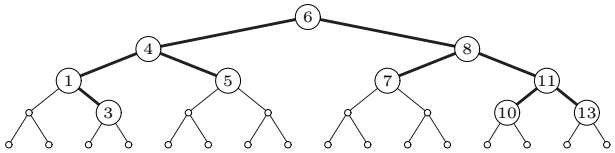

Figure 2: The embedding of a search tree with height 4 and size 10 in a complete tree with height 5

2 Memory Layouts of Static Trees

In this section we discuss four memory layouts for static trees: *DFS*, *inorder*, *BFS*, and *van Emde Boas* layouts. We assume that each node is represented by a node record and that all node records for a tree are stored in one array. We distinguish between *pointer based* and *implicit* layouts. In pointer based layouts the navigation between a node and its children is done via pointers stored in the node records. In implicit layouts no pointers are stored; the navigation is based solely on address arithmetic. Whereas all layouts have pointer based versions, implicit versions are only possible for layouts where the address computation is feasible. In this paper we will only consider implicit layouts of complete trees. A complete tree of size n is stored in an array of n node records.

DFS layout The nodes of T are stored in the order they are visited by a left-to-right depth first traversal of T (i.e. a preorder traversal).

Inorder layout The nodes of T are stored in the order that they are visited by a left-to-right inorder traversal of T.

BFS layout The nodes of T are stored in the order they are visited by a left-to-right breath first traversal of T.

van Emde Boas layout The layout is defined recursively: A tree with only one node is a single node record. If a tree T has two or more nodes, let $H_0 = \lceil h(T)/2 \rceil$, let T_0 be the tree consisting of all nodes in T with depth at most H_0, and let $T_1, \ldots, T_k$ be the subtrees of T rooted at nodes with depth $H_0 + 1$, numbered from left to right. We will denote T_0 the *top tree* and $T_1, \ldots, T_k$ the *bottom trees* of the recursion. The van Emde Boas layout of T consists of the van Emde Boas layout of T_0 followed by the van Emde Boas layouts of $T_1, \ldots, T_k$.

Figure 3 gives the implicit DFS, inorder, BFS, and van Emde Boas layouts for a complete tree with height four.

We now discuss how to calculate the position of the children of a node v at position i in the implicit layouts. For the BFS layout, the children are at position $2i$ and $2i + 1$—a fact exploited already in the 1960s in the design of the implicit binary heap [23]. For the DFS layout, the two children are at

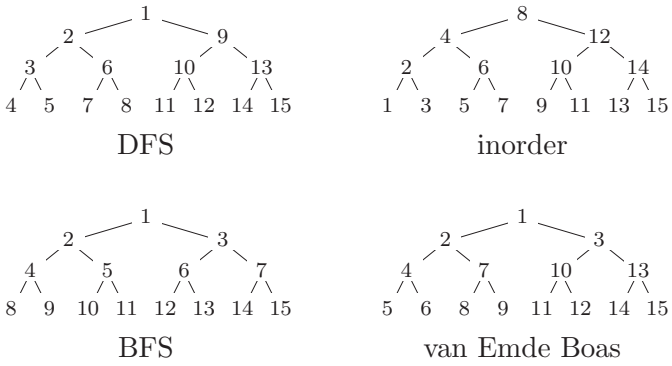

Figure 3: The DFS, inorder, BFS, and van Emde Boas layouts for a complete tree with height 4. Numbers designate positions in the array of node records

positions $i+1$ and $i+2^{h(v)-1}$, and in the inorder layout the two children are at positions $i-2^{h(v)-2}$ and $i+2^{h(v)-2}$.

For the implicit van Emde Boas layout the computations are more involved. Our solution is based on the fact that if we for a node in the tree unfold the recursion in the van Emde Boas layout until this node is the root of a bottom tree, then the unfolding will be the same for all nodes of the same depth. In a precomputed table of size $O(\log n)$, we for each depth d store the size $B[d]$ of this bottom tree, the size $T[d]$ of the corresponding top tree, and the depth $D[d]$ of the root of the corresponding top tree. When laying out a static tree, we build this table in $O(\log n)$ time by a straightforward recursive algorithm.

