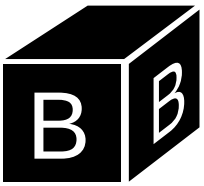


Chapter *20*

Recursive Searching and Sorting

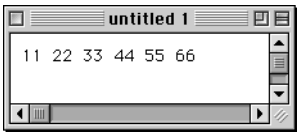


Figure 20.1
The input window and dialog box for testing a binary search algorithm.

```
PROCEDURE Search (IN v: ARRAY OF INTEGER; first, last, srchNum: INTEGER;
        OUT i: INTEGER; OUT fnd: BOOLEAN);
    VAR
        mid: INTEGER;
BEGIN
    ASSERT((0 <= first) & (last < LEN(v)), 20);
```

```
PROCEDURE Search (IN v: ARRAY OF INTEGER; first, last, srchNum: INTEGER;
        OUT i: INTEGER; OUT fnd: BOOLEAN);
    VAR
        mid: INTEGER;
BEGIN
    ASSERT((0 <= first) & (last < LEN(v)), 20);
    IF first > last THEN
        fnd := FALSE
    ELSE
```

```
PROCEDURE Search (IN v: ARRAY OF INTEGER; first, last, srchNum: INTEGER;
        OUT i: INTEGER; OUT fnd: BOOLEAN);
    VAR
        mid: INTEGER;
BEGIN
    ASSERT((0 <= first) & (last < LEN(v)), 20);
    IF first > last THEN
        fnd := FALSE
    ELSE
        mid := (first + last) DIV 2;
        IF srchNum < v[mid] THEN
            Search(v, first, mid - 1, srchNum, i, fnd)
```

```
PROCEDURE Search (IN v: ARRAY OF INTEGER; first, last, srchNum: INTEGER;
      OUT i: INTEGER; OUT fnd: BOOLEAN);
   VAR
      mid: INTEGER;
BEGIN
   ASSERT((0 <= first) & (last < LEN(v)), 20);
   IF first > last THEN
      fnd := FALSE
   ELSE
      mid := (first + last) DIV 2;
      IF srchNum < v[mid] THEN
         Search(v, first, mid - 1, srchNum, i, fnd)
      ELSIF srchNum > v[mid] THEN
         Search(v, mid + 1, last, srchNum, i, fnd)
      ELSE
         fnd := TRUE;
         i := mid
      END
   END
END Search;
```

```
MODULE Pbox20A;
    IMPORT Dialog, TextModels, TextControllers, PboxMappers, PboxStrings;
    VAR
        d*: RECORD
            numItems-: INTEGER;
            searchNumber*: INTEGER;
            indexString-: ARRAY 16 OF CHAR;
        END;
        list: ARRAY 1024 OF INTEGER;

    PROCEDURE LoadList*;
        VAR
            md: TextModels.Model;
            cn: TextControllers.Controller;
            sc: PboxMappers.Scanner;
    BEGIN
        cn := TextControllers.Focus();
        IF cn # NIL THEN
            md := cn.text;
            sc.ConnectTo(md);
            sc.ScanIntVector(list, d.numItems)
        END;
        Dialog.Update(d)
    END LoadList;
```

Figure 20.2
A recursive version of the binary search algorithm.

```
PROCEDURE Search (IN v: ARRAY OF INTEGER; first, last, srchNum: INTEGER;
      OUT i: INTEGER; OUT fnd: BOOLEAN);
   VAR
      mid: INTEGER;
BEGIN
   ASSERT((0 <= first) & (last < LEN(v)), 20);
   IF first > last THEN
      fnd := FALSE
   ELSE
      mid := (first + last) DIV 2;
      IF srchNum < v[mid] THEN
         Search(v, first, mid - 1, srchNum, i, fnd)
      ELSIF srchNum > v[mid] THEN
         Search(v, mid + 1, last, srchNum, i, fnd)
      ELSE
         fnd := TRUE;
         i := mid
      END
   END
END Search;
```

```
    PROCEDURE LookUp*;
        VAR
            j: INTEGER;
            found: BOOLEAN;
    BEGIN
        Search(list, 0, d.numItems - 1, d.searchNumber, j, found);
        IF found THEN
            PboxStrings.IntToString(j, 1, d.indexString)
        ELSE
            d.indexString := "No entry"
        END;
        Dialog.Update(d)
    END LookUp;

BEGIN
    d.numItems := 0;
    d.searchNumber := 0; d.indexString := ""
END Pbox20A.
```

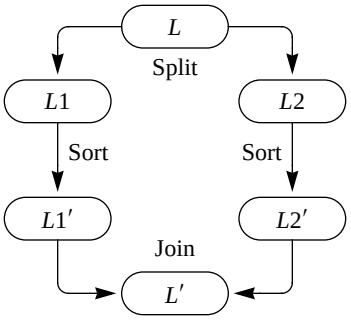
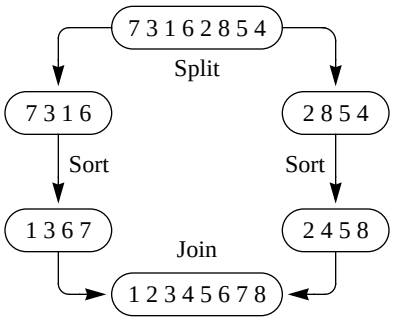
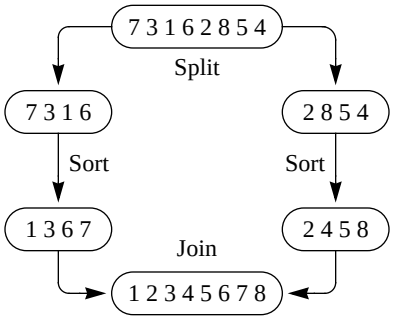


Figure 20.3
The general sort algorithm in the Merritt sort taxonomy.

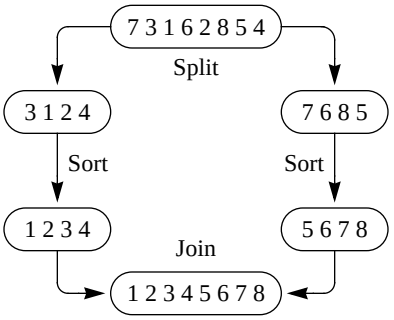


(a) The merge sort algorithm.

Figure 20.4
The two basic sort algorithms in the Merritt sort taxonomy.

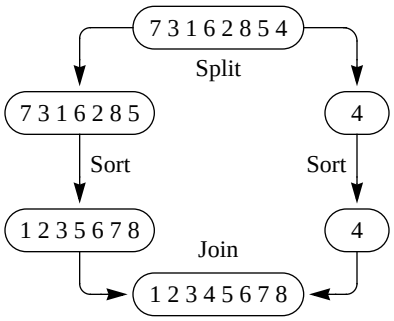


(a) The merge sort algorithm.



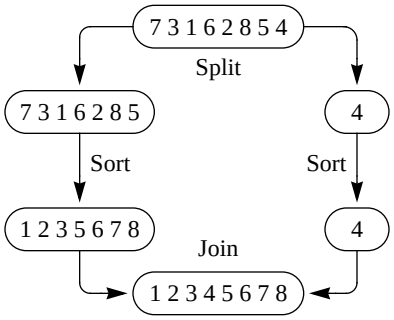
(b) The quick sort algorithm.

Figure 20.4
The two basic sort algorithms in the Merritt sort taxonomy.

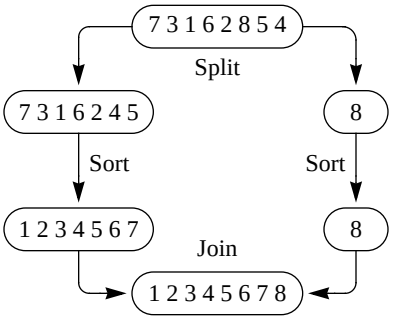


(a) The insertion sort algorithm.

Figure 20.5
Sorting with a split of one element.



(a) The insertion sort algorithm.



(b) The selection sort algorithm.

Figure 20.5
Sorting with a split of one element.

Initial list	7	3	1	6	2	8	5	4
Pass 1, insert 3	3	7	1	6	2	8	5	4
Pass 2, insert 1	1	3	7	6	2	8	5	4
Pass 3, insert 6	1	3	6	7	2	8	5	4
Pass 4, insert 2	1	2	3	6	7	8	5	4
Pass 5, insert 8	1	2	3	6	7	8	5	4
Pass 6, insert 5	1	2	3	5	6	7	8	4
Pass 7, insert 4	1	2	3	4	5	6	7	8

(a) The insertion sort algorithm.

Figure 20.6
Nonrecursive traces of the single element sort algorithms.

Initial list	7	3	1	6	2	8	5	4
Pass 1, insert 3	3	7	1	6	2	8	5	4
Pass 2, insert 1	1	3	7	6	2	8	5	4
Pass 3, insert 6	1	3	6	7	2	8	5	4
Pass 4, insert 2	1	2	3	6	7	8	5	4
Pass 5, insert 8	1	2	3	6	7	8	5	4
Pass 6, insert 5	1	2	3	5	6	7	8	4
Pass 7, insert 4	1	2	3	4	5	6	7	8

(a) The insertion sort algorithm.

Initial list	7	3	1	6	2	8	5	4
Pass 1, select 8	7	3	1	6	2	4	5	8
Pass 2, select 7	5	3	1	6	2	4	7	8
Pass 3, select 6	5	3	1	4	2	6	7	8
Pass 4, select 5	2	3	1	4	5	6	7	8
Pass 5, select 4	2	1	3	4	5	6	7	8
Pass 6, select 3	2	1	3	4	5	6	7	8
Pass 7, select 2	1	2	3	4	5	6	7	8

(b) The selection sort algorithm.

Figure 20.6
Nonrecursive traces of the single element sort algorithms.

