

# Efficient Algorithms to Rank and Unrank Permutations in Lexicographic Order

Blai Bonet Universidad Simón Bolívar bonet@ldc.usb.ve

## Introduction

- Ranking is the process of assigning a given permutation  $\pi$  over  $n$  numbers, a unique index (rank)  $r(\pi) \in \{0, \dots, n! - 1\}$
- Unranking is the inverse: given a rank  $r \in \{0, \dots, n! - 1\}$ , find a permutation  $\pi$  such that  $r(\pi) = r$
- Ranking is used to compute the index for PDB lookups, and is the **most critical** operation during node generation
- Ranking/unranking algorithms are said to be in lexicographic order iff lexicographically consecutive permutations are ranked into consecutive integers. Thus, e.g., the identity permutation is ranked into 0 and the reverse permutations is ranked into  $n! - 1$
- It has been argued that a lexicographic ranking can be more efficient for search as it tend to generate references (PDB indices) with some degree of locality

## Related Work

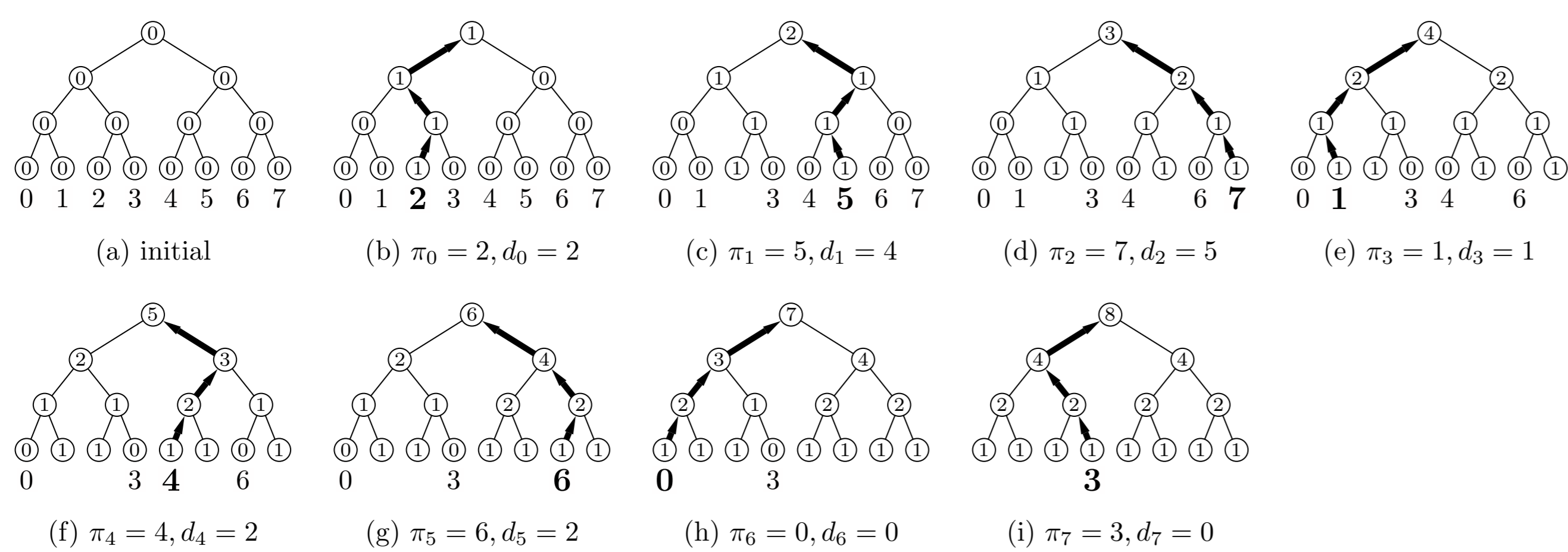
- Naive algorithms run in  $O(n^2)$  time
- Knuth (1973) presents  $O(n \log n)$  lexicographic algorithms based on modular arithmetic
- Using a data structure of Dietz (1989), the number of operations can be reduced to  $O(n \log n / \log \log n)$  yet the implementation is complex and hidden constants are high
- Myrvold and Ruskey (2001) present non-lexicographic algorithms that run in linear time (these are commonly used in heuristic search)
- Korf and Schultze (2005) present linear time lexicographic algorithms that require a precomputed table of exponential size
- Korf and Schultze's algorithms are **non-uniform**

