```
      SUBROUTINE hpsort(n,ra)
      INTEGER n
      REAL ra(n)
         Sorts an array ra(1:n) into ascending numerical order using the Heapsort algorithm. n is
         input; ra is replaced on output by its sorted rearrangement.
      INTEGER i,ir,j,l
      REAL rra
      if (n.lt.2) return
         The index l will be decremented from its initial value down to 1 during the “hiring” (heap
         creation) phase. Once it reaches 1, the index ir will be decremented from its initial value
         down to 1 during the “retirement-and-promotion” (heap selection) phase.
      l=n/2+1
      ir=n
10    continue
        if(l.gt.1)then                Still in hiring phase.
          l=l-1
          rra=ra(l)
        else                          In retirement-and-promotion phase.
          rra=ra(ir)                  Clear a space at end of array.
          ra(ir)=ra(1)                Retire the top of the heap into it.
          ir=ir-1                     Decrease the size of the corporation.
          if(ir.eq.1)then             Done with the last promotion.
            ra(1)=rra                 The least competent worker of all!
            return
          endif
        endif
        i=l                           Whether in the hiring phase or promotion phase, we here
        j=l+l                            set up to sift down element rra to its proper level.
20      if(j.le.ir)then               “Do while j.le.ir:”
          if(j.lt.ir)then
            if(ra(j).lt.ra(j+1))j=j+1        Compare to the better underling.
          endif
          if(rra.lt.ra(j))then        Demote rra.
            ra(i)=ra(j)
            i=j
            j=j+j
          else                        This is rra’s level. Set j to terminate the sift-down.
            j=ir+1
          endif
        goto 20
        endif
        ra(i)=rra                     Put rra into its slot.
      goto 10
      END
```


CITED REFERENCES AND FURTHER READING:

Knuth, D.E. 1973, *Sorting and Searching*, vol. 3 of *The Art of Computer Programming* (Reading, MA: Addison-Wesley), §5.2.3. [1]

Sedgewick, R. 1988, *Algorithms*, 2nd ed. (Reading, MA: Addison-Wesley), Chapter 11. [2]

8.4 Indexing and Ranking

The concept of *keys* plays a prominent role in the management of data files. A data *record* in such a file may contain several items, or *fields*. For example, a record in a file of weather observations may have fields recording time, temperature, and

World Wide Web sample page from NUMERICAL RECIPES IN FORTRAN 77: THE ART OF SCIENTIFIC COMPUTING (ISBN 0-521-43064-X)
Copyright (C) 1988-1992 by Cambridge University Press. Programs Copyright (C) 1988-1992 by Numerical Recipes Software.
Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-
readable files (including this one) to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs
visit website http://www.nr.com or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

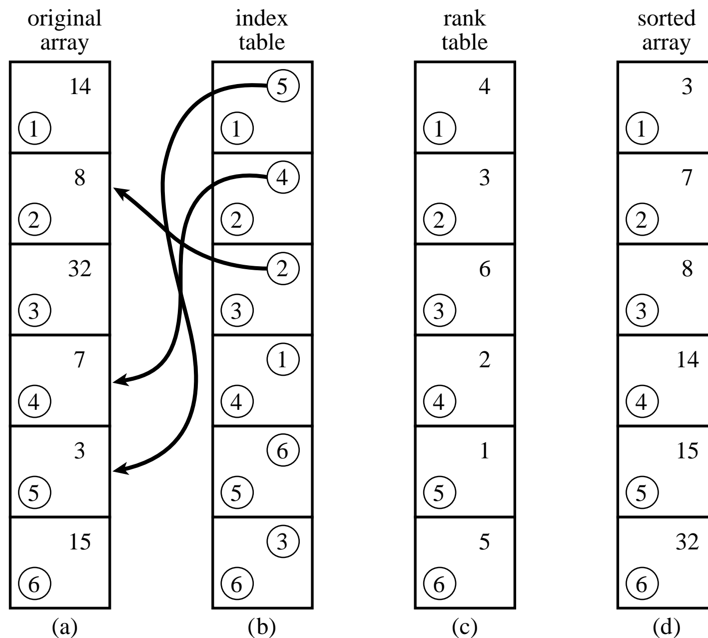

Figure 8.4.1. (a) An unsorted array of six numbers. (b) Index table, whose entries are pointers to the elements of (a) in ascending order. (c) Rank table, whose entries are the ranks of the corresponding elements of (a). (d) Sorted array of the elements in (a).

wind velocity. When we sort the records, we must decide which of these fields we want to be brought into sorted order. The other fields in a record just come along for the ride, and will not, in general, end up in any particular order. The field on which the sort is performed is called the *key* field.