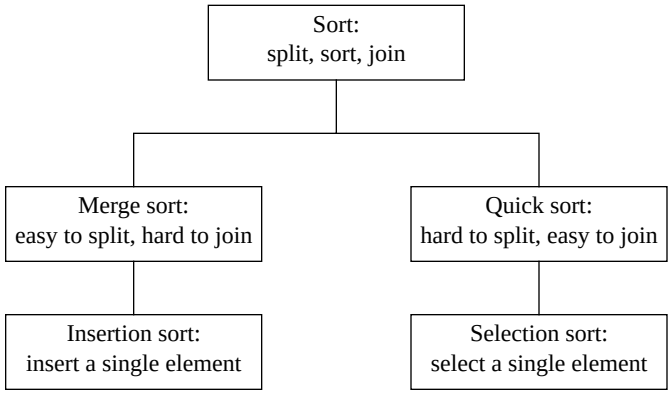
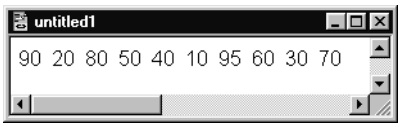
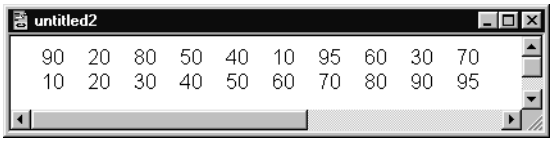


Figure 20.7
Summary of the sort algorithms.





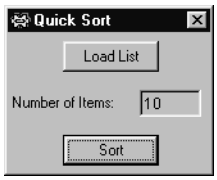


Figure 20.8
The input and output for the quick sort algorithm.

Loop invariant:

- Every element between v[first] and v[i - 1] is less than or equal to every element between v[j + 1] and v[last].

```
PROCEDURE QuickSort (VAR v: ARRAY OF INTEGER; first, last: INTEGER);
    (* Sorts the items of array v between v[first] and v[last]. *)
    VAR
        i, j, key, temp: INTEGER;
BEGIN
    ASSERT((0 <= first) & (first <= last) & (last < LEN(v)) OR (last < 0) & (0 <= first), 20);
    IF first < last THEN
        key := v[(first + last) DIV 2];
        i := first; j := last;
        WHILE i <= j DO
            WHILE v[i] < key DO
                INC(i)
            END;
            WHILE key < v[j] DO
                DEC(j)
            END;
            IF i <= j THEN
                temp := v[j]; v[j] := v[i]; v[i] := temp;
                INC(i); DEC(j)
            END
        END;
        QuickSort (v, first, i - 1);
        QuickSort (v, i, last)
    END
END QuickSort;
```

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
90	20	80	50	40	10	95	60	30	70

first	**last**	**key**
0	9	

```
PROCEDURE QuickSort (VAR v: ARRAY OF INTEGER; first, last: INTEGER);
   (* Sorts the items of array v between v[first] and v[last]. *)
   VAR
      i, j, key, temp: INTEGER;
BEGIN
   ASSERT((0 <= first) & (first <= last) & (last < LEN(v)) OR (last < 0) & (0 <= first), 20);
   IF first < last THEN
      key := v[(first + last) DIV 2];
      i := first; j := last;
      WHILE i <= j DO
         WHILE v[i] < key DO
            INC(i)
         END;
         WHILE key < v[j] DO
            DEC(j)
         END;
         IF i <= j THEN
            temp := v[j]; v[j] := v[i]; v[i] := temp;
            INC(i); DEC(j)
         END
      END;
      QuickSort (v, first, i - 1);
      QuickSort (v, i, last)
   END
END QuickSort;
```

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
90	20	80	50	40	10	95	60	30	70

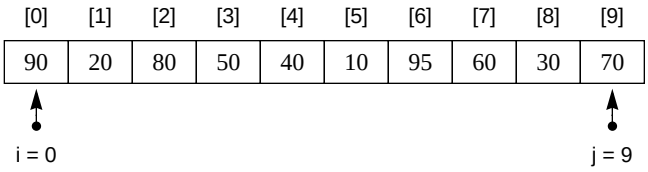
first	**last**	**key**
0	9	

```
PROCEDURE QuickSort (VAR v: ARRAY OF INTEGER; first, last: INTEGER);
    (* Sorts the items of array v between v[first] and v[last]. *)
    VAR
        i, j, key, temp: INTEGER;
BEGIN
    ASSERT((0 <= first) & (first <= last) & (last < LEN(v)) OR (last < 0) & (0 <= first), 20);
    IF first < last THEN
        key := v[(first + last) DIV 2];
        i := first; j := last;
        WHILE i <= j DO
            WHILE v[i] < key DO
                INC(i)
            END;
            WHILE key < v[j] DO
                DEC(j)
            END;
            IF i <= j THEN
                temp := v[j]; v[j] := v[i]; v[i] := temp;
                INC(i); DEC(j)
            END
        END;
        QuickSort (v, first, i - 1);
        QuickSort (v, i, last)
    END
END QuickSort;
```

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
90	20	80	50	40	10	95	60	30	70

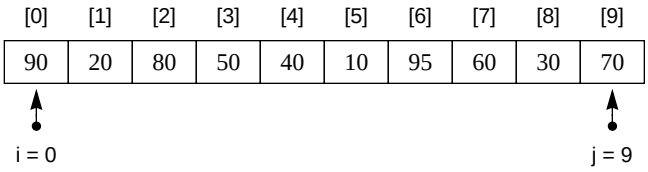
first	**last**	**key**
0	9	40

```
PROCEDURE QuickSort (VAR v: ARRAY OF INTEGER; first, last: INTEGER);
   (* Sorts the items of array v between v[first] and v[last]. *)
   VAR
      i, j, key, temp: INTEGER;
BEGIN
   ASSERT((0 <= first) & (first <= last) & (last < LEN(v)) OR (last < 0) & (0 <= first), 20);
   IF first < last THEN
      key := v[(first + last) DIV 2];
      i := first; j := last;
      WHILE i <= j DO
         WHILE v[i] < key DO
            INC(i)
         END;
         WHILE key < v[j] DO
            DEC(j)
         END;
         IF i <= j THEN
            temp := v[j]; v[j] := v[i]; v[i] := temp;
            INC(i); DEC(j)
         END
      END;
      QuickSort (v, first, i - 1);
      QuickSort (v, i, last)
   END
END QuickSort;
```



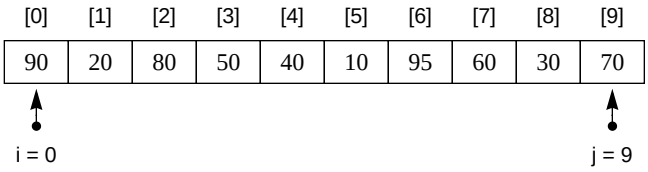
first	**last**	**key**
0	9	40

```
PROCEDURE QuickSort (VAR v: ARRAY OF INTEGER; first, last: INTEGER);
   (* Sorts the items of array v between v[first] and v[last]. *)
   VAR
      i, j, key, temp: INTEGER;
BEGIN
   ASSERT((0 <= first) & (first <= last) & (last < LEN(v)) OR (last < 0) & (0 <= first), 20);
   IF first < last THEN
      key := v[(first + last) DIV 2];
      i := first; j := last;
      WHILE i <= j DO
         WHILE v[i] < key DO
            INC(i)
         END;
         WHILE key < v[j] DO
            DEC(j)
         END;
         IF i <= j THEN
            temp := v[j]; v[j] := v[i]; v[i] := temp;
            INC(i); DEC(j)
         END
      END;
      QuickSort (v, first, i - 1);
      QuickSort (v, i, last)
   END
END QuickSort;
```



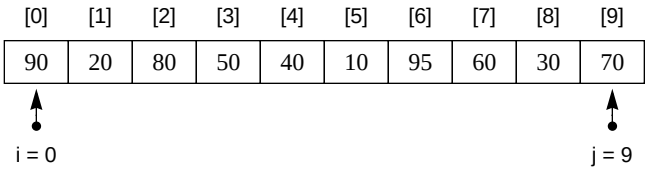
first	**last**	**key**
0	9	40

```
PROCEDURE QuickSort (VAR v: ARRAY OF INTEGER; first, last: INTEGER);
   (* Sorts the items of array v between v[first] and v[last]. *)
   VAR
      i, j, key, temp: INTEGER;
BEGIN
   ASSERT((0 <= first) & (first <= last) & (last < LEN(v)) OR (last < 0) & (0 <= first), 20);
   IF first < last THEN
      key := v[(first + last) DIV 2];
      i := first; j := last;
      WHILE i <= j DO
         WHILE v[i] < key DO
            INC(i)
         END;
         WHILE key < v[j] DO
            DEC(j)
         END;
         IF i <= j THEN
            temp := v[j]; v[j] := v[i]; v[i] := temp;
            INC(i); DEC(j)
         END
      END;
      QuickSort (v, first, i - 1);
      QuickSort (v, i, last)
   END
END QuickSort;
```



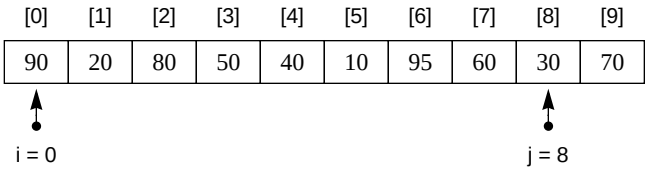
first	**last**	**key**
0	9	40

```
PROCEDURE QuickSort (VAR v: ARRAY OF INTEGER; first, last: INTEGER);
    (* Sorts the items of array v between v[first] and v[last]. *)
    VAR
        i, j, key, temp: INTEGER;
BEGIN
    ASSERT((0 <= first) & (first <= last) & (last < LEN(v)) OR (last < 0) & (0 <= first), 20);
    IF first < last THEN
        key := v[(first + last) DIV 2];
        i := first; j := last;
        WHILE i <= j DO
            WHILE v[i] < key DO
                INC(i)
            END;
            WHILE key < v[j] DO
                DEC(j)
            END;
            IF i <= j THEN
                temp := v[j]; v[j] := v[i]; v[i] := temp;
                INC(i); DEC(j)
            END
        END;
        QuickSort (v, first, i - 1);
        QuickSort (v, i, last)
    END
END QuickSort;
```



