1. Problem 7-5.

   (a) Let x_1, x_2, and x_3 be the three values that are picked at random. Since there are $3! = 6$ ways to arrange 3 elements, we can calculate the probability assuming $x_1 < x_2 < x_3$ and multiply by 6: $p_i = 6\Pr[x_1 < i, x_2 = i, x_3 > i] = 6 \cdot \frac{2}{n} \cdot \frac{1}{n-1} \cdot \frac{n-i}{n-2} = \frac{6(i-1)(n-i)}{n(n-1)(n-2)}$.

   (b) I'll assume n is even, since it doesn't affect the ratio in the limit. $p_{n/2} = \frac{6(\frac{n}{2}-1)(\frac{n}{2})}{n(n-1)(n-2)} = \frac{3(n^2-2n)}{2(n^3-3n^2+2n)}$. For the ordinary implementation, $p = \frac{1}{n}$. So $\lim_{n\to\infty} \frac{p_{n/2}}{p}$ $= \lim \frac{3n(n^2-2n)}{2(n^3-3n^2+2n)} = \lim \frac{3(1-2/n)}{2(1-3/n+2/n^2)} = \frac{3}{2}$.

   (c) For the new method, the probability of a good split is $\Pr[\frac{n}{3} \le i \le \frac{2n}{3}] = \sum_{i=n/3}^{2n/3} \frac{6(i-1)(n-i)}{n(n-1)(n-2)}$. To approximate by an integral, I'll use the variable t to mean the fraction of the way from 1 to n. Then n, $n-1$, $n-2$ are all nearly 1 (all of the way through). So the approximating integral is $\int_{1/3}^{2/3} \frac{6t(1-t)}{1\cdot1\cdot1} dt = \int_{1/3}^{2/3} 6(t-t^2)dt = 6(\frac{1}{2}t^2 - \frac{1}{3}t^3)|_{1/3}^{2/3} = \frac{13}{27}$. For the old method, the probability of a good split is just $\Pr[\frac{n}{3} \le i \le \frac{2n}{3}] = \sum_{i=n/3}^{2n/3} \frac{1}{n} = \frac{1}{n} \cdot \frac{n}{3} = \frac{1}{3} < \frac{13}{27}$.

   (d) The best possible running time for any version of quicksort would be achieved when the median is picked for the pivot every time, but this is still only $\Omega(n \lg n)$ since the array is divided in half every time and the recursion handles both halves. Thus any scheme for picking the pivot, such as the median-of-three considered in this problem, cannot affect the asymptotic running time, but only the constant factor in the running time.

2. Parts a–d of Problem 8-5.

   (a) 1-sorted is equivalent to completely sorted, in the standard use of the term.

   (b) Here are some possible examples: $\{2,1,3,4,5,6,7,8,9,10\}$ $\{2,1,4,3,6,5,8,7,10,9\}$ $\{6,1,7,2,8,3,9,4,10,5\}$.

   (c) Note simply that $A_i \le A_{i+k} \Leftrightarrow \sum_{j=i}^{i+k-1} A_j - \sum_{j=i+1}^{i+k} A_j = A_i + A_{i+1} + \ldots + A_{i+k-2} + A_{i+k-1} - A_{i+1} - A_{i+2} - \ldots - A_{i+k-1} - A_{i+k} = A_i - A_{i+k} \le 0 \Leftrightarrow \frac{1}{k}\sum_{j=i}^{i+k-1} A_j \le \frac{1}{k}\sum_{j=i+1}^{i+k} A_j$.

   (d) Split the array (arbitrarily) into k subarrays of size $\lfloor \frac{n}{k} \rfloor$ or $\lfloor \frac{n}{k} \rfloor + 1$ each. Sort each subarray at a cost of $O(\frac{n}{k} \lg \frac{n}{k})$ each to form sorted subarrays $A_1, \ldots, A_k$. So far this is a total of $O(k(\frac{n}{k} \lg \frac{n}{k})) = O(n \lg \frac{n}{k})$. Finally, interleave the the k sorted subarrays to form one k-sorted array A: $A[i] = A_{i \bmod k}[i \operatorname{div} k]$. This involves $O(n)$ operations but no comparisons. Thus the total time is $O(n \lg \frac{n}{k})$. This works because for $i \in \{1, \ldots, n-k\}$, $A[i] = A_{i \bmod k}[i \operatorname{div} k] < A_{i \bmod k}[(i \operatorname{div} k) + 1] =$

$A_{(i+k) \bmod k}[(i+k) \operatorname{div} k] = A[i+k]$ since $A_{i \bmod k}$ is a sorted array, and therefore A is k-sorted, by part (c) above.

