```
begin
    place A_1, A_2, . . . , A_n in QUEUE;
    for j ← k step −1 until 1 do
        begin
            for l ← 0 until m − 1 do make Q[l] empty;
            while QUEUE not empty do
                begin
                    let A_i be the first element in QUEUE;
                    move A_i from QUEUE to bucket Q[a_ij]
                end;
            for l ← 0 until m − 1 do
                concatenate contents of Q[l] to the end of QUEUE
        end
end
```

Fig. 3.1. Lexicographic sort algorithm.

```
begin
    make QUEUE empty;
    for j ← 0 until m − 1 do make Q[j] empty;
    for l ← l_max step −1 until 1 do
        begin
            concatenate LENGTH[l] to the beginning of
              QUEUE;†
            while QUEUE not empty do
                begin
                    let A_i be the first string on QUEUE;
                    move A_i from QUEUE to bucket Q[a_il]
                end;
            for each j on NONEMPTY[l] do
                begin
                    concatenate Q[j] to the end of QUEUE;
                    make Q[j] empty
                end
        end
end
```

```
     procedure HEAPIFY(i, j):
1.   if i is not a leaf and if a son of i contains a larger element than i
       does then
         begin
2.           let k be a son of i with the largest element;
3.           interchange A[i] and A[k];
4.           HEAPIFY(k, j)
         end
```

The parameter j is used to determine whether i is a leaf and whether i has one or two sons. If $i > j/2$, then i is a leaf and HEAPIFY(i, j) need not do anything, since $A[i]$ is a heap by itself.

The algorithm to give all of A the heap property is simply:

```
procedure BUILDHEAP:
for i ← n† step −1 until 1 do HEAPIFY(i, n) □
```

```
begin
    BUILDHEAP;
    for i ← n step −1 until 2 do
        begin
            interchange A[1] and A[i];
            HEAPIFY(1, i − 1)
        end
end □
```

```
    procedure QUICKSORT(S):
1.  if S contains at most one element then return S
    else
        begin
2.          choose an element a randomly from S;
3.          let S1, S2, and S3 be the sequences of elements in S less
               than, equal to, and greater than a, respectively;
4.          return (QUICKSORT(S1) followed by S2 followed by
               QUICKSORT(S3))
        end
```

Fig. 3.7. Quicksort program.

```
begin
    i ← f;
    j ← l;
    while i ≤ j do
        begin
            while A[j] ≥ a and j ≥ f do j ← j − 1;
            while A[i] < a and i ≤ l do i ← i + 1;
            if i < j then
                begin
                    interchange A[i] and A[j];
                    i ← i + 1;
                    j ← j − 1
                end
        end
end
```

Fig. 3.8. Partitioning S into S_1 and $S_2 \cup S_3$, in place.

```
procedure SELECT(k, S):
if |S| < 50 then
    begin
        sort S;
        return kth smallest element in S
    end
else
    begin
        divide S into ⌊|S|/5⌋ sequences of 5 elements each
        with up to four leftover elements;
        sort each 5-element sequence;
        let M be the sequence of medians of the 5-element sets;
        m ← SELECT(⌈|M|/2⌉, M);
        let S1, S2, and S3 be the sequences of elements in S less
          than, equal to, and greater than m, respectively;
        if |S1| ≥ k then return SELECT(k, S1)
        else
            if (|S1| + |S2| ≥ k) then return m
            else return SELECT(k − |S1| − |S2|, S3)
    end
```

Fig. 3.10. Algorithm to select kth smallest element.

```
        procedure SELECT(k, S):
1.      if |S| = 1 then return the single element in S
        else
            begin
2.              choose an element a randomly from S;
3.              let S1, S2, and S3 be the sequences of elements in S less
                    than, equal to, and greater than a, respectively;
4.              if |S1| ≥ k then return SELECT(k, S1)
                else
5.                  if |S1| + |S2| ≥ k then return a
6.                  else return SELECT(k − |S1| − |S2|, S3)
            end
```

Fig. 3.12. Selection algorithm.