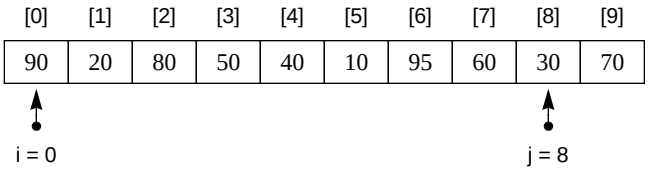
first	**last**	**key**
0	9	40

```
PROCEDURE QuickSort (VAR v: ARRAY OF INTEGER; first, last: INTEGER);
    (* Sorts the items of array v between v[first] and v[last]. *)
    VAR
        i, j, key, temp: INTEGER;
BEGIN
    ASSERT((0 <= first) & (first <= last) & (last < LEN(v)) OR (last < 0) & (0 <= first), 20);
    IF first < last THEN
        key := v[(first + last) DIV 2];
        i := first; j := last;
        WHILE i <= j DO
            WHILE v[i] < key DO
                INC(i)
            END;
            WHILE key < v[j] DO
                DEC(j)
            END;
            IF i <= j THEN
                temp := v[j]; v[j] := v[i]; v[i] := temp;
                INC(i); DEC(j)
            END
        END;
        QuickSort (v, first, i - 1);
        QuickSort (v, i, last)
    END
END QuickSort;
```



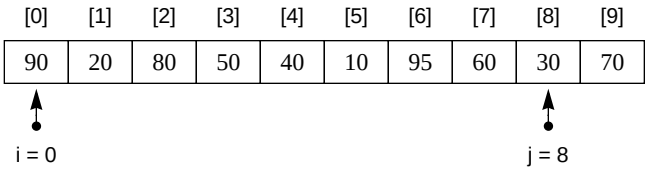
first	**last**	**key**
0	9	40

```
PROCEDURE QuickSort (VAR v: ARRAY OF INTEGER; first, last: INTEGER);
   (* Sorts the items of array v between v[first] and v[last]. *)
   VAR
      i, j, key, temp: INTEGER;
BEGIN
   ASSERT((0 <= first) & (first <= last) & (last < LEN(v)) OR (last < 0) & (0 <= first), 20);
   IF first < last THEN
      key := v[(first + last) DIV 2];
      i := first; j := last;
      WHILE i <= j DO
         WHILE v[i] < key DO
            INC(i)
         END;
         WHILE key < v[j] DO
            DEC(j)
         END;
         IF i <= j THEN
            temp := v[j]; v[j] := v[i]; v[i] := temp;
            INC(i); DEC(j)
         END
      END;
      QuickSort (v, first, i - 1);
      QuickSort (v, i, last)
   END
END QuickSort;
```



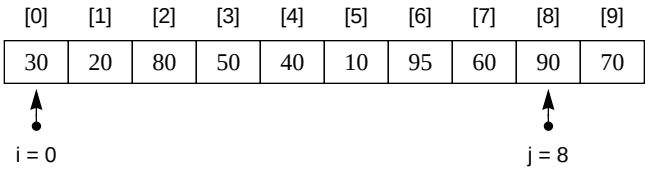
first	**last**	**key**
0	9	40

```
PROCEDURE QuickSort (VAR v: ARRAY OF INTEGER; first, last: INTEGER);
    (* Sorts the items of array v between v[first] and v[last]. *)
    VAR
        i, j, key, temp: INTEGER;
BEGIN
    ASSERT((0 <= first) & (first <= last) & (last < LEN(v)) OR (last < 0) & (0 <= first), 20);
    IF first < last THEN
        key := v[(first + last) DIV 2];
        i := first; j := last;
        WHILE i <= j DO
            WHILE v[i] < key DO
                INC(i)
            END;
            WHILE key < v[j] DO
                DEC(j)
            END;
            IF i <= j THEN
                temp := v[j]; v[j] := v[i]; v[i] := temp;
                INC(i); DEC(j)
            END
        END;
        QuickSort (v, first, i - 1);
        QuickSort (v, i, last)
    END
END QuickSort;
```



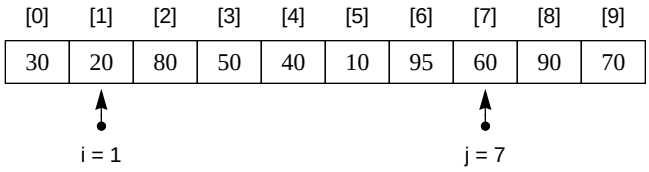
first	**last**	**key**
0	9	40

```
PROCEDURE QuickSort (VAR v: ARRAY OF INTEGER; first, last: INTEGER);
    (* Sorts the items of array v between v[first] and v[last]. *)
    VAR
        i, j, key, temp: INTEGER;
BEGIN
    ASSERT((0 <= first) & (first <= last) & (last < LEN(v)) OR (last < 0) & (0 <= first), 20);
    IF first < last THEN
        key := v[(first + last) DIV 2];
        i := first; j := last;
        WHILE i <= j DO
            WHILE v[i] < key DO
                INC(i)
            END;
            WHILE key < v[j] DO
                DEC(j)
            END;
            IF i <= j THEN
                temp := v[j]; v[j] := v[i]; v[i] := temp;
                INC(i); DEC(j)
            END
        END;
        QuickSort (v, first, i - 1);
        QuickSort (v, i, last)
    END
END QuickSort;
```



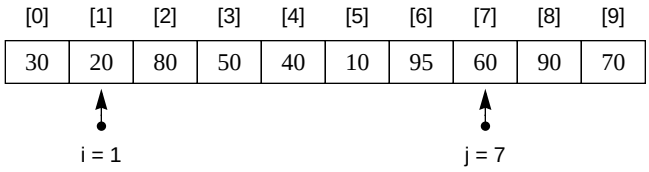
first	**last**	**key**
0	9	40

```
PROCEDURE QuickSort (VAR v: ARRAY OF INTEGER; first, last: INTEGER);
    (* Sorts the items of array v between v[first] and v[last]. *)
    VAR
        i, j, key, temp: INTEGER;
BEGIN
    ASSERT((0 <= first) & (first <= last) & (last < LEN(v)) OR (last < 0) & (0 <= first), 20);
    IF first < last THEN
        key := v[(first + last) DIV 2];
        i := first; j := last;
        WHILE i <= j DO
            WHILE v[i] < key DO
                INC(i)
            END;
            WHILE key < v[j] DO
                DEC(j)
            END;
            IF i <= j THEN
                temp := v[j]; v[j] := v[i]; v[i] := temp;
                INC(i); DEC(j)
            END
        END;
        QuickSort (v, first, i - 1);
        QuickSort (v, i, last)
    END
END QuickSort;
```



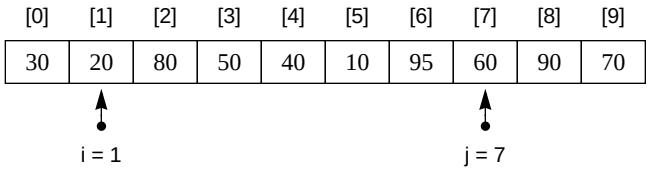
first	**last**	**key**
0	9	40

```
PROCEDURE QuickSort (VAR v: ARRAY OF INTEGER; first, last: INTEGER);
   (* Sorts the items of array v between v[first] and v[last]. *)
   VAR
      i, j, key, temp: INTEGER;
BEGIN
   ASSERT((0 <= first) & (first <= last) & (last < LEN(v)) OR (last < 0) & (0 <= first), 20);
   IF first < last THEN
      key := v[(first + last) DIV 2];
      i := first; j := last;
      WHILE i <= j DO
         WHILE v[i] < key DO
            INC(i)
         END;
         WHILE key < v[j] DO
            DEC(j)
         END;
         IF i <= j THEN
            temp := v[j]; v[j] := v[i]; v[i] := temp;
            INC(i); DEC(j)
         END
      END;
      QuickSort (v, first, i - 1);
      QuickSort (v, i, last)
   END
END QuickSort;
```



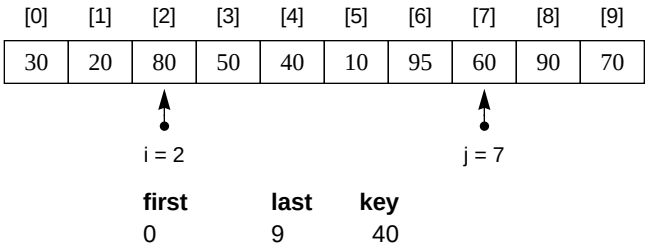
first	**last**	**key**
0	9	40

```
PROCEDURE QuickSort (VAR v: ARRAY OF INTEGER; first, last: INTEGER);
   (* Sorts the items of array v between v[first] and v[last]. *)
   VAR
      i, j, key, temp: INTEGER;
BEGIN
   ASSERT((0 <= first) & (first <= last) & (last < LEN(v)) OR (last < 0) & (0 <= first), 20);
   IF first < last THEN
      key := v[(first + last) DIV 2];
      i := first; j := last;
      WHILE i <= j DO
         WHILE v[i] < key DO
            INC(i)
         END;
         WHILE key < v[j] DO
            DEC(j)
         END;
         IF i <= j THEN
            temp := v[j]; v[j] := v[i]; v[i] := temp;
            INC(i); DEC(j)
         END
      END;
      QuickSort (v, first, i - 1);
      QuickSort (v, i, last)
   END
END QuickSort;
```



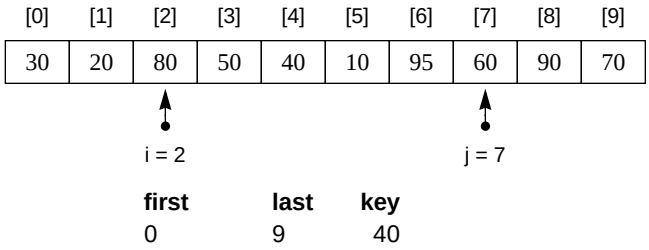
first	**last**	**key**
0	9	40

```
PROCEDURE QuickSort (VAR v: ARRAY OF INTEGER; first, last: INTEGER);
   (* Sorts the items of array v between v[first] and v[last]. *)
   VAR
      i, j, key, temp: INTEGER;
BEGIN
   ASSERT((0 <= first) & (first <= last) & (last < LEN(v)) OR (last < 0) & (0 <= first), 20);
   IF first < last THEN
      key := v[(first + last) DIV 2];
      i := first; j := last;
      WHILE i <= j DO
         WHILE v[i] < key DO
            INC(i)
         END;
         WHILE key < v[j] DO
            DEC(j)
         END;
         IF i <= j THEN
            temp := v[j]; v[j] := v[i]; v[i] := temp;
            INC(i); DEC(j)
         END
      END;
      QuickSort (v, first, i - 1);
      QuickSort (v, i, last)
   END
END QuickSort;
```



