to N:
        count[Get-Number-Of-Digits{A[i]}] += 1
    for i := 2 to M:
        accum[i] := count[i] + accum[i - 1]
    for i := N to 1:
        output[accum[Get-Number-Of-Digits{A[i]}]] := A[i]
        accum[Get-Number-Of-Digits{A[i]}] -= 1
    return output

Radix-Sort(A, N, d, start, end)
    let B[1:N] be a new array
    for i := 0 to d - 1:
        B := Counting-Sort(B, N, i, start, end)
    return B

Counting-Sort(A, N, index, start, end)
    M := 10
    let count[1:M] be a new array
    let accum[1:M] be a new array
```

```
    let tmp[start:end - 1] be a new array
    let output[1:N] be a new array
    output := A
    for i := 1 to M:
        count[i] := 0
        accum[i] := 0
    for i := start to end - 1:
        count[Get-I-th-Digit{A[i], index}] += 1
    for i := 2 to M:
        accum[i] := count[i] + accum[i - 1]
    for i := end - 1 to start:
        tmp[accum[Get-I-th-Digit{A[i], index}]] := A[i]
        accum[Get-I-th-Digit{A[i], index}] -= 1
    for i := start to end - 1:
        output[i] := tmp[i - start + 1]
    return output
```

(c) **(+0.5% extra credit)** Justify the time complexity.

Solution.

Suppose p is amount of numbers in the array.

The COUNTING-SORT is $\Theta(p+n)$, but it is obvious that $p \leq n$,
so $\Theta(p+n) = \Theta(n)$.

Let us calculate the time complexity of running RADIX-SORT for every group. Time complexity of RADIX-SORT for one group is $\Theta(d \cdot (N+k))$, where $d = d_i$ — amount of digits in one number in group i, $N = p_i$ — amount of numbers in group i, $k = 9$ — maximum value of a digit. Or it can be rewritten as $\Theta(d_i \cdot (p_i + 9)) = \Theta(d_i \cdot p_i)$. It is obvious that $\sum d_i \cdot p_i = n$. Therefore, running RADIX-SORT for all groups will be $\Theta(n)$.

The overall time complexity will be $\Theta(n) + \Theta(n) = \Theta(n)$.

Q.E.D.

References

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein. *Introduction to Algorithms, Fourth Edition.* The MIT Press 2022