Algorithm dySkySeq_build This procedure[1] (see Algorithm 1) takes $\mathcal{D}$ and T as input, and returns a *seqStruct* $\mathcal{F}$. At the beginning, $\mathcal{F}$ is empty. Variable i indicates the dimension the algorithm is processing and is initialized to 1. Variable *seq* is a stack structure and is used to store the sequences. The algorithm proceeds in a Depth-First fashion. It calls the recursive function *recursiveSeq* with parameters (i) i to indicate the dimension B_i, (ii) T, (iii) *seq*, and (iv) the set $\mathcal{F}$ (line 5). Inside *recursiveSeq*, T' is filtered wrt B_i and o is pushed onto *seq* (line 3-4). If $(i < l)$, i.e. , the algorithm is not processing the last dimension, it recalls *recursiveSeq* with new parameters (line 6). Otherwise, i.e. $(i = l)$, at this step, *seq* contains l orders. Hence it computes the complementary skyline wrt T'' and inserts the pair $(seq, NSky_{\{seq\}}(T''))$ in $\mathcal{F}$ (line 8). Finally, o is popped from the sequence (line 9).

Algorithm 1: dySkySeq_build

Input: a set of dimensions $\mathcal{D} = \{A_1, \ldots, A_s, B_1, \ldots, B_l\}$, a dataset T
Output: *seqStruct* $\mathcal{F}$

```
1 Procedure recursiveSeq(i, T', seq, F)
2     foreach o ∈ Orders(B_i) in parallel do
3         T'' ← Π_[B_i|o.left](T') ∪ Π_[B_i|o.right](T')
4         seq.push(o)
5         if i < l then
6             recursiveSeq(i + 1, T'', seq, F)
7         else
8             F ← F ∪ (seq, NSky_{seq}(T''))
9         seq.pop(o)
1 begin
2     F ← ∅
3     i ← 1
4     seq ← ∅
5     recursiveSeq(i, T, seq, F)
6 return V
```

[1]This algorithms implementation is in `https://github.com/DynamicPartialSky/dySky/blob/master/dySky/dySky.h`