```
Type DAndC(P)
{
    if Small(P) return S(P);
    else {
        divide P into smaller instances P_1, P_2, ..., P_k, k ≥ 1;
        Apply DAndC to each of these subproblems;
        return Combine(DAndC(P_1),DAndC(P_2),...,DAndC(P_k));
    }
}
```

Program 3.1 Control abstraction for divide and conquer

recursive applications of DAndC. Combine is a function that determines the solution to P using the solutions to the k subproblems. If the size of P is n and the sizes of the k subproblems are $n_1, n_2, \ldots, n_k$, respectively, then the computing time of DAndC is described by the recurrence relation

$$T(n) = \begin{cases} g(n) & n \text{ small} \\ T(n_1) + T(n_2) + \cdots + T(n_k) + f(n) & \text{otherwise} \end{cases} \quad (3.1)$$

where $T(n)$ is the time for DAndC on any input of size n and $g(n)$ is the time to compute the answer directly for small inputs. The function $f(n)$ is the time for dividing P and combining the solutions to subproblems. For divide-and-conquer-based algorithms that produce subproblems of the same type as the original problem, it is very natural to first describe such algorithms using recursion.

```
int BinSrch(Type a[], int i, int l, Type x)
// Given an array a[i:l] of elements in nondecreasing
// order, 1<=i<=l, determine whether x is present, and
// if so, return j such that x == a[j]; else return 0.
{
   if (l==i) { // If Small(P)
      if (x==a[i]) return i;
      else return 0;
   }
   else { // Reduce P into a smaller subproblem.
      int mid = (i+l)/2;
      if (x == a[mid]) return mid;
      else if (x < a[mid]) return BinSrch(a,i,mid-1,x);
      else return BinSrch(a,mid+1,l,x);
   }
}
```

Program 3.3 Recursive binary search