```
3       continue                        Beginning of innermost loop.
            i=i+1                       Scan up to find element > a.
        if(arr(i).lt.a)goto 3
4       continue
            j=j-1                       Scan down to find element < a.
        if(arr(j).gt.a)goto 4
        if(j.lt.i)goto 5                Pointers crossed. Exit with partitioning complete.
        temp=arr(i)                     Exchange elements of both arrays.
        arr(i)=arr(j)
        arr(j)=temp
        temp=brr(i)
        brr(i)=brr(j)
        brr(j)=temp
        goto 3                          End of innermost loop.
5       arr(l+1)=arr(j)                 Insert partitioning element in both arrays.
        arr(j)=a
        brr(l+1)=brr(j)
        brr(j)=b
        jstack=jstack+2
          Push pointers to larger subarray on stack, process smaller subarray immediately.
        if(jstack.gt.NSTACK)pause 'NSTACK too small in sort2'
        if(ir-i+1.ge.j-l)then
            istack(jstack)=ir
            istack(jstack-1)=i
            ir=j-1
        else
            istack(jstack)=j-1
            istack(jstack-1)=l
            l=i
        endif
    endif
    goto 1
    END
```

You could, in principle, rearrange any number of additional arrays along with `brr`, but this becomes wasteful as the number of such arrays becomes large. The preferred technique is to make use of an index table, as described in §8.4.

CITED REFERENCES AND FURTHER READING:

Sedgewick, R. 1978, *Communications of the ACM*, vol. 21, pp. 847–857. [1]

8.3 Heapsort

While usually not quite as fast as Quicksort, Heapsort is one of our favorite sorting routines. It is a true "in-place" sort, requiring no auxiliary storage. It is an $N \log_2 N$ process, not only on average, but also for the worst-case order of input data. In fact, its worst case is only 20 percent or so worse than its average running time.

It is beyond our scope to give a complete exposition on the theory of Heapsort. We will mention the general principles, then let you refer to the references [1,2], or analyze the program yourself, if you want to understand the details.

A set of N numbers $a_i,\ i = 1, \ldots, N$, is said to form a "heap" if it satisfies the relation

$$a_{j/2} \geq a_j \quad \text{for} \quad 1 \leq j/2 < j \leq N \tag{8.3.1}$$

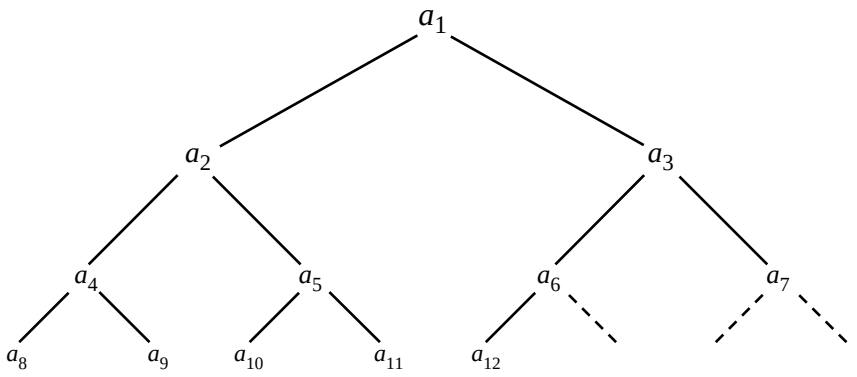

Figure 8.3.1. Ordering implied by a "heap," here of 12 elements. Elements connected by an upward path are sorted with respect to one another, but there is not necessarily any ordering among elements related only "laterally."

Here the division in $j/2$ means "integer divide," i.e., is an exact integer or else is rounded down to the closest integer. Definition (8.3.1) will make sense if you think of the numbers a_i as being arranged in a binary tree, with the top, "boss," node being a_1, the two "underling" nodes being a_2 and a_3, *their* four underling nodes being a_4 through a_7, etc. (See Figure 8.3.1.) In this form, a heap has every "supervisor" greater than or equal to its two "supervisees," down through the levels of the hierarchy.

If you have managed to rearrange your array into an order that forms a heap, then sorting it is very easy: You pull off the "top of the heap," which will be the largest element yet unsorted. Then you "promote" to the top of the heap its largest underling. Then you promote *its* largest underling, and so on. The process is like what happens (or is supposed to happen) in a large corporation when the chairman of the board retires. You then repeat the whole process by retiring the new chairman of the board. Evidently the whole thing is an $N \log_2 N$ process, since each retiring chairman leads to $\log_2 N$ promotions of underlings.

Well, how do you arrange the array into a heap in the first place? The answer is again a "sift-up" process like corporate promotion. Imagine that the corporation starts out with $N/2$ employees on the production line, but with no supervisors. Now a supervisor is hired to supervise two workers. If he is less capable than one of his workers, that one is promoted in his place, and he joins the production line. After supervisors are hired, then supervisors of supervisors are hired, and so on up the corporate ladder. Each employee is brought in at the top of the tree, but then immediately sifted down, with more capable workers promoted until their proper corporate level has been reached.

In the Heapsort implementation, the same "sift-up" code can be used for the initial creation of the heap and for the subsequent retirement-and-promotion phase. One execution of the Heapsort subroutine represents the entire life-cycle of a giant corporation: $N/2$ workers are hired; $N/2$ potential supervisors are hired; there is a sifting up in the ranks, a sort of super Peter Principle: in due course, each of the original employees gets promoted to chairman of the board.

```
      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