## Contribution

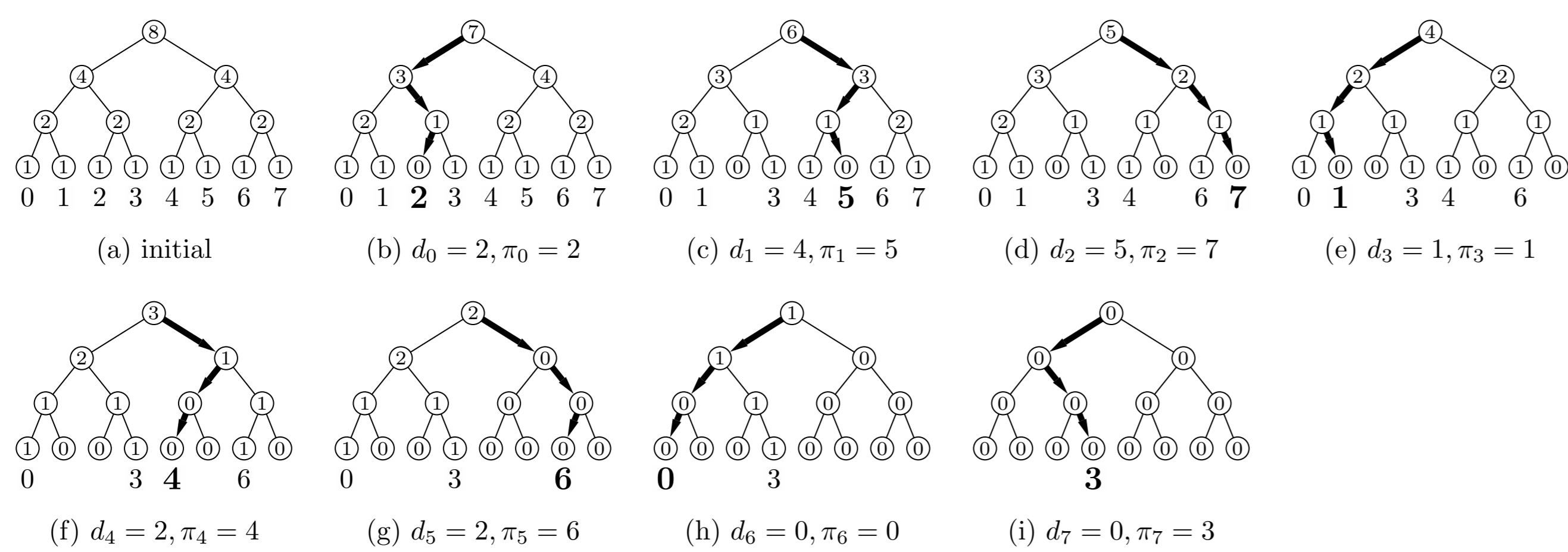
- Novel lexicographic algorithms that run in  $O(n \log n)$  time
- The algorithms are simple and efficient and, in our experiments, ran faster than the linear time algorithms of Myrvold and Ruskey for permutations up to size 128, and fell a bit short for sizes up to 1,024
- Also, give novel non-uniform lexicographic algorithms that require much less space than Korf and Schultze's

## Uniform Algorithms

Ranking:  $\pi = \langle 25714603 \rangle \rightarrow d = \langle 24512200 \rangle \rightarrow r = \#$



Unranking:  $r = \# \rightarrow d = \langle 24512200 \rangle \rightarrow \pi = \langle 25714603 \rangle$



**Input:** permutation  $\pi$  and array  $T$   
**Output:** rank of  $\pi$   
 begin  
 $k := \lceil \log n \rceil$   
 $rank := 0$   
 for  $i = 1$  to  $2^{k-1} - 1$  do  $T[i] := 0$   
 for  $i = 1$  to  $n$  do  
 $ctr := \pi[i]$   
 $node := 2^k + \pi[i]$   
 for  $j = 1$  to  $k$  do  
 if  $node$  is odd then  
 $ctr := ctr - T[(node \gg 1) \ll 1]$   
 $T[node] := T[node] + 1$   
 $node := node \gg 1$   
 $T[node] := T[node] + 1$   
 $rank := rank \cdot (n + 1 - i) + ctr$   
 return  $rank$   
 end

