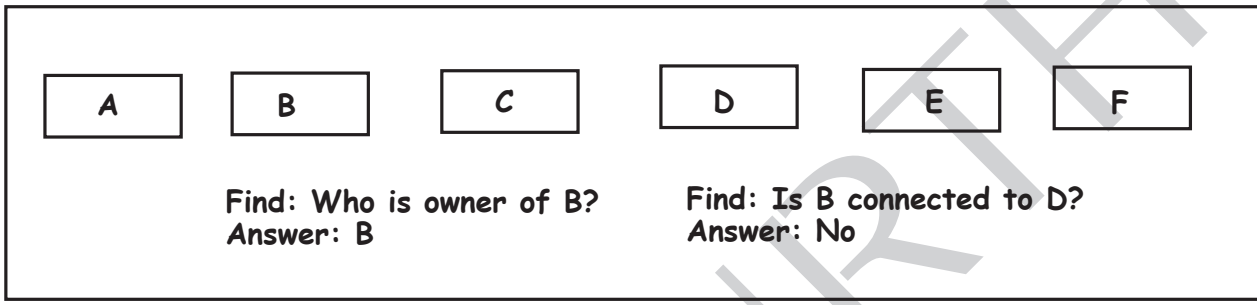
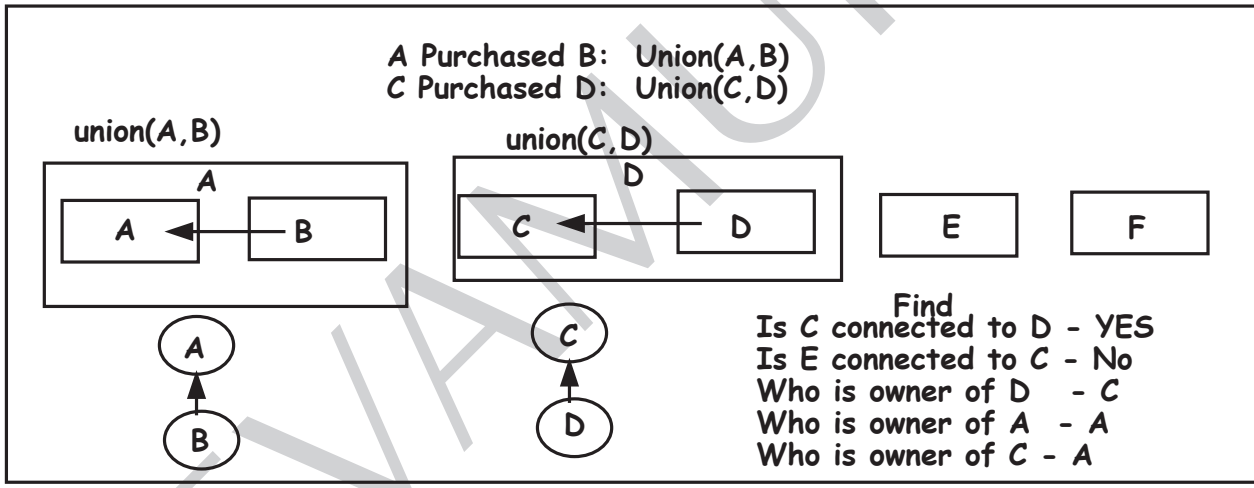
Chapter 17

