DVA245 Lab Assignment 4
Merge Sort with Queues

Anna Friebe

January 2023

1 Introduction

Merge sort is an based on the following ideas:

1. To find the smallest element of two sorted lists, we only need to compare the first element of each list. So, when merging two sorted lists into one, for each element to move, we only need to compare two elements - one from each of the input lists.

2. A list with only one element is always sorted.

An illustration of a merge of two sorted lists is provided in fig. 1

Looking at the first point - when merging we compare the first elements of the input lists, to determine which of these two elements to add to the end of the sorted output list. From this it is clear that we can implement merge sort using queues, and this is what you will do in this lab. Code skeleton and test are in a zip-file on Canvas.

2 Merge sort implementation with queues

The merge sort implementation is an imperative, non-recursive variant, and consists of three functions:

1. `merge`, merging two sorted queues into one, as illustrated in fig. 1.

2. `merge_level_queues`, merging a number of queues two-by-two using `merge` into half the number of merged queues. If the number of queues is odd the last one is added to the resulting queues without merging.

3. `merge_sort`, that creates a queue for each element to sort (since we know that a single element is sorted), and repeatedly calls `merge_level_queues` until just one sorted queue remains.

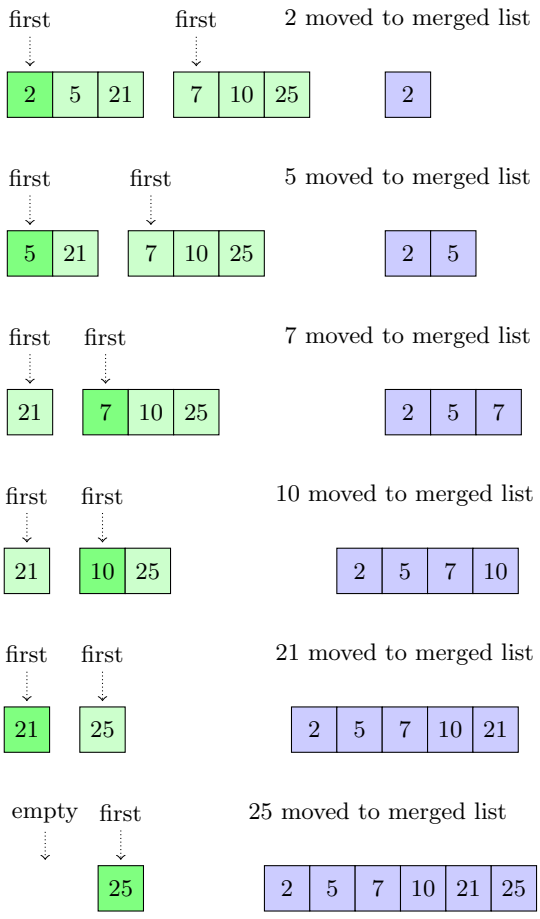

Figure 1: Merging of two sorted lists.

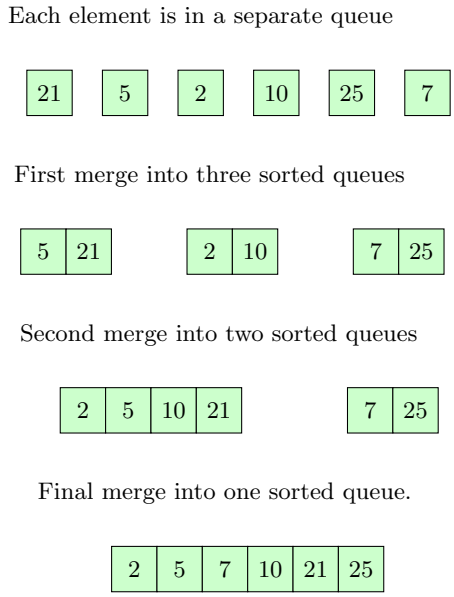

Figure 2: Merge sort with queues.

The full process is illustrated in fig. 2

We use the `deque` (double-ended quque) container from python's `collections` module for our queues. See the documentation of `deque` here. To enqueue elements we use `append`, and to dequeue elements we use `popleft`. The front element of the queue can be accessed without dequeueing with the operator `[0]`.

The file `merge_queue_begin.py` contains skeletons for the three functions, and `merge_queue_test.py` contains tests for them. In the zip-archive there is also a file `merge_array.py` with a recursive merge sort implementation for arrays (python lists), and accompanying tests in `merge_array_test.py`. The queue-based merge sort that you will implement is not recursive. Instead of dividing a list until we have the base case of one-element lists that are already sorted, we start by creating sorted queues containing just one element, and we start merging them two-by-two.

2.1 Merge

Implement the `merge` function to merge two sorted queues. As long as both input queues `S1` and `S2` contain elements, compare the front elements of both queues. Dequeue the smallest, and enqueue on the output queue `S`. When one of the input queues is empty, move the remaining elements from the non-empty input queue to the output queue. This requires about 10 lines of code. In the skeleton the queue `S` is created and returned, you need to add the described

functionality in between.

2.2 Merge level queues

Implement the `merge_level_queues` function. This function takes a queue of input queues, representing one level of queues as one line of the visualization in fig. 2, and returns a queue of merged queues as the line below in the same figure. Pseudocode for the `merge_level_queues` function is given in section 2.2. On lines 10-14, input queues are dequeued two-by-two and merged. The result is enqueued on `next_level_queues`. On lines 15-17 it is checked if the number of queues in the input level was odd and one queue is remaining. That is enqueued on `next_level_queues` without merging. You are not required to follow the pseudocode strictly.

```
1:  function MERGE_LEVEL_QUEUES(level_queues) returns a queue of merged
    queues
2:      inputs:
3:          level_queues, the queue of input queues to merge
4:      local variables:
5:          next_level_queues, the list of merged queues
6:          q1, one input queue from level_queues
7:          q2, one input queue from level_queues
8:
9:      next_level_queues ← the empty queue
10:     while LEN(level_queues) > 1 do
11:         q1 ← level_queue.DEQUEUE()
12:         q2 ← level_queue.DEQUEUE()
13:         next_level_queues.ENQUEUE(MERGE(q1, q2))
14:     end while
15:     if LEN(level_queues) > 0 then
16:         next_level_queues.ENQUEUE(level_queue.DEQUEUE())
17:     end if
18:     return next_level_queues
19: end function
```

2.3 Merge sort

Implement the `merge_sort` function, that takes an input queue `S` and returns a queue with the elements of `S` sorted.

1. Create the level queue.
2. For each element in `S`, dequeue it and add it as the single element of a new queue. Add the new single-element queue to the level queue. See the first line of fig. 2.

3. While the level queue contains more than one sorted queue, merge one level, and replace the level queue with the queue returned from `merge_level_queues`. For each iteration of the while-loop, we move one step down in fig. 2.

4. When there is just one sorted queue remaining in the level queue, dequeue it and return the resulting sorted queue.

The function contains less than 10 lines of code.

3 How to display

You need to explain your implementation, and also run the tests in `merge_queue_test.py` and show that they pass.