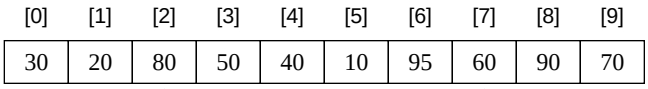
first	**last**	**key**
0	9	40

```
PROCEDURE QuickSort (VAR v: ARRAY OF INTEGER; first, last: INTEGER);
    (* Sorts the items of array v between v[first] and v[last]. *)
    VAR
        i, j, key, temp: INTEGER;
BEGIN
    ASSERT((0 <= first) & (first <= last) & (last < LEN(v)) OR (last < 0) & (0 <= first), 20);
    IF first < last THEN
        key := v[(first + last) DIV 2];
        i := first; j := last;
        WHILE i <= j DO
            WHILE v[i] < key DO
                INC(i)
            END;
            WHILE key < v[j] DO
                DEC(j)
            END;
            IF i <= j THEN
                temp := v[j]; v[j] := v[i]; v[i] := temp;
                INC(i); DEC(j)
            END
        END;
        QuickSort (v, first, i - 1);
        QuickSort (v, i, last)
    END
END QuickSort;
```



first	**last**	**key**
0	9	40

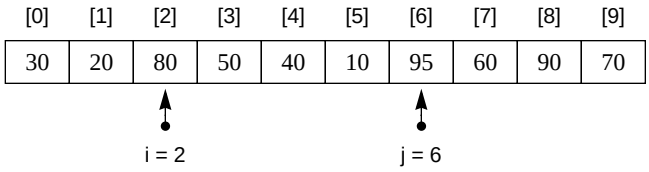
```
PROCEDURE QuickSort (VAR v: ARRAY OF INTEGER; first, last: INTEGER);
    (* Sorts the items of array v between v[first] and v[last]. *)
    VAR
        i, j, key, temp: INTEGER;
BEGIN
    ASSERT((0 <= first) & (first <= last) & (last < LEN(v)) OR (last < 0) & (0 <= first), 20);
    IF first < last THEN
        key := v[(first + last) DIV 2];
        i := first; j := last;
        WHILE i <= j DO
            WHILE v[i] < key DO
                INC(i)
            END;
            WHILE key < v[j] DO
                DEC(j)
            END;
            IF i <= j THEN
                temp := v[j]; v[j] := v[i]; v[i] := temp;
                INC(i); DEC(j)
            END
        END;
        QuickSort (v, first, i - 1);
        QuickSort (v, i, last)
    END
END QuickSort;
```



i = 2 j = 7

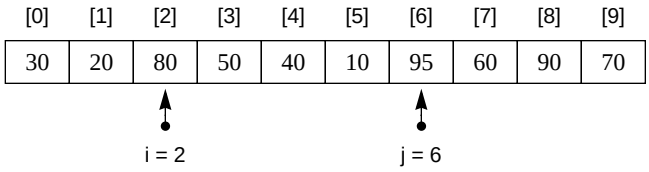
first	**last**	**key**
0	9	40

```
PROCEDURE QuickSort (VAR v: ARRAY OF INTEGER; first, last: INTEGER);
    (* Sorts the items of array v between v[first] and v[last]. *)
    VAR
        i, j, key, temp: INTEGER;
BEGIN
    ASSERT((0 <= first) & (first <= last) & (last < LEN(v)) OR (last < 0) & (0 <= first), 20);
    IF first < last THEN
        key := v[(first + last) DIV 2];
        i := first; j := last;
        WHILE i <= j DO
            WHILE v[i] < key DO
                INC(i)
            END;
            WHILE key < v[j] DO
                DEC(j)
            END;
            IF i <= j THEN
                temp := v[j]; v[j] := v[i]; v[i] := temp;
                INC(i); DEC(j)
            END
        END;
        QuickSort (v, first, i - 1);
        QuickSort (v, i, last)
    END
END QuickSort;
```



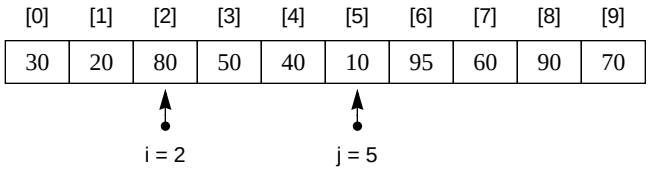
first	**last**	**key**
0	9	40

```
PROCEDURE QuickSort (VAR v: ARRAY OF INTEGER; first, last: INTEGER);
    (* Sorts the items of array v between v[first] and v[last]. *)
    VAR
        i, j, key, temp: INTEGER;
BEGIN
    ASSERT((0 <= first) & (first <= last) & (last < LEN(v)) OR (last < 0) & (0 <= first), 20);
    IF first < last THEN
        key := v[(first + last) DIV 2];
        i := first; j := last;
        WHILE i <= j DO
            WHILE v[i] < key DO
                INC(i)
            END;
            WHILE key < v[j] DO
                DEC(j)
            END;
            IF i <= j THEN
                temp := v[j]; v[j] := v[i]; v[i] := temp;
                INC(i); DEC(j)
            END
        END;
        QuickSort (v, first, i - 1);
        QuickSort (v, i, last)
    END
END QuickSort;
```



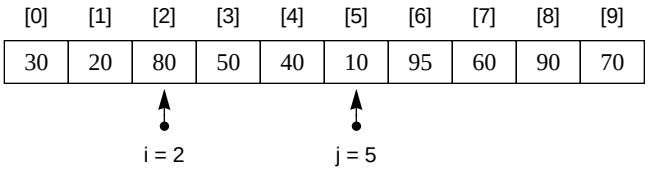
first	**last**	**key**
0	9	40

```
PROCEDURE QuickSort (VAR v: ARRAY OF INTEGER; first, last: INTEGER);
    (* Sorts the items of array v between v[first] and v[last]. *)
    VAR
        i, j, key, temp: INTEGER;
BEGIN
    ASSERT((0 <= first) & (first <= last) & (last < LEN(v)) OR (last < 0) & (0 <= first), 20);
    IF first < last THEN
        key := v[(first + last) DIV 2];
        i := first; j := last;
        WHILE i <= j DO
            WHILE v[i] < key DO
                INC(i)
            END;
            WHILE key < v[j] DO
                DEC(j)
            END;
            IF i <= j THEN
                temp := v[j]; v[j] := v[i]; v[i] := temp;
                INC(i); DEC(j)
            END
        END;
        QuickSort (v, first, i - 1);
        QuickSort (v, i, last)
    END
END QuickSort;
```



first	**last**	**key**
0	9	40

```
PROCEDURE QuickSort (VAR v: ARRAY OF INTEGER; first, last: INTEGER);
    (* Sorts the items of array v between v[first] and v[last]. *)
    VAR
        i, j, key, temp: INTEGER;
BEGIN
    ASSERT((0 <= first) & (first <= last) & (last < LEN(v)) OR (last < 0) & (0 <= first), 20);
    IF first < last THEN
        key := v[(first + last) DIV 2];
        i := first; j := last;
        WHILE i <= j DO
            WHILE v[i] < key DO
                INC(i)
            END;
            WHILE key < v[j] DO
                DEC(j)
            END;
            IF i <= j THEN
                temp := v[j]; v[j] := v[i]; v[i] := temp;
                INC(i); DEC(j)
            END
        END;
        QuickSort (v, first, i - 1);
        QuickSort (v, i, last)
    END
END QuickSort;
```



first	**last**	**key**
0	9	40

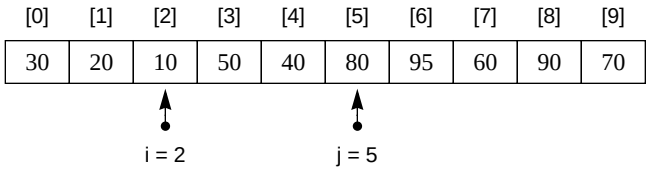
```
PROCEDURE QuickSort (VAR v: ARRAY OF INTEGER; first, last: INTEGER);
    (* Sorts the items of array v between v[first] and v[last]. *)
    VAR
        i, j, key, temp: INTEGER;
BEGIN
    ASSERT((0 <= first) & (first <= last) & (last < LEN(v)) OR (last < 0) & (0 <= first), 20);
    IF first < last THEN
        key := v[(first + last) DIV 2];
        i := first; j := last;
        WHILE i <= j DO
            WHILE v[i] < key DO
                INC(i)
            END;
            WHILE key < v[j] DO
                DEC(j)
            END;
            IF i <= j THEN
                temp := v[j]; v[j] := v[i]; v[i] := temp;
                INC(i); DEC(j)
            END
        END;
        QuickSort (v, first, i - 1);
        QuickSort (v, i, last)
    END
END QuickSort;
```

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
30	20	80	50	40	10	95	60	90	70

i = 2 j = 5

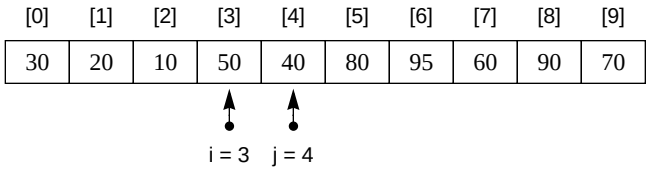
first	**last**	**key**
0	9	40

```
PROCEDURE QuickSort (VAR v: ARRAY OF INTEGER; first, last: INTEGER);
   (* Sorts the items of array v between v[first] and v[last]. *)
   VAR
      i, j, key, temp: INTEGER;
BEGIN
   ASSERT((0 <= first) & (first <= last) & (last < LEN(v)) OR (last < 0) & (0 <= first), 20);
   IF first < last THEN
      key := v[(first + last) DIV 2];
      i := first; j := last;
      WHILE i <= j DO
         WHILE v[i] < key DO
            INC(i)
         END;
         WHILE key < v[j] DO
            DEC(j)
         END;
         IF i <= j THEN
            temp := v[j]; v[j] := v[i]; v[i] := temp;
            INC(i); DEC(j)
         END
      END;
      QuickSort (v, first, i - 1);
      QuickSort (v, i, last)
   END
END QuickSort;
```



