```
// Math 188, Winter 2001, Prof. Tesler

// Ordinary quicksort
// Input:  integer n
//         array S[0 .. n-1] of integers
// Output: array is sorted in increasing order
void quicksort (int n, int S[]) {
  quicksort(S,0,n-1);
}
void quicksort (int S[], int low, int high) {
  int pivotpoint;

  if (high > low) {
    q_partition(S,low,high,pivotpoint);
    quicksort(S,low,pivotpoint-1);
    quicksort(S,pivotpoint+1,high);
  }
}

// Input: array of integers S[low..high]
// Output: array is partitioned into S[low..(p-1)]; S[p]; S[(p+1)..high]
//                                    #s < S[p]             #s >= S[p]
//         The pivot location p is returned in pivotpoint
void q_partition(int S[], int low, int high, int& pivotpoint) {
  int i,j;

  int pivotitem = S[low];               // 1st item is pivot item

  j=low;                                // S[(low+1)..j] are entries < pivot
  for(i=low+1; i<=high; i++) {
    if (S[i]<pivotitem) {               // move down entries < pivot
      j++;
      swap(S,j,i);
    }
  }

  pivotpoint = j;                       // put pivot item at its final
  swap(S,low,pivotpoint);             //   location
}

void swap(int S[], int i, int j) {
  if (i==j) return;

  int temp = S[i];
  S[i] = S[j];
  S[j] = temp;
}

/***************************************************************************/

// Quicksort with tail recursion
// The recursive calls achieve a stack depth between 1 and 1+lg(n) frames
// Ordinary quicksort achieves a depth between 1+lg(n) and n frames
void quicksort_t (int n, int S[]) {
  quicksort_t(S,0,n-1);
}
void quicksort_t(int S[], int low, int high) {
  int pivotpoint;

  while (high > low) {
    q_partition(S,low,high,pivotpoint);

    if (high-pivotpoint > pivotpoint-low) { // right half is larger, so
      quicksort_t(S,low,pivotpoint-1);      //   recurse on left half
      low = pivotpoint+1;                   //   and tail recurse on right half
    } else {                                // left half is larger, so
      quicksort_t(S,pivotpoint+1,high);     //   recurse on right half
      high = pivotpoint-1;                  //   and tail recurse on left half
    }
  }
}
```