Institute of Pervasive Computing

Algorithms and Data Structures 1
Summer term 2023
Jäger, Beck, Anzengruber

Deadline CS: **Tue. 06.06.2023, 08:00**
Deadline AI: **Thu. 08.06.2023, 08:00**
Submission via: **Moodle**

Assignment 6

Elaboration time

Remember the time you need for the elaboration of this assignment and document it in moodle.

Sorting

For this assignment, please submit source code of your **`max_heap.py`** implementation (example 1), the **`radix_sort.py`** implementation (example 2b) and the PDF of the pen-and-paper-work (example 2a). Stick to the given interfaces and skeletons provided and don't forget to test your code!

1. HeapSort — 10 points

Implement the **HeapSort** algorithm based on the heap in the skeleton of the class **MaxHeap**. The heap must be created using **bottom-up** construction approach (as described in lecture and exercise) and **in-place** (without using an additional data structure). Make sure to use the slightly **different template** of **MaxHeap** compared to assignment 5. You may reuse suitable parts of **your own** code of assignment 05.

The following methods shall be implemented:

```
class MaxHeap:
    def __init__(self, list):
        # Creates a bottom-up maxheap in-place from the input list of numbers.

    def contains(self, val):
        # Tests if an item (val) is contained in the heap. Do not search the array sequentially, but use the heap properties.

    def sort(self):
        # This method sorts the numbers in-place using the maxheap data structure in ascending order.
```

a) The **MaxHeap()** constructor should create a valid maxheap using **in-place *bottom-up*** *construction*.

b) The method **contains()** should return *true*, if the element *val* is found in the heap (duplicates allowed). Use the properties of the maxheap for efficient implementation and do not search the array sequentially!

c) The **sort** method should implement the **HeapSort** algorithm in ***ascending*** order, using the created maxheap by repeated removal of the top element **in-place**. **Don't** use intermediate data structures!

You can use private (help) methods for your implementation.

Institute of Pervasive Computing

Assignment 6

Algorithms and Data Structures 1
Summer term 2023
Jäger, Beck, Anzengruber

Deadline CS: **Tue. 06.06.2023, 08:00**
Deadline AI: **Thu. 08.06.2023, 08:00**
Submission via: **Moodle**

2. RadixSort 2+12 points

Implement the **direct RadixSort** (base 7) for ***descending*** sorting of **positive** integer numbers. The task consists of the following elements:

a) Assuming the list contains **n** integer numbers, **d** is the number of digits in the largest number and **m=7** buckets are used, determine the runtime complexity of your algorithm in Big O notation. Submit your description as a PDF file.

b) Implement the method `sort()`, which takes a list of **positive** Integer numbers and sorts them in descending order using the RadixSort algorithm.

- use list of lists to store the buckets and their content
- in the end of every sorting iteration (when all list elements are assigned to buckets) take a snapshot of the entire bucketlist by calling the provided method `_add_bucket_list_to_history(...)`

Example for sorting the sequence {111, 32, 4, 50}:

```
Iteration #1
Bucket: 0       1       2       3       4       5       6
--------------------------------------------------------------------
        50      111     32              4

Iteration #2
Bucket: 0       1       2       3       4       5       6
--------------------------------------------------------------------
        4       111             32              50

Iteration #n
...
```

```python
# Runtime Complexity O(...)
class RadixSort:
    def __init__(self):

    def get_bucket_list_history(self):
        return self.bucket_list_history

    def sort(self, list):
        # Sorts a given list using radixsort in descending order and returns the sorted list

    def _add_bucket_list_to_history(self, bucket_list):
        # This method creates a snapshot (clone) of the bucketlist and adds it to the bucketlistHistory.
        @param bucket_list is your current bucketlist, after assigning all elements to be sorted to the buckets.
```

You can use private methods that help implementing the specified functionality, below are some suggestions that might be useful:

```
# returns the digit (base7) of val at position pos
def get_digit(val, pos)

# calculates buckets for position pos
def bucketSort(pos, buckets)

# Merges bucket lists into one list
def merge(buckets)
```