For a data file with many records and many fields, the actual movement of N records into the sorted order of their keys $K_i,\ i = 1, \ldots, N$, can be a daunting task. Instead, one can construct an *index table* $I_j,\ j = 1, \ldots, N$, such that the smallest K_i has $i = I_1$, the second smallest has $i = I_2$, and so on up to the largest K_i with $i = I_N$. In other words, the array

$$K_{I_j} \quad j = 1, 2, \ldots, N \tag{8.4.1}$$

is in sorted order when indexed by j. When an index table is available, one need not move records from their original order. Further, different index tables can be made from the same set of records, indexing them to different keys.

The algorithm for constructing an index table is straightforward: Initialize the index array with the integers from 1 to N, then perform the Quicksort algorithm, moving the elements around *as if* one were sorting the keys. The integer that initially numbered the smallest key thus ends up in the number one position, and so on.

```
SUBROUTINE indexx(n,arr,indx)
INTEGER n,indx(n),M,NSTACK
REAL arr(n)
PARAMETER (M=7,NSTACK=50)
```

Indexes an array `arr(1:n)`, i.e., outputs the array `indx(1:n)` such that `arr(indx(j))` is in ascending order for $j = 1, 2, \ldots, N$. The input quantities `n` and `arr` are not changed.

World Wide Web sample page from NUMERICAL RECIPES IN FORTRAN 77: THE ART OF SCIENTIFIC COMPUTING (ISBN 0-521-43064-X)
Copyright (C) 1988-1992 by Cambridge University Press. Programs Copyright (C) 1988-1992 by Numerical Recipes Software.
Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one) to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs
visit website http://www.nr.com or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

```
      INTEGER i,indxt,ir,itemp,j,jstack,k,l,istack(NSTACK)
      REAL a
      do 11 j=1,n
          indx(j)=j
      enddo 11
      jstack=0
      l=1
      ir=n
1     if(ir-l.lt.M)then
          do 13 j=l+1,ir
              indxt=indx(j)
              a=arr(indxt)
              do 12 i=j-1,l,-1
                  if(arr(indx(i)).le.a)goto 2
                  indx(i+1)=indx(i)
              enddo 12
              i=l-1
2             indx(i+1)=indxt
          enddo 13
          if(jstack.eq.0)return
          ir=istack(jstack)
          l=istack(jstack-1)
          jstack=jstack-2
      else
          k=(l+ir)/2
          itemp=indx(k)
          indx(k)=indx(l+1)
          indx(l+1)=itemp
          if(arr(indx(l)).gt.arr(indx(ir)))then
              itemp=indx(l)
              indx(l)=indx(ir)
              indx(ir)=itemp
          endif
          if(arr(indx(l+1)).gt.arr(indx(ir)))then
              itemp=indx(l+1)
              indx(l+1)=indx(ir)
              indx(ir)=itemp
          endif
          if(arr(indx(l)).gt.arr(indx(l+1)))then
              itemp=indx(l)
              indx(l)=indx(l+1)
              indx(l+1)=itemp
          endif
          i=l+1
          j=ir
          indxt=indx(l+1)
          a=arr(indxt)
3         continue
              i=i+1
          if(arr(indx(i)).lt.a)goto 3
4         continue
              j=j-1
          if(arr(indx(j)).gt.a)goto 4
          if(j.lt.i)goto 5
          itemp=indx(i)
          indx(i)=indx(j)
          indx(j)=itemp
          goto 3
5         indx(l+1)=indx(j)
          indx(j)=indxt
          jstack=jstack+2
          if(jstack.gt.NSTACK)pause 'NSTACK too small in indexx'
          if(ir-i+1.ge.j-l)then
              istack(jstack)=ir
```

World Wide Web sample page from NUMERICAL RECIPES IN FORTRAN 77: THE ART OF SCIENTIFIC COMPUTING (ISBN 0-521-43064-X)
Copyright (C) 1988-1992 by Cambridge University Press. Programs Copyright (C) 1988-1992 by Numerical Recipes Software.
Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one) to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs
visit website http://www.nr.com or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

```
            istack(jstack-1)=i
            ir=j-1
        else
            istack(jstack)=j-1
            istack(jstack-1)=l
            l=i
        endif
    endif
    goto 1
    END
```

