
Reading: GT 2.4.1-2.4.2, p. 94–98 [up to 25 minutes]

It is most important to understand what is a priority queue, and how it can be used to sort data. You should also understand in detail how selection and insertion sorting works (but I assume that you know this from the basic programming course — the description in the book is not a very good one).

The precise link between the priority queue sorting and selection/insertion sorting is not so important.

Ingredients: Priority queue, Sorting, Sorting with PQ, Selection Sort, Insertion Sort.

Priority Queue

A *priority queue* stores a collection of items with keys. Operations:

INSERT(k, o) — add object o with key k

REMOVE-MIN() — remove and return the object with minimum priority out of all in the queue

Additional operations: MIN-KEY(), SIZE(), IS-EMPTY()

Applications ubiquitous: real-time processes in scheduling, boarding standby passengers, stock market transactions, routing multimedia packets.

Priority queues are also used in construction of more complex algorithms, in particular in sorting and searching (also in *route planning*).

We shall talk about sorting applications today, and postpone the implementation details until next time.

Sorting Problem

Input: an array A of n objects with keys

Output: a *permutation* of A (=a reordering of A) such that the elements are placed in an increasing order of keys.

Sorting with a Priority Queue

Put all the elements into the queue using keys as priorities. Then remove the minimal element n times placing them in the output array.

PQ-SORT (A)

```

1  ▷ Create an empty priority queue  $Q$ 
2  for  $i \leftarrow 0$  to  $A.size - 1$ 
3      do  $Q.ININSERT(A[i].key, A[i])$ 
4  for  $i \leftarrow 0$  to  $A.size - 1$ 
5      do  $A[i] \leftarrow Q.REMOVE-MIN(Q)$ 
6  return

```

Let us consider the running time of the above algorithm, assuming that Q is implemented using a doubly-linked list. Insertion is done in constant time (by appending in the end of the list), and REMOVE-MIN is done in $O(n)$ time by traversing the list and comparing the keys, the same way we have solved ARRAY-MAX previously.

Q. What is the running time of PQ-SORT then?

As we can see we spend most of the time selecting the minimal element from the queue. This is in fact the same way, which is used by the standard SELECTION-SORT algorithm, which is in-place when implemented directly, without using a priority queue.

Selection Sort

PQ-SORT (A)

```

1  ▷ Create an empty priority queue  $Q$ 
2  for  $i \leftarrow 1$  to  $A.size - 1$ 
3      do Find the smallest element  $x$  in  $A[i..n - 1]$ 
4          Swap  $A[i] \leftrightarrow x$ 
5  return

```

Insertion Sort

Alternatively we could implement the priority queue with a doubly-linked list by inserting the elements always in the right positions. Then insertion is $O(n)$ and REMOVE-MIN is $O(1)$.

Q. What is the running time of PQ-SORT with such a priority queue?

We obtain an algorithm that is more traditionally known as INSERTION-SORT.

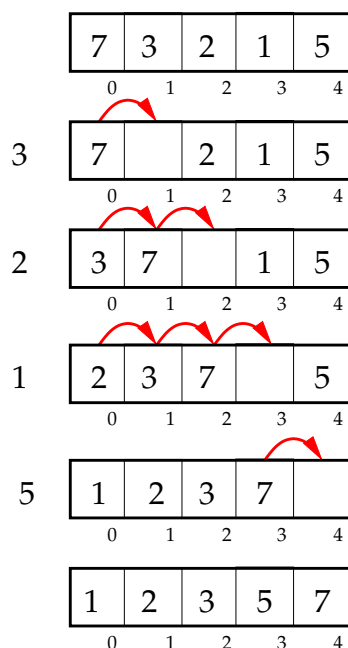


Figure 1: An example execution of INSERTION-SORT.

A direct, in-place, implementation of INSERTION-SORT:

INSERTION-SORT(A :array)

```

1  for  $i \leftarrow 1$  to  $A.size - 1$ 
2      do  $x \leftarrow A[i]$ 
3          $j \leftarrow i - 1$ 
4         while  $j \geq 0$  and  $x < A[j]$ 
5             do  $A[j + 1] \leftarrow A[j]$ 
6                 $j ++$ 
7          $A[j + 1] \leftarrow x$ 

```