3. Problem 9.1-1.
First find the minimum in the list in rounds in this way: Split the array into pairs and compare each pair. The smaller in each comparison advances to the next round, which has half as many elements to compare. At the end only the global minimum will be left. This is $\frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \ldots + 1 = \frac{n}{2}(2)(1 - \frac{1}{n}) = n - 1$ comparisons. Now, note that the second smallest element could only have been eliminated by the global minimum. Therefore, it must be one of the $\lceil \lg n \rceil$ elements that the global min was compared to. So simply find the min among these elements and return it as the second smallest in the whole array. The total is $n - 1 + \lceil \lg n \rceil - 1$ comparisons.

4. Problem 9.3-1.
If the input elements are divided into groups of 7, the analysis will mostly be the same as when they are divided into groups of 5. The number of elements greater than x will now be at least $4\left(\left\lceil \frac{1}{2}\left\lceil \frac{n}{7} \right\rceil \right\rceil - 2\right)$, and so SELECT is called recursively on at most $5n/7+8$ elements in step 5. Thus we have the recurrence $T(n) \leq T(\lceil n/7 \rceil)+T(5n/7+8)+O(n)$, and the substitution method gives us $T(n) \leq 6cn/7 + 9c + an \leq cn$ for an appropriate value of c and for large enough n. Thus groups of 7 will work.
However, this same analysis does not go through for groups of 3. In fact, one can show that the running time for SELECT using groups of 3 is $\Omega(n \lg n)$ and therefore cannot be $O(n)$: The number of elements greater than x is at most $2\left(\left\lfloor \frac{1}{2}\left\lfloor \frac{n}{3} \right\rfloor \right\rfloor\right) \leq \frac{n}{3}$, so SELECT is called recursively on at least $\frac{2n}{3}$ elements in step 5. We can use the substituion method to prove $T(n) = \Omega(n \lg n)$ like this: $T(n) \geq T(\lceil \frac{n}{3} \rceil) + T(\frac{2n}{3}) + \Omega(n) \geq T(\frac{n}{3}) + T(\frac{2n}{3}) + \Omega(n) \geq c(\frac{n}{3}) \lg(\frac{n}{3}) + c(\frac{2n}{3}) \lg(\frac{2n}{3}) + an = cn \lg n - cn \lg 3 + \frac{2}{3}cn + an \geq cn \lg n$ if $c \leq \frac{a}{\lg 3 - 2/3}$. This completes the argument.

5. Problem 9.3-7.
Use the $O(n)$-time SELECT algorithm to find the median m of S. I assume their measure of "closest" means that the distance of a number x to m is $|m - x|$. We create a new array D, the same size as S, that stores the distance to m for each element in S. Now we use the $O(n)$-time SELECT algorithm again but to find the kth smallest element in D. Say this occurs at index j. Now run through D and keep track of all the indices I_1 that have value less than $D[j]$ and all the indices I_2 that have value equal to $D[j]$. Finally, output all the elements of S that correspond to I_1 and enough elements that correspond to I_2 to total k output values.

   The other way to interpret this problem is that they wanted the k elements of S centered at the median, in other words, $k/2$ elements on each side of m. We can solve this problem as follows: Use the $O(n)$-time SELECT algorithm to find order statistics

$n/2 - k/2$ and $n/2 + k/2$. Then run through the list S and output all the values that are between the two found values.

6. Problem 9.3-8.
To get an $O(\lg n)$-time algorithm for this search problem, we will need to eliminate half the input at each step. Note that finding the median in a sorted array A of length n is $O(1)$, since the median is just $A[n/2]$. So let $x_1 = X[n/2]$ and $y_1 = Y[n/2]$, and let m be the median value we're looking for. There are four possible locations for m: $X[1, \ldots, \frac{n}{2}]$, $X[\frac{n}{2}, \ldots, n]$, $Y[1, \ldots, \frac{n}{2}]$, and $Y[\frac{n}{2}, \ldots, n]$. There are $2n/2 = n$ values less than m in X and Y, but only (say) k of them in X and $n - k$ of them in Y. Suppose first that $x_1 < y_1$. If $m < x_1$, then $k < \frac{n}{2}$. But that would mean the median is greater than $Y[1], \ldots, Y[n - k]$, including $Y[\frac{n}{2}] = y_1 > x_1$, which contradicts that $m < x_1$. Therefore we can eliminate $X[1, \ldots, \frac{n}{2}]$ for the median search. Likewise, $m > y_1$ leads to a contradiction, eliminating $Y[\frac{n}{2}, \ldots, n]$. Thus we have eliminated half the input, and we recurse on the two subarrays $X[\frac{n}{2}, \ldots, n]$ and $Y[1, \ldots, \frac{n}{2}]$. If $x_1 > y_1$, we recurse on $X[1, \ldots, \frac{n}{2}]$ and $Y[\frac{n}{2}, \ldots, n]$ instead. The recursion stops when $n = 1$, and by convention (see CLRS p. 190) we return $\min\{X[1], Y[1]\}$.