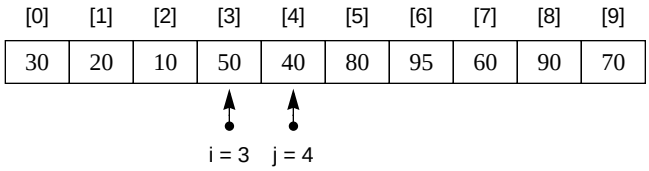
first	**last**	**key**
0	9	40

```
PROCEDURE QuickSort (VAR v: ARRAY OF INTEGER; first, last: INTEGER);
    (* Sorts the items of array v between v[first] and v[last]. *)
    VAR
        i, j, key, temp: INTEGER;
BEGIN
    ASSERT((0 <= first) & (first <= last) & (last < LEN(v)) OR (last < 0) & (0 <= first), 20);
    IF first < last THEN
        key := v[(first + last) DIV 2];
        i := first; j := last;
        WHILE i <= j DO
            WHILE v[i] < key DO
                INC(i)
            END;
            WHILE key < v[j] DO
                DEC(j)
            END;
            IF i <= j THEN
                temp := v[j]; v[j] := v[i]; v[i] := temp;
                INC(i); DEC(j)
            END
        END;
        QuickSort (v, first, i - 1);
        QuickSort (v, i, last)
    END
END QuickSort;
```



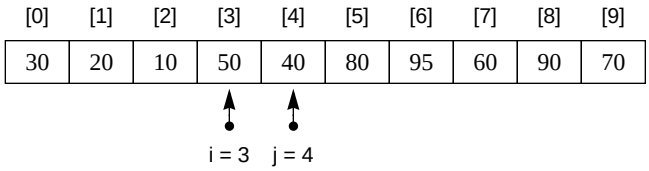
first	**last**	**key**
0	9	40

```
PROCEDURE QuickSort (VAR v: ARRAY OF INTEGER; first, last: INTEGER);
   (* Sorts the items of array v between v[first] and v[last]. *)
   VAR
      i, j, key, temp: INTEGER;
BEGIN
   ASSERT((0 <= first) & (first <= last) & (last < LEN(v)) OR (last < 0) & (0 <= first), 20);
   IF first < last THEN
      key := v[(first + last) DIV 2];
      i := first; j := last;
      WHILE i <= j DO
         WHILE v[i] < key DO
            INC(i)
         END;
         WHILE key < v[j] DO
            DEC(j)
         END;
         IF i <= j THEN
            temp := v[j]; v[j] := v[i]; v[i] := temp;
            INC(i); DEC(j)
         END
      END;
      QuickSort (v, first, i - 1);
      QuickSort (v, i, last)
   END
END QuickSort;
```



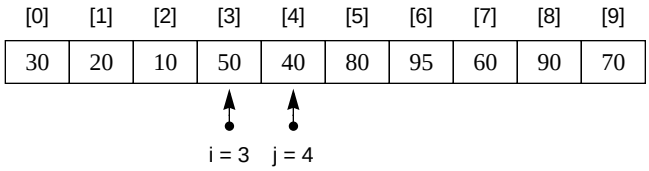
first	**last**	**key**
0	9	40

```
PROCEDURE QuickSort (VAR v: ARRAY OF INTEGER; first, last: INTEGER);
   (* Sorts the items of array v between v[first] and v[last]. *)
   VAR
      i, j, key, temp: INTEGER;
BEGIN
   ASSERT((0 <= first) & (first <= last) & (last < LEN(v)) OR (last < 0) & (0 <= first), 20);
   IF first < last THEN
      key := v[(first + last) DIV 2];
      i := first; j := last;
      WHILE i <= j DO
         WHILE v[i] < key DO
            INC(i)
         END;
         WHILE key < v[j] DO
            DEC(j)
         END;
         IF i <= j THEN
            temp := v[j]; v[j] := v[i]; v[i] := temp;
            INC(i); DEC(j)
         END
      END;
      QuickSort (v, first, i - 1);
      QuickSort (v, i, last)
   END
END QuickSort;
```



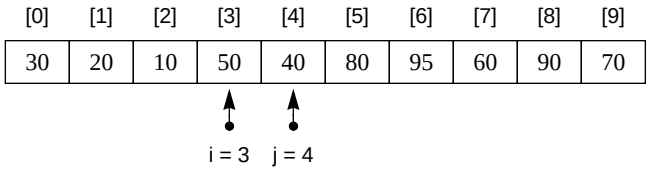
first	**last**	**key**
0	9	40

```
PROCEDURE QuickSort (VAR v: ARRAY OF INTEGER; first, last: INTEGER);
   (* Sorts the items of array v between v[first] and v[last]. *)
   VAR
      i, j, key, temp: INTEGER;
BEGIN
   ASSERT((0 <= first) & (first <= last) & (last < LEN(v)) OR (last < 0) & (0 <= first), 20);
   IF first < last THEN
      key := v[(first + last) DIV 2];
      i := first; j := last;
      WHILE i <= j DO
         WHILE v[i] < key DO
            INC(i)
         END;
         WHILE key < v[j] DO
            DEC(j)
         END;
         IF i <= j THEN
            temp := v[j]; v[j] := v[i]; v[i] := temp;
            INC(i); DEC(j)
         END
      END;
      QuickSort (v, first, i - 1);
      QuickSort (v, i, last)
   END
END QuickSort;
```



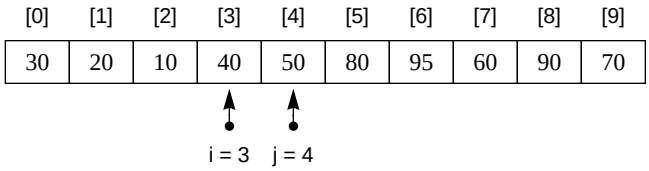
first	**last**	**key**
0	9	40

```
PROCEDURE QuickSort (VAR v: ARRAY OF INTEGER; first, last: INTEGER);
   (* Sorts the items of array v between v[first] and v[last]. *)
   VAR
      i, j, key, temp: INTEGER;
BEGIN
   ASSERT((0 <= first) & (first <= last) & (last < LEN(v)) OR (last < 0) & (0 <= first), 20);
   IF first < last THEN
      key := v[(first + last) DIV 2];
      i := first; j := last;
      WHILE i <= j DO
         WHILE v[i] < key DO
            INC(i)
         END;
         WHILE key < v[j] DO
            DEC(j)
         END;
         IF i <= j THEN
            temp := v[j]; v[j] := v[i]; v[i] := temp;
            INC(i); DEC(j)
         END
      END;
      QuickSort (v, first, i - 1);
      QuickSort (v, i, last)
   END
END QuickSort;
```



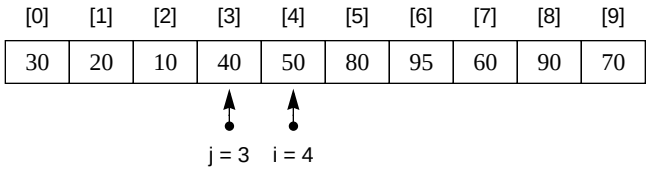
first	**last**	**key**
0	9	40

```
PROCEDURE QuickSort (VAR v: ARRAY OF INTEGER; first, last: INTEGER);
   (* Sorts the items of array v between v[first] and v[last]. *)
   VAR
      i, j, key, temp: INTEGER;
BEGIN
   ASSERT((0 <= first) & (first <= last) & (last < LEN(v)) OR (last < 0) & (0 <= first), 20);
   IF first < last THEN
      key := v[(first + last) DIV 2];
      i := first; j := last;
      WHILE i <= j DO
         WHILE v[i] < key DO
            INC(i)
         END;
         WHILE key < v[j] DO
            DEC(j)
         END;
         IF i <= j THEN
            temp := v[j]; v[j] := v[i]; v[i] := temp;
            INC(i); DEC(j)
         END
      END;
      QuickSort (v, first, i - 1);
      QuickSort (v, i, last)
   END
END QuickSort;
```



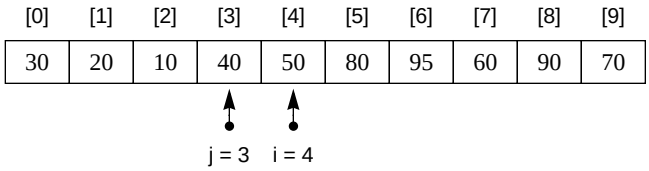
first	**last**	**key**
0	9	40

```
PROCEDURE QuickSort (VAR v: ARRAY OF INTEGER; first, last: INTEGER);
    (* Sorts the items of array v between v[first] and v[last]. *)
    VAR
        i, j, key, temp: INTEGER;
BEGIN
    ASSERT((0 <= first) & (first <= last) & (last < LEN(v)) OR (last < 0) & (0 <= first), 20);
    IF first < last THEN
        key := v[(first + last) DIV 2];
        i := first; j := last;
        WHILE i <= j DO
            WHILE v[i] < key DO
                INC(i)
            END;
            WHILE key < v[j] DO
                DEC(j)
            END;
            IF i <= j THEN
                temp := v[j]; v[j] := v[i]; v[i] := temp;
                INC(i); DEC(j)
            END
        END;
        QuickSort (v, first, i - 1);
        QuickSort (v, i, last)
    END
END QuickSort;
```



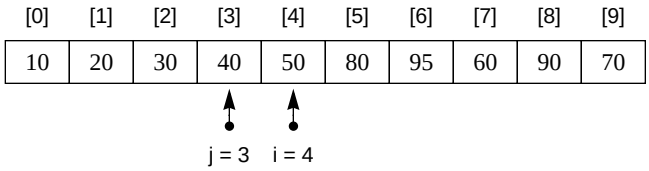
first	**last**	**key**
0	9	40

```
PROCEDURE QuickSort (VAR v: ARRAY OF INTEGER; first, last: INTEGER);
   (* Sorts the items of array v between v[first] and v[last]. *)
   VAR
      i, j, key, temp: INTEGER;
BEGIN
   ASSERT((0 <= first) & (first <= last) & (last < LEN(v)) OR (last < 0) & (0 <= first), 20);
   IF first < last THEN
      key := v[(first + last) DIV 2];
      i := first; j := last;
      WHILE i <= j DO
         WHILE v[i] < key DO
            INC(i)
         END;
         WHILE key < v[j] DO
            DEC(j)
         END;
         IF i <= j THEN
            temp := v[j]; v[j] := v[i]; v[i] := temp;
            INC(i); DEC(j)
         END
      END;
      QuickSort (v, first, i - 1);
      QuickSort (v, i, last)
   END
END QuickSort;
```