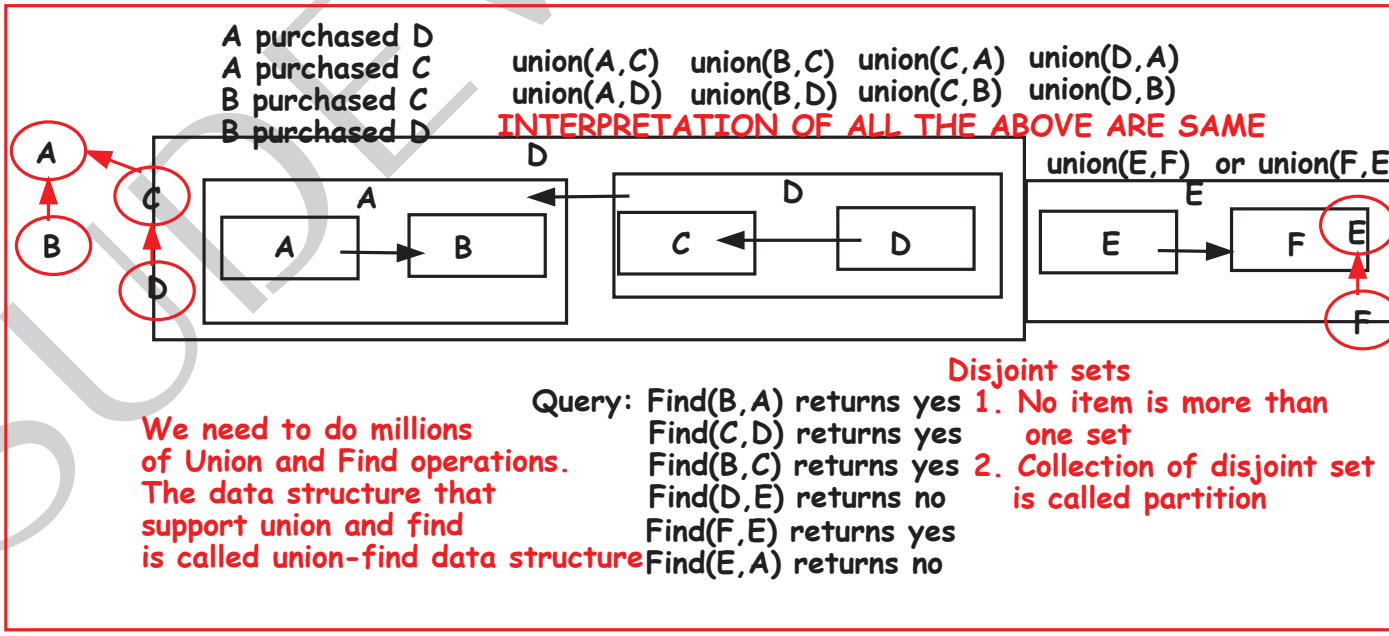
Disjoint sets

17.1 Introduction