If you want to sort an array while making the corresponding rearrangement of several or many other arrays, you should first make an index table, then use it to rearrange each array in turn. This requires two arrays of working space: one to hold the index, and another into which an array is temporarily moved, and from which it is redeposited back on itself in the rearranged order. For 3 arrays, the procedure looks like this:

```
      SUBROUTINE sort3(n,ra,rb,rc,wksp,iwksp)
      INTEGER n,iwksp(n)
      REAL ra(n),rb(n),rc(n),wksp(n)
C     USES indexx
         Sorts an array ra(1:n) into ascending numerical order while making the corresponding
         rearrangements of the arrays rb(1:n) and rc(1:n). An index table is constructed via the
         routine indexx.
      INTEGER j
      call indexx(n,ra,iwksp)          Make the index table.
      do 11 j=1,n                      Save the array ra.
          wksp(j)=ra(j)
      enddo 11
      do 12 j=1,n                      Copy it back in the rearranged order.
          ra(j)=wksp(iwksp(j))
      enddo 12
      do 13 j=1,n                      Ditto rb.
          wksp(j)=rb(j)
      enddo 13
      do 14 j=1,n
          rb(j)=wksp(iwksp(j))
      enddo 14
      do 15 j=1,n                      Ditto rc.
          wksp(j)=rc(j)
      enddo 15
      do 16 j=1,n
          rc(j)=wksp(iwksp(j))
      enddo 16
      return
      END
```

The generalization to any other number of arrays is obviously straightforward.

A *rank table* is different from an index table. A rank table's jth entry gives the rank of the jth element of the original array of keys, ranging from 1 (if that element was the smallest) to N (if that element was the largest). One can easily construct a rank table from an index table, however:

World Wide Web sample page from NUMERICAL RECIPES IN FORTRAN 77: THE ART OF SCIENTIFIC COMPUTING (ISBN 0-521-43064-X)
Copyright (C) 1988-1992 by Cambridge University Press. Programs Copyright (C) 1988-1992 by Numerical Recipes Software.
Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one) to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs
visit website http://www.nr.com or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).

```
SUBROUTINE rank(n,indx,irank)
INTEGER n,indx(n),irank(n)
    Given indx(1:n) as output from the routine indexx, this routine returns an array irank(1:n),
    the corresponding table of ranks.
INTEGER j
do 11 j=1,n
    irank(indx(j))=j
enddo 11
return
END
```

Figure 8.4.1 summarizes the concepts discussed in this section.

8.5 Selecting the Mth Largest

Selection is sorting's austere sister. (Say *that* five times quickly!) Where sorting demands the rearrangement of an entire data array, selection politely asks for a single returned value: What is the kth smallest (or, equivalently, the $m = N+1-k$th largest) element out of N elements? The fastest methods for selection do, unfortunately, rearrange the array for their own computational purposes, typically putting all smaller elements to the left of the kth, all larger elements to the right, and scrambling the order within each subset. This side effect is at best innocuous, at worst downright inconvenient. When the array is very long, so that making a scratch copy of it is taxing on memory, or when the computational burden of the selection is a negligible part of a larger calculation, one turns to selection algorithms without side effects, which leave the original array undisturbed. Such *in place* selection is slower than the faster selection methods by a factor of about 10. We give routines of both types, below.

The most common use of selection is in the statistical characterization of a set of data. One often wants to know the median element in an array, or the top and bottom quartile elements. When N is odd, the median is the kth element, with $k = (N+1)/2$. When N is even, statistics books define the median as the arithmetic mean of the elements $k = N/2$ and $k = N/2 + 1$ (that is, $N/2$ from the bottom and $N/2$ from the top). If you accept such pedantry, you must perform two separate selections to find these elements. For $N > 100$ we usually define $k = N/2$ to be the median element, pedants be damned.

The fastest general method for selection, allowing rearrangement, is *partitioning*, exactly as was done in the Quicksort algorithm (§8.2). Selecting a "random" partition element, one marches through the array, forcing smaller elements to the left, larger elements to the right. As in Quicksort, it is important to optimize the inner loop, using "sentinels" (§8.2) to minimize the number of comparisons. For sorting, one would then proceed to further partition both subsets. For selection, we can ignore one subset and attend only to the one that contains our desired kth element. Selection by partitioning thus does not need a stack of pending operations, and its operations count scales as N rather than as $N \log N$ (see [1]). Comparison with `sort` in §8.2 should make the following routine obvious:

World Wide Web sample page from NUMERICAL RECIPES IN FORTRAN 77: THE ART OF SCIENTIFIC COMPUTING (ISBN 0-521-43064-X)
Copyright (C) 1988-1992 by Cambridge University Press. Programs Copyright (C) 1988-1992 by Numerical Recipes Software.
Permission is granted for internet users to make one paper copy for their own personal use. Further reproduction, or any copying of machine-readable files (including this one) to any server computer, is strictly prohibited. To order Numerical Recipes books, diskettes, or CDROMs
visit website http://www.nr.com or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).