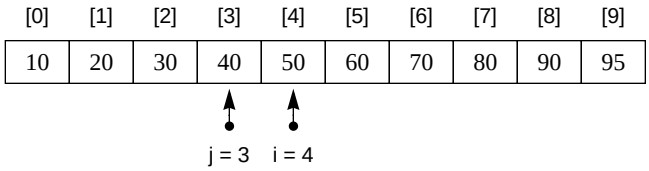
first	last	key
0	9	40

```
PROCEDURE QuickSort (VAR v: ARRAY OF INTEGER; first, last: INTEGER);
    (* Sorts the items of array v between v[first] and v[last]. *)
    VAR
        i, j, key, temp: INTEGER;
BEGIN
    ASSERT((0 <= first) & (first <= last) & (last < LEN(v)) OR (last < 0) & (0 <= first), 20);
    IF first < last THEN
        key := v[(first + last) DIV 2];
        i := first; j := last;
        WHILE i <= j DO
            WHILE v[i] < key DO
                INC(i)
            END;
            WHILE key < v[j] DO
                DEC(j)
            END;
            IF i <= j THEN
                temp := v[j]; v[j] := v[i]; v[i] := temp;
                INC(i); DEC(j)
            END
        END;
        QuickSort (v, first, i - 1);
        QuickSort (v, i, last)
    END
END QuickSort;
```



first	last	key
0	9	40

```
PROCEDURE QuickSort (VAR v: ARRAY OF INTEGER; first, last: INTEGER);
   (* Sorts the items of array v between v[first] and v[last]. *)
   VAR
      i, j, key, temp: INTEGER;
BEGIN
   ASSERT((0 <= first) & (first <= last) & (last < LEN(v)) OR (last < 0) & (0 <= first), 20);
   IF first < last THEN
      key := v[(first + last) DIV 2];
      i := first; j := last;
      WHILE i <= j DO
         WHILE v[i] < key DO
            INC(i)
         END;
         WHILE key < v[j] DO
            DEC(j)
         END;
         IF i <= j THEN
            temp := v[j]; v[j] := v[i]; v[i] := temp;
            INC(i); DEC(j)
         END
      END;
      QuickSort (v, first, i - 1);
      QuickSort (v, i, last)
   END
END QuickSort;
```



first	last	key
0	9	40

```
MODULE Pbox20B;
    IMPORT Dialog, TextModels, TextViews, Views, TextControllers, PboxMappers;
    VAR
        d*: RECORD
            numItems-: INTEGER;
        END;
        list: ARRAY 1024 OF INTEGER;

    PROCEDURE LoadList*;
        VAR
            md: TextModels.Model;
            cn: TextControllers.Controller;
            sc: PboxMappers.Scanner;
    BEGIN
        cn := TextControllers.Focus();
        IF cn # NIL THEN
            md := cn.text;
            sc.ConnectTo(md);
            sc.ScanIntVector(list, d.numItems);
            Dialog.Update(d)
        END;
    END LoadList;
```

Figure 20.9
An implementation of the quick sort algorithm.

```
PROCEDURE QuickSort (VAR v: ARRAY OF INTEGER; first, last: INTEGER);
    (* Sorts the items of array v between v[first] and v[last]. *)
    VAR
        i, j: INTEGER;
        key: INTEGER;
        temp: INTEGER;
```

```
BEGIN
   ASSERT((0 <= first) & (first <= last) & (last < LEN(v)) OR (last < 0) & (0 <= first), 20);
   IF first < last THEN
      key := v[(first + last) DIV 2];
      i := first;
      j := last;
      (* Invariant 1: key <= v[j + 1..last]. *)
      (* Invariant 2: v[first..i - 1] <= key. *)
      (* Invariant 3: if i <= j, there exists k in [first..j] such that v[k] <= key. *)
      (* Invariant 4: if i <= j, there exists k in [i..last] such that key <= v[k]. *)
      WHILE i <= j DO
         WHILE v[i] < key DO
            INC(i)
         END;
         WHILE key < v[j] DO
            DEC(j)
         END;
         IF i <= j THEN
            temp := v[j];
            v[j] := v[i];  (* Establish invariant 4. *)
            v[i] := temp;  (* Establish invariant 3. *)
            INC(i);  (* Establish invariant 2. *)
            DEC(j)  (* Establish invariant 1. *)
         END
      END;
      QuickSort (v, first, i - 1);
      QuickSort (v, i, last)
   END
END QuickSort;
```

```
    PROCEDURE SortList*;
        VAR
            md: TextModels.Model;
            vw: TextViews.View;
            fm: PboxMappers.Formatter;
    BEGIN
        md := TextModels.dir.New();
        fm.ConnectTo(md);
        fm.WriteIntVector(list, d.numItems, 4); fm.WriteLn;
        QuickSort(list, 0, d.numItems - 1);
        fm.WriteIntVector(list, d.numItems, 4); fm.WriteLn;
        vw := TextViews.dir.New(md);
        Views.OpenView(vw)
    END SortList;

BEGIN
    d.numItems := 0
END Pbox20B.
```

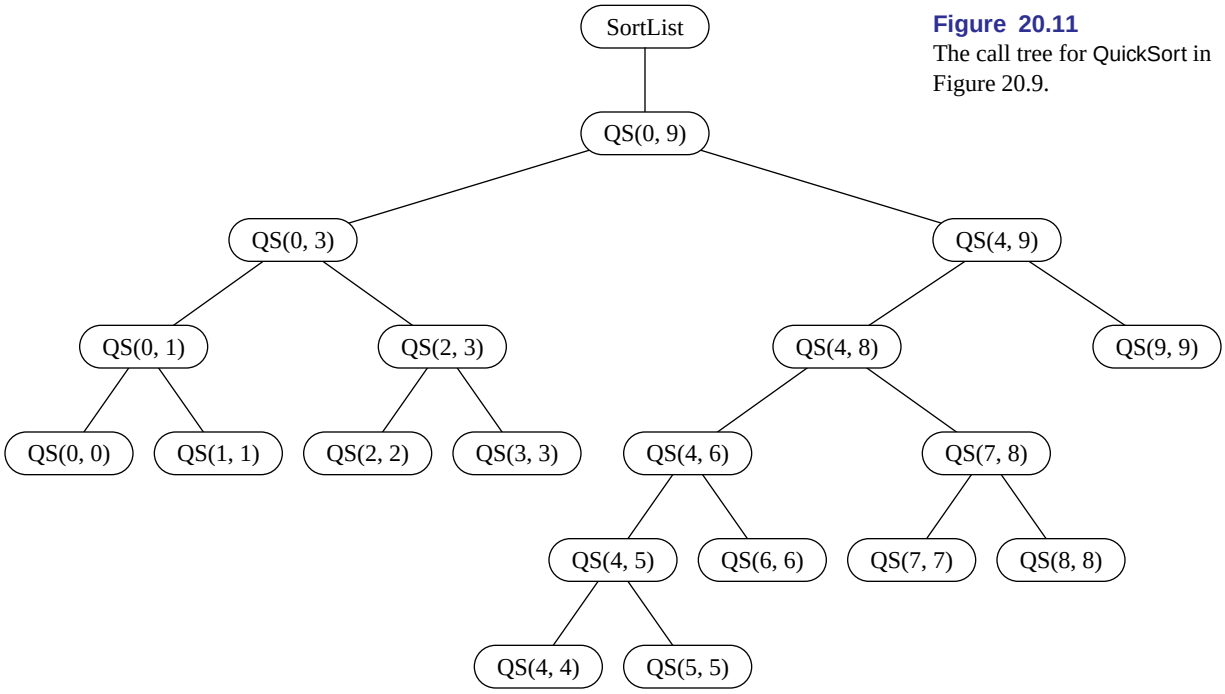


Figure 20.11
The call tree for QuickSort in Figure 20.9.

```
procedure QuickSort(v, first, last);
   if first < last →
      key := v[(first + last) div 2]; i := first; j := last;
      {(∀k | j + 1 ≤ k ≤ last : key ≤ v[k]) ∧
      (∀k | first ≤ k ≤ i – 1 : v[k] ≤ key) ∧
      (i ≤ j ⇒ (∃k | first ≤ k ≤ j : v[k] ≤ key)) ∧
      (i ≤ j ⇒ (∃k | i ≤ k ≤ last : key ≤ v[k]))}
      do i ≤ j →
         do v[i] < key → i := i + 1 od;
         do key < v[j] → j := j – 1 od;
         if i ≤ j → v[i], v[j] := v[j], v[i]; i, j := i + 1, j – 1
          [] i > j → skip
         fi
      od;
      QuickSort(v, first, i – 1);
      QuickSort(v, i, last)
    [] first ≥ last → skip
   fi
end QuickSort
```

```
PROCEDURE MergeSort (VAR v: ARRAY OF INTEGER; first, last: INTEGER);
    (* Sorts the items of array v between v[first] and v[last]. *)
    VAR
        i, j, k, mid: INTEGER;
        temp: ARRAY 128 OF INTEGER;
BEGIN
    ASSERT((0 <= first) & (first <= last) & (last < LEN(v)) OR (last < 0) & (0 <= first), 20);
    IF first < last THEN
        mid := (first + last) DIV 2;
        MergeSort(v, first, mid);
        MergeSort(v, mid + 1, last);
        (* Problem for the student to join v[first..mid] and v[mid + 1..last] *)
    END
END MergeSort;
```

Figure 20.12
An implementation of the merge sort algorithm.

(a) first = 0
last = 9
mid := 4

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
90	20	80	50	40	10	95	60	30	70

Figure 20.13
A trace of the top-level call to MergeSort in Figure 20.12.