17.2 Need for Union and Find Data structure







17.3 Smart Union Algorithm

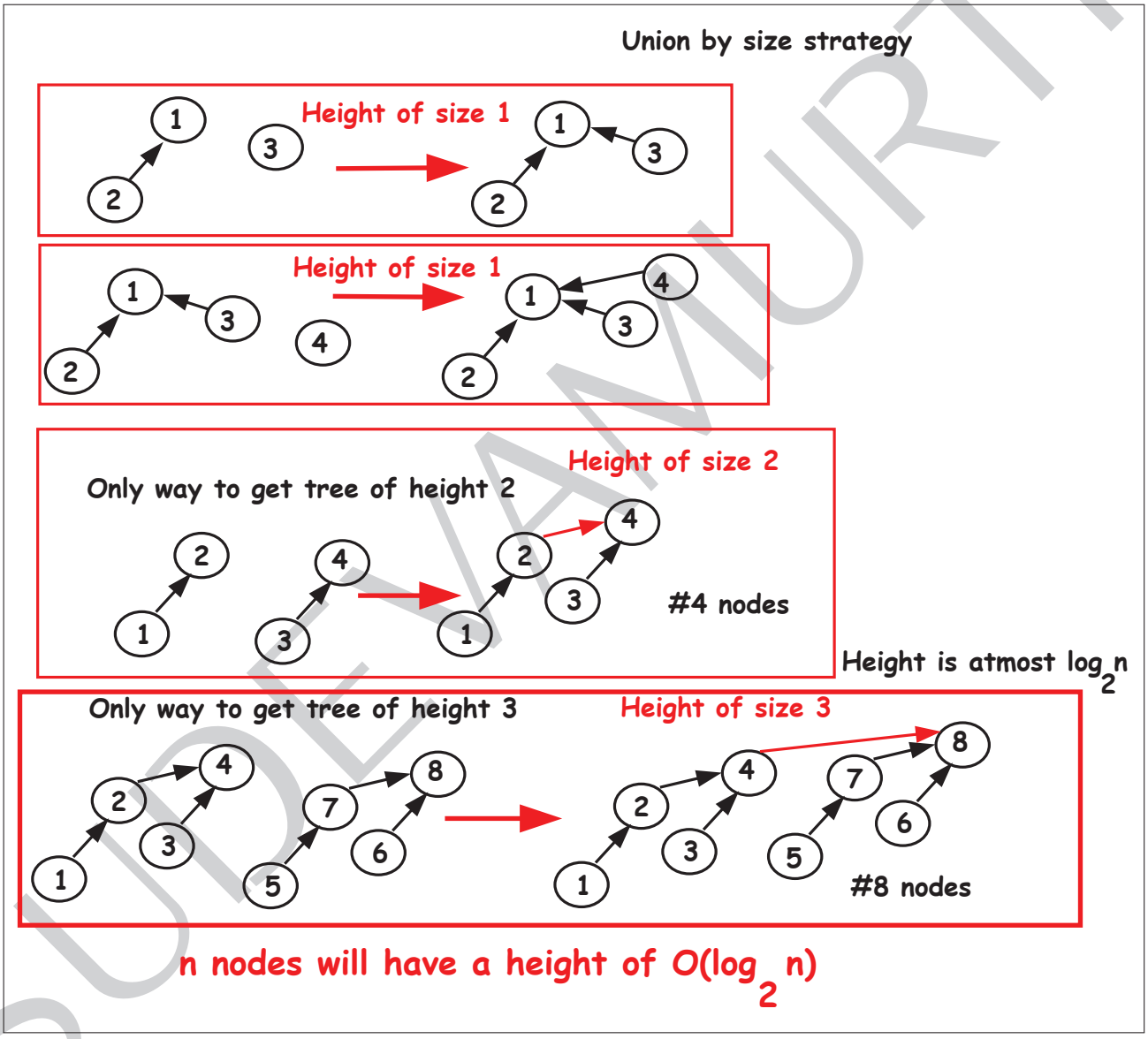