During a search from the root, we keep track of the position i in a BFS layout of the current node v of depth d. We also store the position $Pos[j]$ in the van Emde Boas layout of the node passed at depth j for $j < d$ during the current search. As the bits of the BFS number i represents the left and right turns made during the search, the $\log(T[d]+1)$ least significant bits of i gives the index of the bottom tree with v as root among all the bottom trees of the corresponding top tree. Since $T[d]$ is of the form 2^k-1, these bits can be found as i AND $T[d]$. It follows that for $d > 1$, we can calculate the position $Pos[d]$ of v by the expression

$$Pos[d] = Pos[D[d]] + T[d] + (i \text{ AND } T[d]) \cdot B[d] \ .$$

At the root, we have $i = 1$, $d = 1$, and $Pos[1] = 1$. Navigating from a node to a child is done by first calculating the new BFS position from the old, and then finding the value of the expression above.

The worst case number of memory transfers during a top down traversal of a path using the above layout schemes is as follows, assuming each block stores B nodes. With the BFS layout, the topmost $\lfloor \log(B+1) \rfloor$ levels of the tree

will be contained in at most two blocks, whereas each of the following blocks read only contains one node from the path. The total number of memory transfers is therefore $\Theta(\log(n/B))$. For the DFS and inorder layouts, we get the same worst case bound when following the path to the rightmost leaf, since the first $\lceil \log(n+1) \rceil - \lceil \log B \rceil$ nodes have distance at least B in memory, whereas the last $\lfloor \log(B+1) \rfloor$ nodes are stored in at most two blocks. As Prokop [19, Section 10.2] observed, in the van Emde Boas layout there are at most $O(\log_B n)$ memory transfers. Note that only the van Emde Boas layout has the asymptotically optimal bound achieved by B-trees [4].

We note that DFS, inorder, BFS, and van Emde Boas layouts all support efficient range queries (i.e. the reporting of all elements with keys within a given query interval), by the usual recursive inorder traversal of the relevant part of the tree, starting at the root.

We argue below that the number of memory transfers for a range query in each of the four layouts equals the number of memory transfers for two searches plus $O(k/B)$, where k is the number of elements reported. If a range reporting query visits a node that is not contained in one of the search paths to the endpoints of the query interval, then all elements in the subtree rooted at the node will be reported. As a subtree of height $\lceil \log(B+1) \rceil$ stores between B and $2B-1$ elements, at most k/B nodes with height larger than $\lceil \log(B+1) \rceil$ are visited which are not on the search paths to the two endpoints. Since subtrees are stored contiguously for both the inorder and DFS layouts, a subtree of height $\lceil \log(B+1) \rceil$ is stored in at most three blocks. The claimed bound follows for these layouts. For the van Emde Boas layout, consider a subtree T of height $\lceil \log(B+1) \rceil$. There exists a level in the recursive layout where the topmost levels of T will be stored in a recursive top tree and the remaining levels of T will be stored in a contiguous sequence of bottom trees. Since the top tree and each bottom tree has size less than $2B-1$ and the bottom trees are stored contiguously in memory, the bound for range reportings in the van Emde Boas layout follows.

For the BFS layout, we prove the bound under the assumption that the memory size is $\Omega(B \log B)$. Observe that the inorder traversal of the relevant nodes consists of a left-to-right scan of each level of the tree. Since each level is stored contiguously in memory, the bound follows under the assumption above, as the memory can hold one block for each of the lowest $\lceil \log(B+1) \rceil$ levels simultaneously.

3 Search Trees of Small Height

In the previous section, we considered how to lay out a static complete tree in memory. In this section, we describe how the static layouts can be used to store dynamic balanced trees. We first describe an insertions only scheme