1. 1. [11, 5, 24, 13, 6, 2, 9, 14, 4, 7, 8]  
       pivot = 7  
       i = -1
      1. = j, 11 > 7; no swap  
         [11, 5, 24, 13, 6, 2, 9, 14, 4, 7, 8]
      2. = j, 5 < 7; i = 0, swap xs[0] xs[1]  
         [ 5, 11, 24, 13, 6, 2, 9, 14, 4, 7, 8]
      3. = j, 24 > 7; no swap  
         [ 5, 11, 24, 13, 6, 2, 9, 14, 4, 7, 8]
      4. = j, 13 > 7; no swap  
         [ 5, 11, 24, 13, 6, 2, 9, 14, 4, 7, 8]
      5. = j, 6 < 7; i = 1, swap xs[1] xs[4]  
         [ 5, 6, 24, 13, 11, 2, 9, 14, 4, 7, 8]
      6. = j, 2 < 7; i = 2, swap xs[2] xs[5]  
         [ 5, 6, 2, 13, 11, 24, 9, 14, 4, 7, 8]
      7. = j, 9 > 7; no swap   
         [ 5, 6, 2, 13, 11, 24, 9, 14, 4, 7, 8]
      8. = j, 14 > 7; no swap  
         [ 5, 6, 2, 13, 11, 24, 9, 14, 4, 7, 8]
      9. = j, 4 < 7; i = 3, swap xs[3] xs[8]  
         [ 5, 6, 2, 4, 11, 24, 9, 14, 13, 7, 8]
      10. = j, i = 4, swap xs[4] xs[9]  
          [ 5, 6, 2, 4, 7, 24, 9, 14, 13, 11, 8]
   2. rval = 4
2. 1. Worst case is when array is already sorted so algo must traverse the entire subarray on each iteration
   2. Randomizing the pivot reduces the chance of the first/last element being picked as pivot which brings the time complexity closer to nlogn.
   3. QuickSort(xs, l, r):  
       if (l < r)  
       pivot = PickPivot(xs[l..r])  
       index = Partition(xs, l, r, pivot)  
       QuickSort(xs, l, index - 1)  
       QuickSort(xs, l, index + 1)
   4. LowerMed(xs, l, r):  
       if l == r  
       return xs[l]  
       pivot = PickPivot(xs[l..r])  
       index = Partition(xs, l, r, pivot)  
       k = index – l + 1  
       if k == floor((r – l + 1) / 2)  
       return xs[index]  
       else if k > floor ((r – l + 1) / 2)  
       return LowerMed(xs, l, index - 1)  
       else  
       return LowerMed(xs, l, index + 1)
   5. Analysis does not change since analysis is based on if n is large, and n is large for asymptotic PickPivot
3. kval(xs, l, r, k):   
    divide array into groups of 5 elements O(n)  
    array medians = HappyMedian(each group) O(n)  
    median = kval(medians, 0, len - 1, len / 2) O(n/5)  
    pos = Partition(xs, l, r, median) O(n)  
    if pos == k  
    return xs[pos] O(1)  
    if pos > k  
    return kval(xs, l, pos - 1, k) O(n)  
    return kval(xs, pos + 1, r, k – pos + l - 1) O(n)  
   T(n) = O(n)