Figure 17.2: To show the height of the tree is at most $O(\log_2 n)$

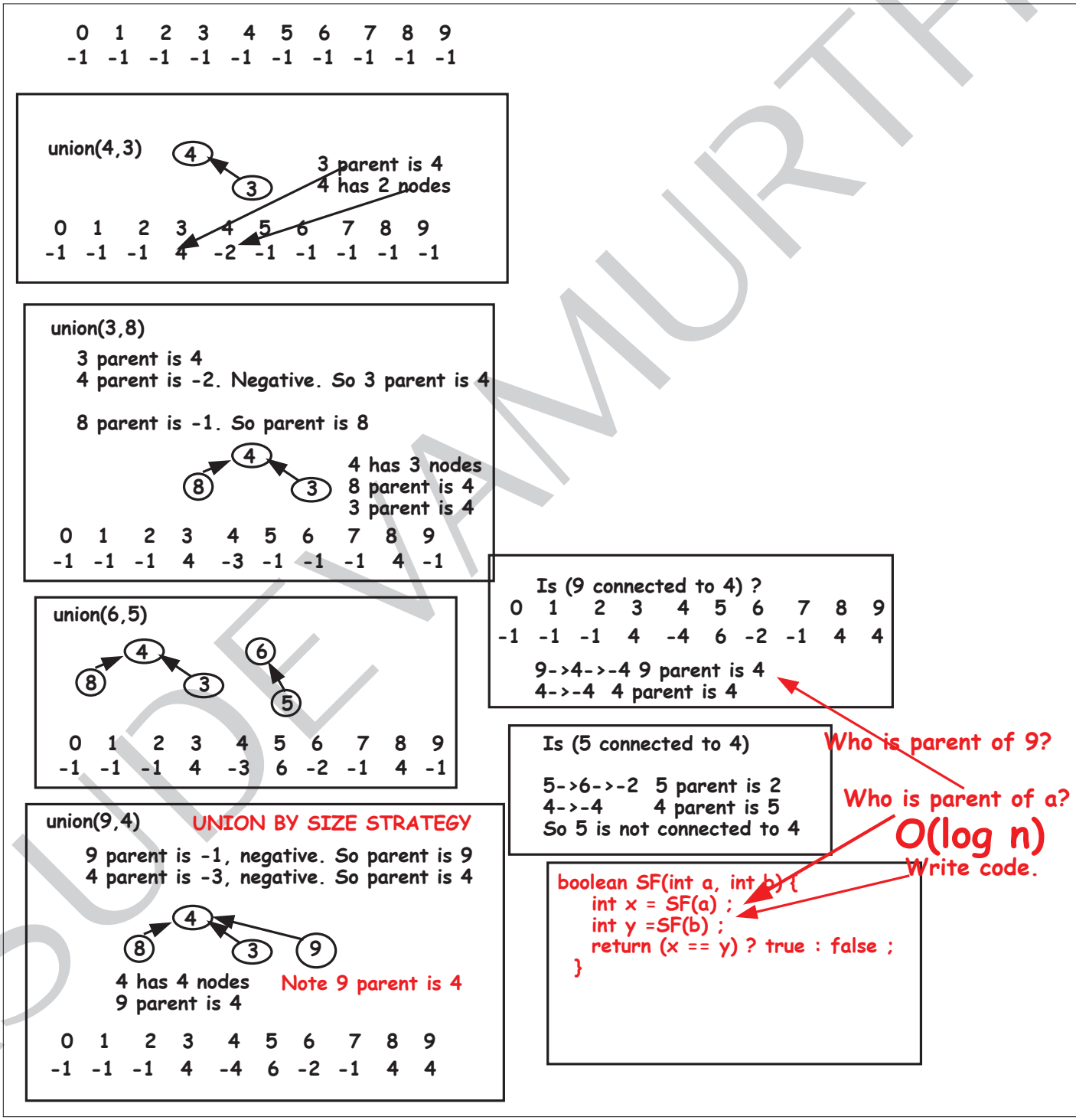
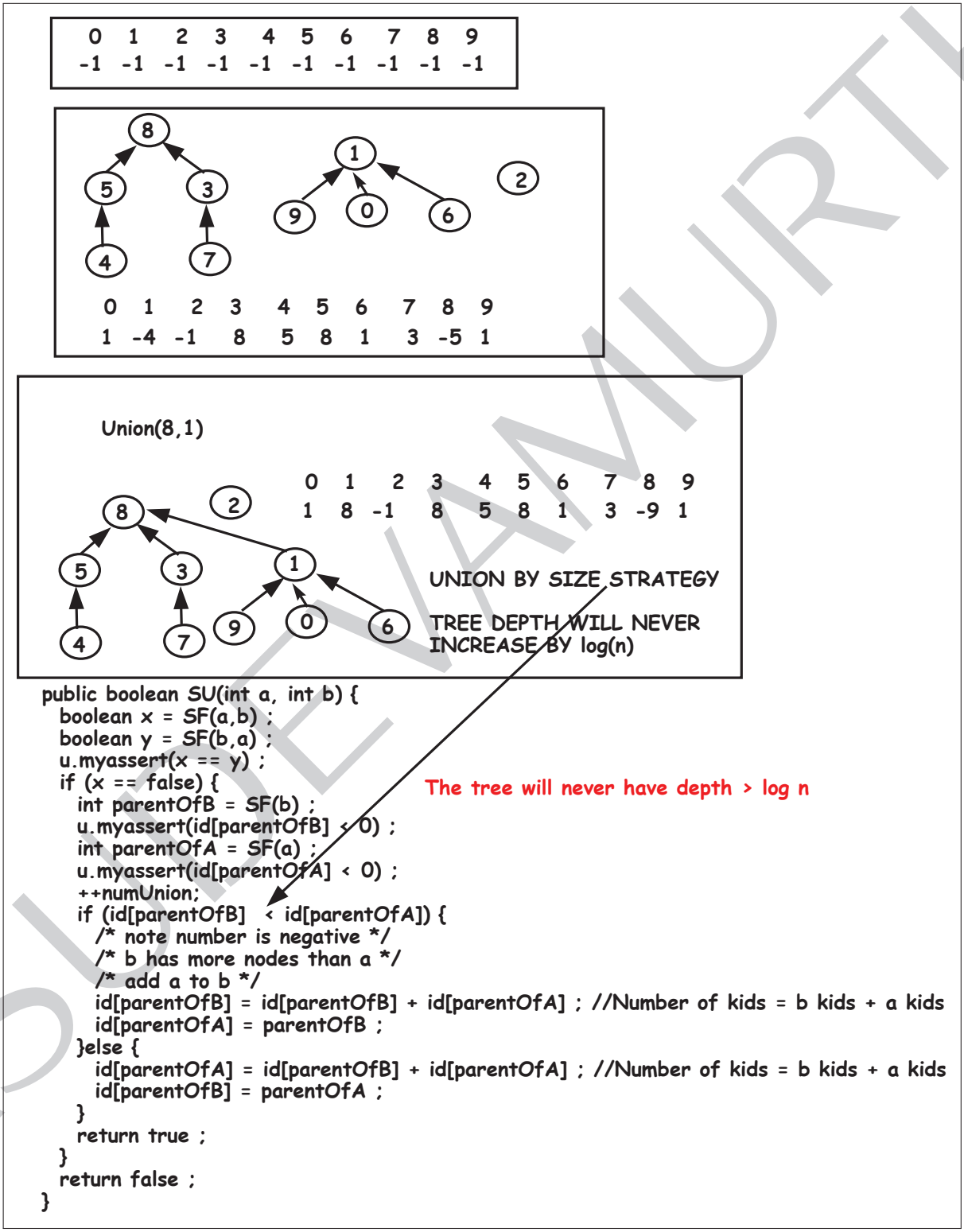


