Solutions to Exercises

Solutions to Exercises in Section 1.1

1. See [649].
2. There are $d!$ permutations of d attributes. It can be shown using symmetry arguments, as well as additional properties, that only $\binom{d}{\lfloor (d+1)/2 \rfloor}$ of the permutations are needed. For more details, see [1214], as well as [1046, pp. 577, 743–744].
3. The worst case is $N + d - 1$, where the first attribute value is different for all records, and now the remaining chains are one element long at each level.
4. In the case of a doubly chained tree, in the worst case, we only test one attribute value at a time for a total of $O(N)$ tests. On the other hand, in the case of a sequential list, in the worst case, we may have to test all attribute values of each record for a total of $O(N \cdot d)$ tests.
5. (Hans-Peter Kriegel) This worst case arises when the resulting tree has the depth $O(d \log_2 N)$. To see how this case arises, assume a configuration of N records in the following sequence. The first N/d records have N/d different values for the first attribute, and the remaining $d-1$ attributes have arbitrary values. The first attribute value of the remaining $N \cdot (d-1)/d$ records is a constant, say c_1. The second set of N/d records in the sequence have N/d different values for the second attribute, and the remaining $d-2$ attributes have arbitrary values. The second attribute value of the remaining $N \cdot (d-2)/d$ records is a constant, say c_2. This same process is applied for the remaining $d-2$ sets of N/d records—that is, the ith set of N/d records in the sequence have N/d different values for the ith attribute and the remaining $d-i$ attributes have arbitrary values. The second attribute value of the remaining $N \cdot (d-i)/d$ records is a constant, say c_i. This results in a sequence of d height-balanced binary trees of height $\log_2 N/d$ apiece. Thus, the total height of the structure is $O(d \cdot \log_2 N/d) = O(d \cdot (\log_2 N - \log_2 d))$, which can be approximated by $O(d \log_2 N)$ since $d \ll N$.
6. Assume that all of the attribute values are distinct. This means that the subfile H of the records that lie on the partitioning hyperplane is empty as this maximizes the size of the left and right subtrees of each node. Let $S(N,d)$ denote the time needed to build the tree and solve the following recurrence relation [1133]:

$$S(N,d) = 2 \cdot S(N/2,d) + 2 \cdot S(N/2,d-1)$$

   where $S(1,d) = O(d)$ and $S(N,1) = O(N \cdot \log_2 N)$. The recurrence relation indicates the space needed for the trees corresponding to the subfiles L and R, which contain $N/2$ records, and the subfiles LH and RH, which also contain $N/2$ records and are projections on the $(d-1)$-dimensional hyperplane. $S(N,1)$ is the space needed for the one-dimensional structure where at each level LH and RH are just sets of pointers to the records corresponding to the elements of L and H, respectively, and there are N records at each of the $\log_2 N$ levels. $S(1,d)$ is the space needed for a d-dimensional record that lies on the hyperplane corresponding to the record, and there are d levels in this structure with an empty right subtree and a nonempty left subtree as there are d attributes. The solution to this recurrence relation is $S(N,d) = O(N \cdot (\log_2 N)^d)$. For more details, see [1133].

4. Bentley [164] shows that the probability of constructing a given k-d tree of N nodes by inserting N nodes in a random order into an initially empty k-d tree is the same as the probability of constructing the same tree by random insertion into a one-dimensional binary search tree. Once this is done, results that have been proved for one-dimensional binary search trees will be applicable to k-d trees. The proof relies on viewing the records as d-tuples of permutations of the integers $1,\ldots,N$. The nodes are considered random if all of the $(N!)^k$ d-tuples of permutations are permitted to occur. It is assumed that the key values in a given record are independent.

5. See [1912] for some hints.

Solutions to Exercises in Section 1.5.1.2

1.
```
recursive pointer node procedure FINDDMINIMUM(P,D)
/* Find the node with the smallest value for the D coordinate in the k-d tree
   rooted at P */
value pointer node P
value direction D
pointer node L,H
if ISNULL(P) then return NIL
elseif DISC(P) = D then /* The 'MINIMUM' node is in the left subtree (i.e.,
                           LOCHILD) */
   if ISNULL(LOCHILD(P)) then return P
   else P ← LOCHILD(P)
   endif
endif
/* Now, DISC(P) ≠ D. The 'MINIMUM' node is the node with the smallest D
   coordinate value of P, min(LOCHILD(P)), and min(HICHILD(P)) */
L ← FINDDMINIMUM(LOCHILD(P),D)
H ← FINDDMINIMUM(HICHILD(P),D)
if not ISNULL(H) and COORD(H,D) ≤ COORD(P,D) then P ← H
endif
if not ISNULL(L) and COORD(L,D) ≤ COORD(P,D) then P ← L
endif
return P
```

2.
```
procedure KDDELETE(P,R)
/* Delete node P from the k-d tree rooted at node R. If the root of the tree was
   deleted, then reset R. */
value pointer node P
reference pointer node R
pointer node F,N
N ← KDDELETE1(P)
F ← FINDFATHER(P,R,NIL)
if ISNULL(F) then R ← N /* Reset the pointer to the root of the tree */
else CHILD(F,CHILDTYPE(P)) ← N
endif
RETURNTOAVAIL(P)
```

```
recursive pointer node procedure KDDELETE1(P)
/* Delete node P and return a pointer to the root of the resulting subtree. */
value pointer node P
pointer node F,R
direction D
if ISNULL(LOCHILD(P)) and ISNULL(HICHILD(P)) then
   return NIL /* P is a leaf */
else D ← DISC(P)
endif
if ISNULL(HICHILD(P)) then /* Special handling when HICHILD is empty */
```