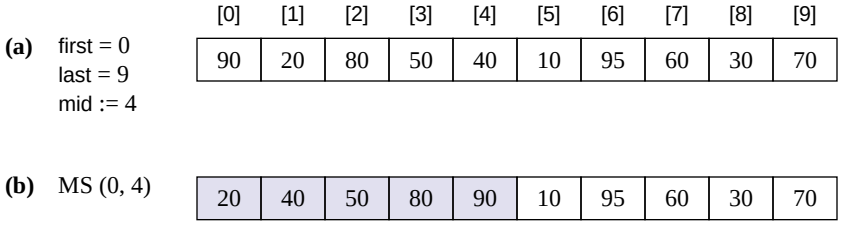


Figure 20.13
A trace of the top-level call to MergeSort in Figure 20.12.

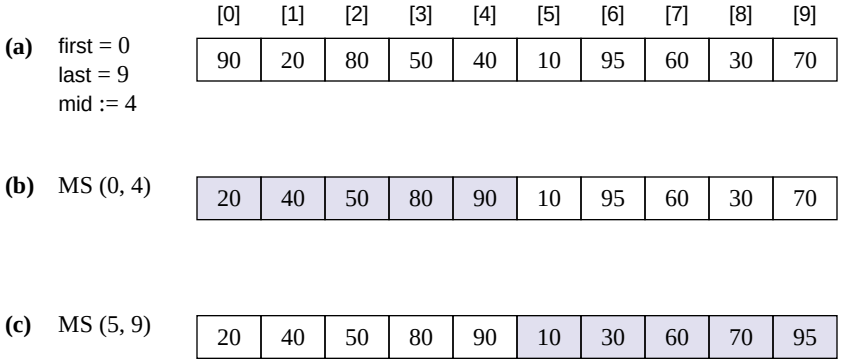


Figure 20.13
A trace of the top-level call to MergeSort in Figure 20.12.

(a) first = 0
last = 9
mid := 4

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
90	20	80	50	40	10	95	60	30	70

(b) MS (0, 4)

20	40	50	80	90	10	95	60	30	70

(c) MS (5, 9)

20	40	50	80	90	10	30	60	70	95

(d) Problem for the student.

10	20	30	40	50	60	75	80	90	95

Figure 20.13
A trace of the top-level call to MergeSort in Figure 20.12.

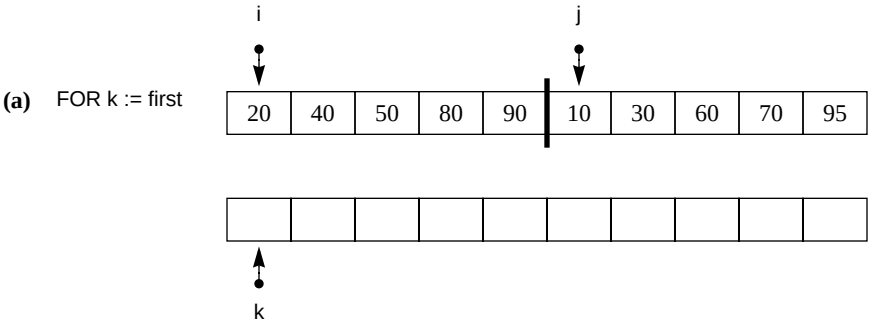


Figure 20.14
A trace of the top-level call to MergeSort in Figure 20.12. Seven steps are not shown between parts (c) and (d).

(b) temp[k] := v[j];
INC(j)

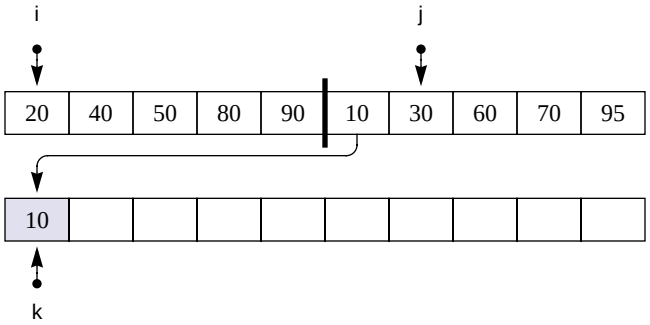


Figure 20.14
A trace of the top-level call to MergeSort in Figure 20.12. Seven steps are not shown between parts (c) and (d).

(c) temp[k] := v[i];
INC(i)

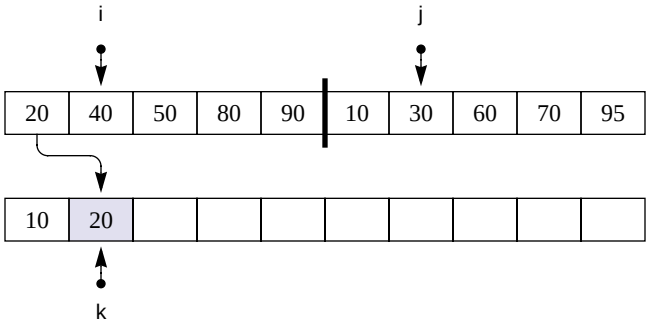


Figure 20.14
A trace of the top-level call to MergeSort in Figure 20.12. Seven steps are not shown between parts (c) and (d).

etc.

Figure 20.14
A trace of the top-level call to MergeSort in Figure 20.12. Seven steps are not shown between parts (c) and (d).

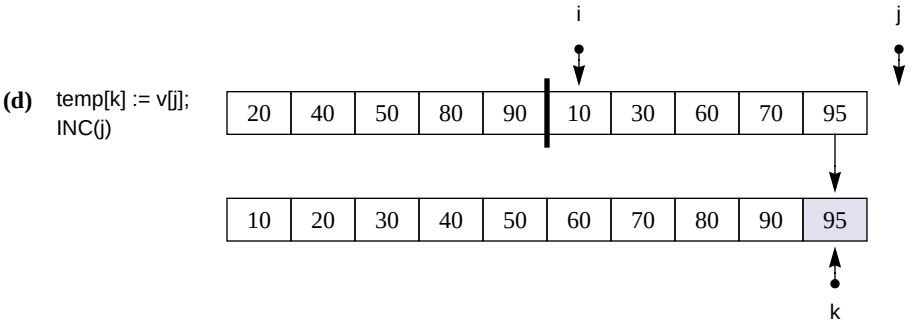


Figure 20.14
A trace of the top-level call to MergeSort in Figure 20.12. Seven steps are not shown between parts (c) and (d).

(e) Copy array temp to array v.

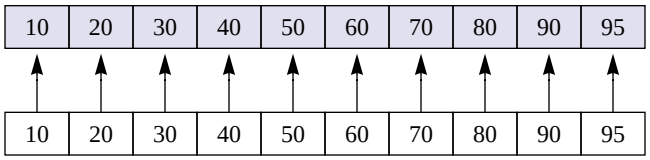


Figure 20.14
A trace of the top-level call to MergeSort in Figure 20.12. Seven steps are not shown between parts (c) and (d).

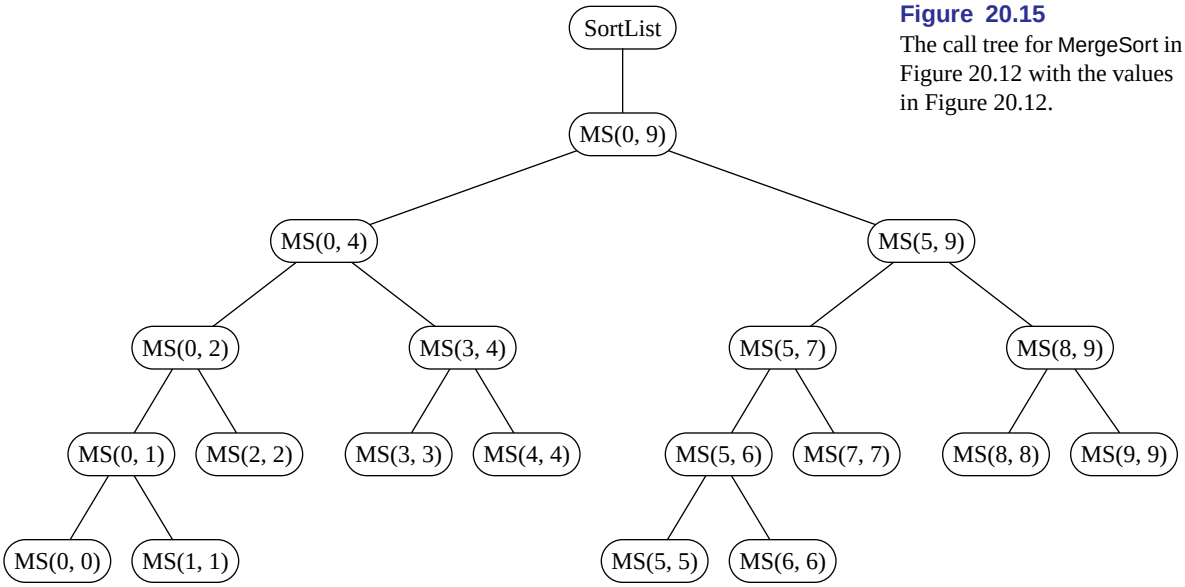


Figure 20.15
The call tree for MergeSort in Figure 20.12 with the values in Figure 20.12.