Figure 17.3: Smart union by size strategy algorithm




```
public boolean SU(int a, int b) {
  boolean x = SF(a,b) ;
  boolean y = SF(b,a) ;
  u.myassert(x == y) ;
  if (x == false) {
    int parentOfB = SF(b) ;
    u.myassert(id[parentOfB] < 0) ;
    int parentOfA = SF(a) ;
    u.myassert(id[parentOfA] < 0) ;
    ++numUnion;
    if (id[parentOfB]  < id[parentOfA]) {
      /* note number is negative */
      /* b has more nodes than a */
      /* add a to b */
      id[parentOfB] = id[parentOfB] + id[parentOfA] ; //Number of kids = b kids + a kids
      id[parentOfA] = parentOfB ;
    }else {
      id[parentOfA] = id[parentOfB] + id[parentOfA] ; //Number of kids = b kids + a kids
      id[parentOfB] = parentOfA ;
    }
    return true ;
  }
  return false ;
}
```

Figure 17.4: Smart union by size strategy algorithm

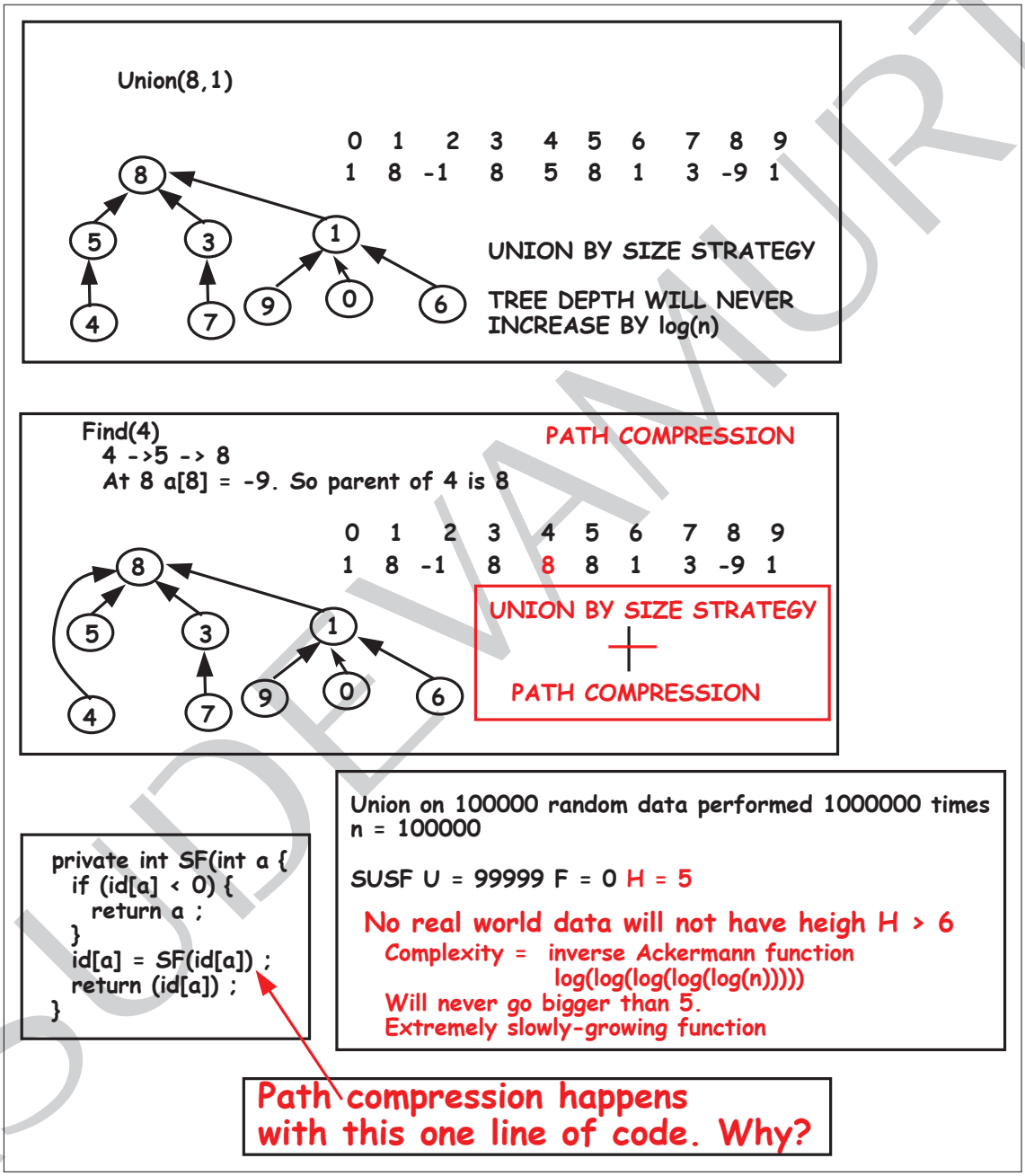


Figure 17.5: Smart union by size strategy with path compression algorithm

17.4 Problem set

Problem 17.4.1. Implement **Kruskal algorithm** as shown in figures 17.6 and 17.7. You must use **SUSF** algorithm for finding the loop.

