Later we will show that quicksort runs in **$\Theta(n^2)$ time** in the worst case, but that it runs in **$\Theta(n\lg(n))$ time** in the expected case, assuming the input is randomly ordered.

At the moment, let us consider the extra space (in addition to the array being sorted) used by quicksort.

- *partition*() uses only constant extra space.
- At first, it might appear that *quicksort*() also uses only constant extra space.
  - But this can never be the case for a recursive function (unless the maximum depth of recursion is bounded by a constant).
  - Each time a function is called (recursively or otherwise), we get a new activation of the function. The activation continues to exist until the function returns to its caller.
  - Each activation has its own *activation record*, or *stack frame*, on the run-time stack.
    - The activation record may contain space for automatic local variables and parameters, a return address, and an area to save the current status.
  - For a recursive function, the memory requirements are determined by the maximum number of activations existing at any one time, that is, by the maximum depth of recursion.
- For *quicksort*() as we have implemented it, the maximum depth of recursion will be n in the worst case.
  - This would occur if every call to *partition*() produced as uneven a split as possible (sublist sizes $n-1$ and 0).

  - It would make the extra space requirements $\Theta(n)$ — possibly with a fairly large constant multiplying n.

- With a little effort, we can limit the maximum depth of recursion to approximately $\lg(n)$, even in the worst case.
  - Then the extra space required will be $\Theta(\lg(n))$ — insignificant compared to the $\Theta(n)$ space for the array being sorted.
  - Note, however, the worst case running time remains $\Theta(n^2)$.

quicksort2(A, left, right) performs the same function as *quicksort*(*A*, *left*, *right*), but the depth of recursion is limited to $\lg(n)$, and the extra space is only $\Theta(\lg(n))$ even in the worst case.

```
void quicksort2( T[] A, Integer left, Integer right)
    while ( left < right )
        q = partition( A, left, right);
        if ( q – left < right – q )
            quicksort2( A, left, q–1);    // Recursive call sorts smaller sublist
            left = q + 1;                 // Loop back to sort larger sublist
        else
            quicksort2( A, q+1, right);   // Recursive call sorts smaller sublist
            right = q – 1;                // Loop back to sort larger sublist
```

Each time *quicksort2*() calls itself recursively, it does so only for the smaller of the two sublists. This means that each time the depth of recursion rises by one, the size of the input to quicksort2() is cut at least in half. Thus the depth of recursion cannot exceed $\lg(n)$.

Note the larger sublist is sorted by repeating the while-loop with the value of *left* or *right* adjusted to correspond to the larger sublist, rather than the original list. This requires no additional space.