**Input:** factorial digits  $d$ , array  $\pi$ , and array  $T$   
**Output:** permutation  $\pi$   
 begin  
 $k := \lceil \log n \rceil$   
 for  $i = 0$  to  $k$  do  
 for  $j = 1$  to  $2^i$  do  
 $T[2^i + j - 1] := 1 \ll (k - i)$   
 for  $i = 1$  to  $n$  do  
 $digit := d[i]$   
 $node := 1$   
 for  $j = 1$  to  $k$  do  
 $T[node] := T[node] - 1$   
 $node := node \ll 1$   
 if  $digit \geq T[node]$  then  
 $digit := digit - T[node]$   
 $node := node + 1$   
 $T[node] := 0$   
 $\pi[i] \leftarrow node - 2^k$   
 end

## Non-Uniform Algorithms

Korf and Schultze's Ranking Algorithm:

$$d_i = \pi_i - \#\{\text{elements to the left of position } i \text{ that are less than } \pi_i\} \\ = \pi_i - T_r[N \ll (n - \pi_i)]$$

where  $T_r[x]$  equals the number of 1-bits in the binary representation of  $x$ . In the example,

$i$	$\pi_i$	$N$ before	$N \ll (n - \pi_i)$	$T_r$	$d_i$	$N$ after
0	2	0000 0000	0000 0000	0	2	0000 0100
1	5	0000 0100	0010 0000	1	4	0010 0100
2	7	0010 0100	0100 1000	2	5	1010 0100
3	1	1010 0100	0000 0000	0	1	1010 0110
4	4	1010 0110	0110 0000	2	2	1011 0110
5	6	1011 0110	1101 1000	4	2	1111 0110
6	0	1111 0110	0000 0000	0	0	1111 0111
7	3	1111 0111	1110 0000	3	0	1111 1111

New Algorithm:

- Replace table  $T_r$  of size  $O(2^n \log n)$  bits by a single table  $T_r^*$  of size  $O(2^m \log m)$  bits, and perform  $\lceil n/m \rceil$  lookups instead of one. For example, for  $m = 4$ , we have that

$$T_r[x] = T_r^*[x \& 1111] + T_r^*[(x \gg 4) \& 1111].$$

Therefore,

$$d_i = \pi_i - T_r[N \ll (n - \pi_i)] \\ = \pi_i - T_r^*[(N \ll (n - \pi_i)) \& 1111] - T_r^*[(N \ll (n - \pi_i)) \gg 4] \& 1111 \\ = \pi_i - T_r^*[N_1] - T_r^*[N_2].$$

In the example,

$i$	$\pi_i$	$N$ before	$N \ll (n - \pi_i)$	$T_r^*[N_2]$	$T_r^*[N_1]$	$d_i$	$N$ after
0	2	0000 0000	0000 0000	0	0	2	0000 0100
1	5	0000 0100	0010 0000	1	0	4	0010 0100
2	7	0010 0100	0100 1000	1	1	5	1010 0100
3	1	1010 0100	0000 0000	0	0	1	1010 0110
4	4	1010 0110	0110 0000	2	0	2	1011 0110
5	6	1011 0110	1101 1000	3	1	2	1111 0110
6	0	1111 0110	0000 0000	0	0	0	1111 0111
7	3	1111 0111	1110 0000	3	0	0	1111 1111

Interesting Tradeoffs:

$\Theta$	$m = 1$	$m = \log n$	$m = n^\epsilon$	$m = n / \log n$	$m = n/k$	$m = n$
time	$n^2$	$n^2 / \log n$	$n^{2-\epsilon}$	$n \log n$	$n$	$n$
space	1	$n \log \log n$	$2^{n^\epsilon} \log n$	$2^{n/\log n} \log n$	$2^{n/k} \log n$	$2^n \log n$

## Experiments