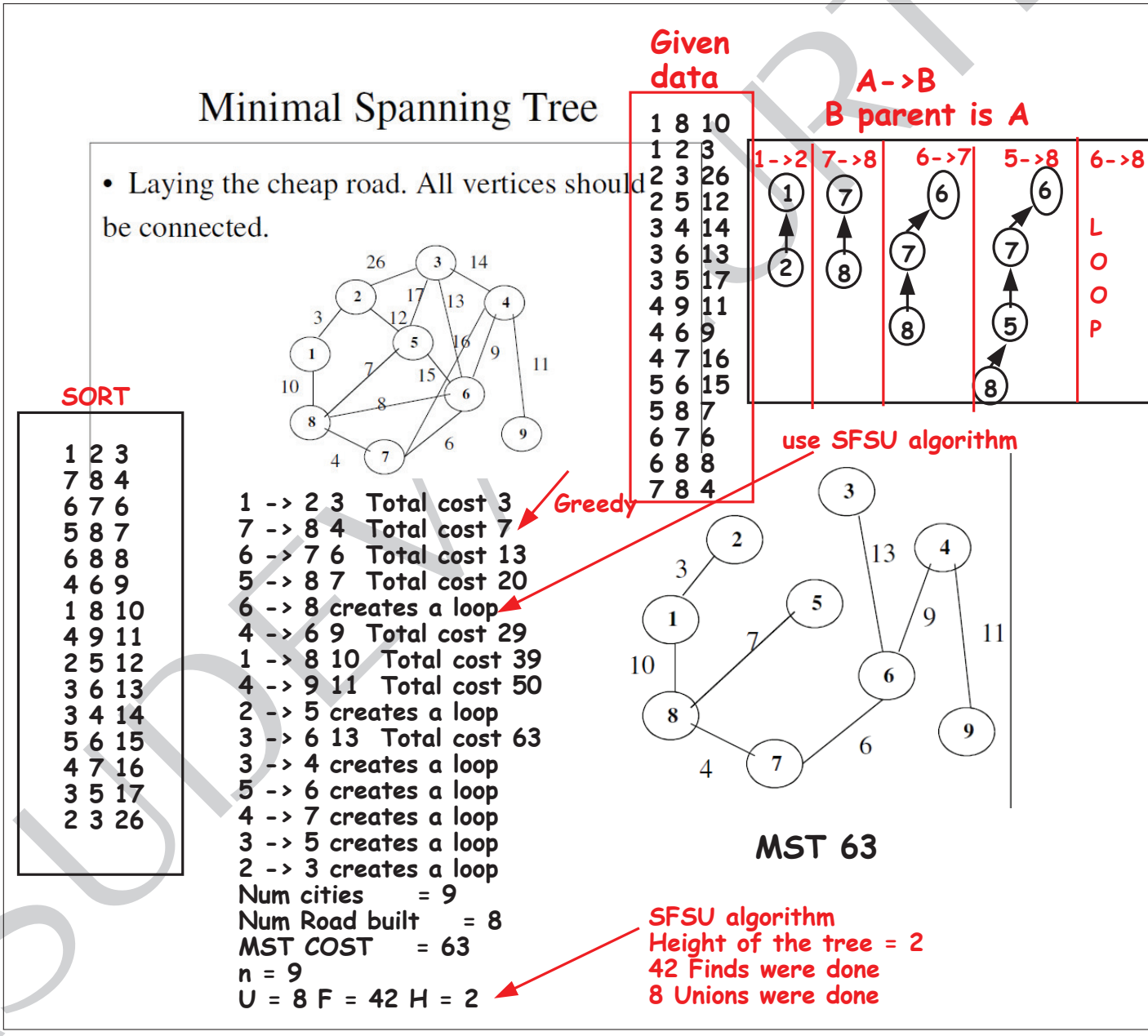


Figure 17.6: Kruskal Algorithm in action