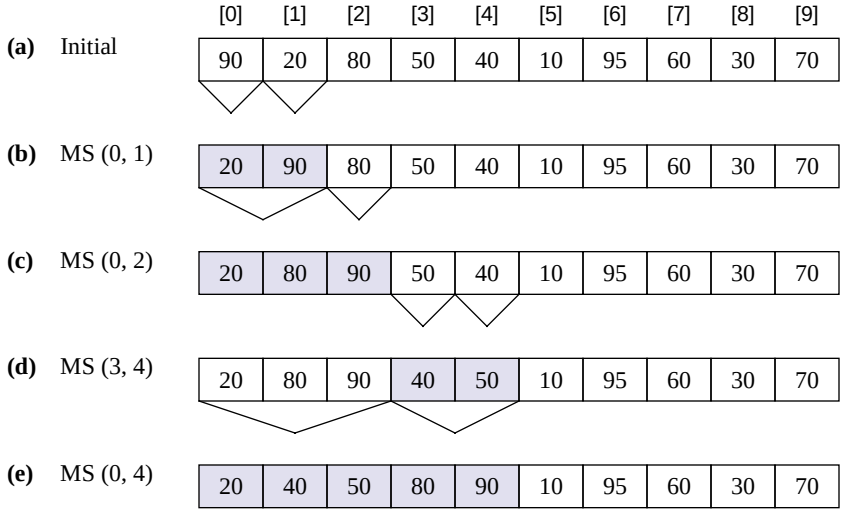


Figure 20.16
The merges in MergeSort in the order they occur for the left half of the array.

```
MODULE Pbox20C;
    IMPORT Dialog, TextModels, TextViews, Views, TextControllers, PboxMappers;
    TYPE
        Item = RECORD
            value: INTEGER;
            link: INTEGER
        END;
    VAR
        d*: RECORD
            numItems-: INTEGER;
        END;
        list: ARRAY 1024 OF Item;
```

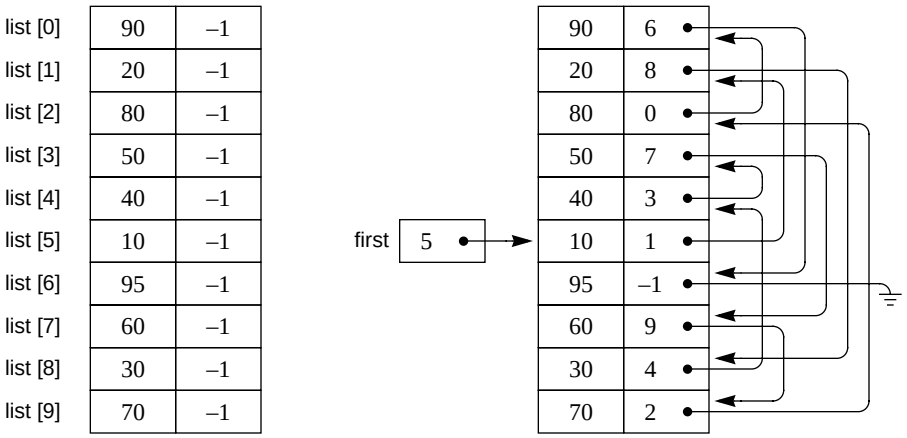
Figure 20.17
An implementation of the in-place merge sort algorithm.

```
PROCEDURE LoadList*;
    VAR
        md: TextModels.Model;
        cn: TextControllers.Controller;
        sc: PboxMappers.Scanner;
        i: INTEGER;
BEGIN
    cn := TextControllers.Focus();
    IF cn # NIL THEN
        md := cn.text;
        sc.ConnectTo(md);
        i := 0;
        sc.ScanInt(list[i].value); list[i].link := -1;
        WHILE ~sc.eot DO
            INC(i);
            sc.ScanInt(list[i].value); list[i].link := -1
        END;
        d.numItems := i;
        Dialog.Update(d)
    END;
END LoadList;
```

```
PROCEDURE MergeSort (VAR v: ARRAY OF Item; first, last: INTEGER; OUT start: INTEGER);
    (* Sorts the items of array v between v[first] and v[last]. *)
    VAR
        mid, loStart, hiStart: INTEGER;
        i, j, k: INTEGER;
BEGIN
    ASSERT((0 <= first) & (first <= last) & (last < LEN(v) - 1), 20);
    IF first = last THEN
        start := first
    ELSE
        mid := (first + last) DIV 2;
        MergeSort(v, first, mid, loStart);
        MergeSort(v, mid + 1, last, hiStart);
        i := loStart;
        j := hiStart;
        k := LEN(v) - 1; (* Temporary start of merged list *)
        WHILE (i # -1) & (j # -1) DO
            IF v[i].value <= v[j].value THEN
                v[k].link := i;
                k := i;
                i := v[i].link
            ELSE
                v[k].link := j;
                k := j;
                j := v[j].link
            END
        END;
        IF i = -1 THEN
            v[k].link := j  (* Attach remainder of last list *)
        ELSE
            v[k].link := i  (* Attach remainder of first list *)
        END;
        start := v[LEN(v) - 1].link
    END
END MergeSort;
```

```
    PROCEDURE SortList*;
        VAR
            md: TextModels.Model;
            vw: TextViews.View;
            fm: PboxMappers.Formatter;
            i: INTEGER;
            first: INTEGER;
    BEGIN
        md := TextModels.dir.New();
        fm.ConnectTo(md);
        FOR i := 0 TO d.numItems -1 DO
            fm.WriteInt(list[i].value, 4)
        END;
        fm.WriteLn;
        IF d.numItems > 0 THEN
            MergeSort(list, 0, d.numItems - 1, first);
            i := first;
            WHILE i # -1 DO
                fm.WriteInt(list[i].value, 4);
                i := list[i].link
            END
        END;
        vw := TextViews.dir.New(md);
        Views.OpenView(vw)
    END SortList;

BEGIN
    d.numItems := 0
END Pbox20C.
```



(a) Before the first MergeSort call. **(b)** After the top-level merge

Figure 20.18
The result of a MergeSort call from procedure SortList of Figure 20.17.

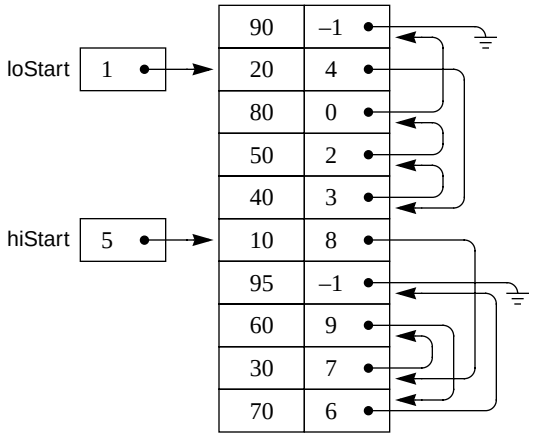
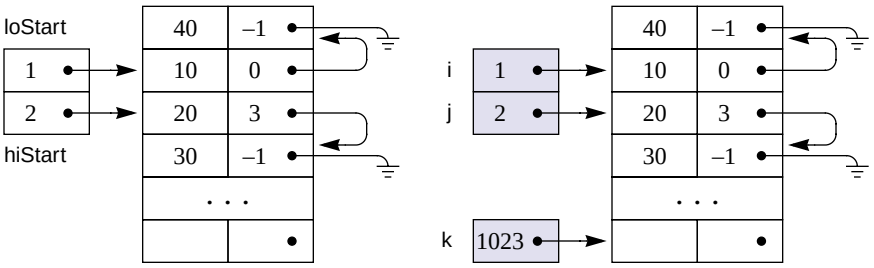


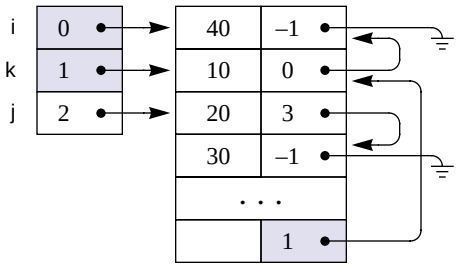
Figure 20.19
The list in the call to MergeSort(v, 0, 9, first) after the recursive calls to MergeSort(v, 0, 4, loStart) and MergeSort(v, 5, 9, hiStart).



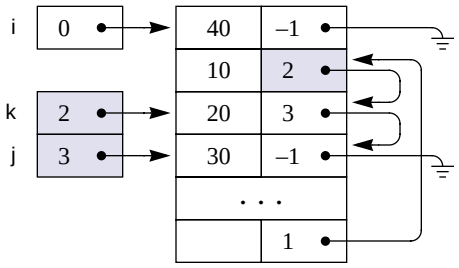
(a) After the two recursive calls.

(b) After initializing i, j, k.

Figure 20.20
A trace of the join operation in MergeSort for two short linked lists.

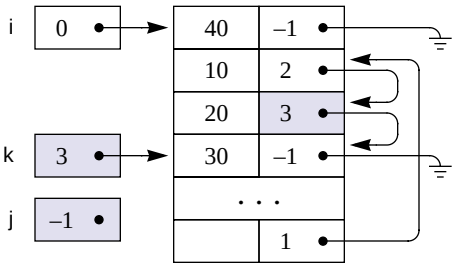


(c) After one loop execution.

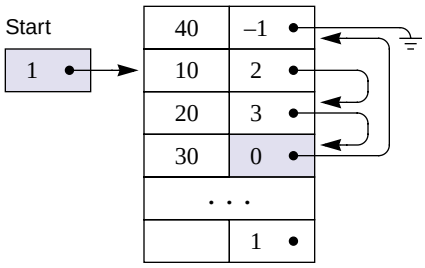


(d) After two loop executions.

Figure 20.20
A trace of the join operation in MergeSort for two short linked lists.

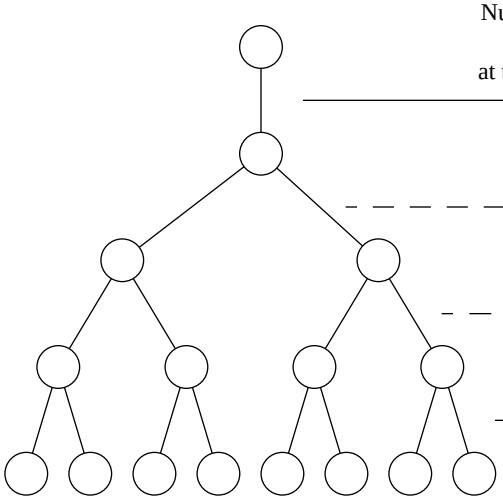


(e) After three loop executions.



(f) At procedure termination.

Figure 20.20
A trace of the join operation in MergeSort for two short linked lists.



Number of calls at this level	Number of executions of each loop	Number of comparisons
1	16	1 × 16 = 16
2	8	2 × 8 = 16
4	4	4 × 4 = 16
8	2	8 × 2 = 16
Total number of comparisons:		4 × 16 = 64

Figure 20.21
The call tree for merge sort and the best case quick sort with a 16-element list.

The five orders encountered thus far, starting with the fastest, are

- $O(\lg n)$ Example: the binary search
- $O(n)$ Example: the sequential search
- $O(n \lg n)$ Example: the merge sort and quick sort
- $O(n^2)$ Example: the single-element sorts
- $O(n^3)$ Example: matrix multiplication