



Algorithms and Datastructures

Winter Term 2022

Exercise Sheet 3

Due: Wednesday, May 10th, 2pm

Exercise 1: Bucket Sort

(7 Points)

Bucketsort is an algorithm to stably sort an array $A[0..n-1]$ of n elements where the sorting keys of the elements take values in $\{0, \dots, k\}$. That is, we have a function `key` assigning a key $\text{key}(x) \in \{0, \dots, k\}$ to each $x \in A$.

The algorithm works as follows. First we construct an array $B[0..k]$ consisting of (initially empty) FIFO queues. That is, for each $i \in \{0, \dots, k\}$, $B[i]$ is a FIFO queue. Then we iterate through A and for each $j \in \{0, \dots, n-1\}$ we attach $A[j]$ to the queue $B[\text{key}(A[j])]$ using the function `enqueue`.

Finally we empty all queues $B[0], \dots, B[k]$ using `dequeue` and write the returned values back to A , one after the other. After that, A is sorted with respect to `key` and elements $x, y \in A$ with $\text{key}(x) = \text{key}(y)$ are in the same order as before.

Implement *Bucketsort* based on this description¹. You can use the template `BucketSort.py` which uses an implementation of FIFO queues that are available in `Queue.py` und `ListElement.py`.²

Exercise 2: Radix Sort

(13 Points)

Assume we want to sort an array $A[0..n-1]$ of size n containing integer values from $\{0, \dots, k\}$ for some $k \in \mathbb{N}$. We describe the algorithm *Radixsort* which uses *Bucketsort* as a subroutine.

Let $m = \lfloor \log_b k \rfloor$. We assume each key $x \in A$ is given in base- b representation, i.e., $x = \sum_{i=0}^m c_i \cdot b^i$ for some $c_i \in \{0, \dots, b-1\}$. First we sort the keys according to c_0 using *Bucketsort*, afterwards we sort according to c_1 and so on.³

- Implement *Radixsort* based on this description. You may assume $b = 10$, i.e., your algorithm should work for arrays containing numbers in base-10 representation. Use *Bucketsort* as a subroutine. If you did not solve task 1, you may use a library function (e.g., `sorted`) as alternative to *Bucketsort*. (7 Points)
- Compare the runtimes of *Bucketsort* and *Radixsort*. For both algorithms and each $k \in \{2 \cdot i \cdot 10^4 \mid i = 1, \dots, 60\}$, use an array of fixed size $n = 10^4$ with randomly chosen keys from $\{0, \dots, k\}$ as input and plot the runtimes. Shortly discuss your results in `experiences.txt`. (3 Points)
- Explain the asymptotic runtime of your implementations of *Bucketsort* und *Radixsort* depending on n and k . (3 Points)

¹Remember to make unit-tests and to add comments to your source code.

²You are allowed to use `librarys`, but note that the names of the methods may differ.

³The i -th digit c_i of a number $x \in \mathbb{N}$ in base- b representation (i.e., $x = c_0 \cdot b^0 + c_1 \cdot b^1 + c_2 \cdot b^2 + \dots$), can be obtained via the formula $c_i = (x \bmod b^{i+1}) \operatorname{div} b^i$, where `mod` is the modulo operation and `div` the integer division.