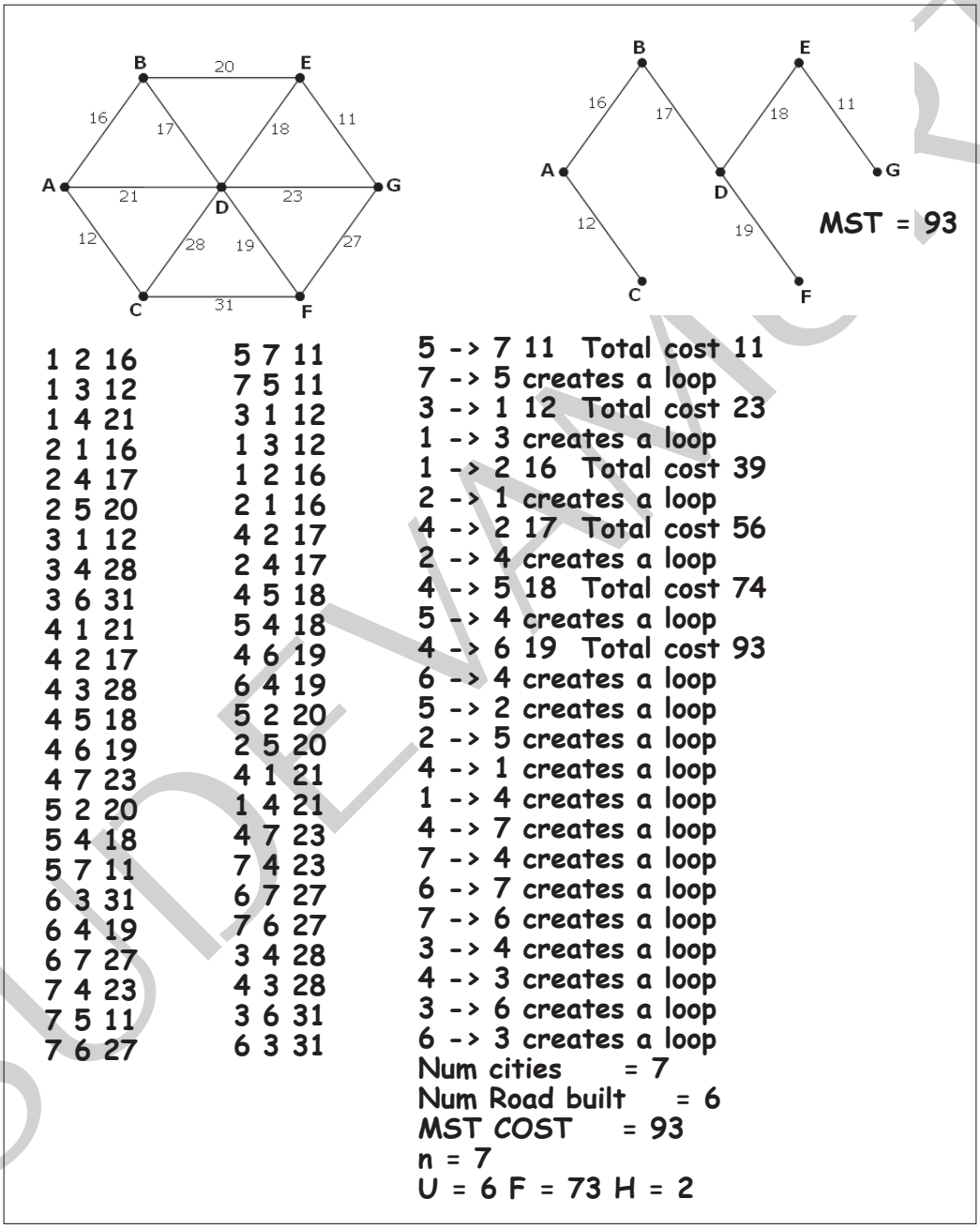


Figure 17.7: Kruskal Algorithm in action

