Question 1

(a)
```
Input: Array 'A', beginning of sublist 'p', end of sublist 'r'
Output: Left boundary 'lb', Right boundary 'rb'
PARTITION (A, p, r)
    x <- A[r]
    rb <- p-1
    for j <- p to r-1
        do if A[j] <= x
            then rb <- rb+1
            exchange A[rb] <-> A[j]
    exchange A[rb+1] <-> A[r]
    lb <- rb
    rb <- rb+1
    for k <- p to lb
        do if A[k] == x
            then lb <- lb-1
            exchange A[lb] <-> A[k]
    return lb, rb
end function
```

(b) In a normal quicksort, a maximum (worst case) of n levels would be required in the recursion tree, for an array of n distinct elements. It is given that the array has n elements, and k distinct elements. At each level of the recursion tree, one distinct element is chosen as the pivot, and all corresponding elements with that value are removed from the array, i.e, won't be considered in the subsequent levels of recursive calls. Thus when there are only k distinct elements in the array, in the worst case, a maximum of k levels in the recursive tree will be required to remove k distinct pivot elements.

(c) The length of the longest simple path from the root of a decision tree to any of its reachable leaves represents the worst-case number of comparisons that the corresponding sorting algorithm performs. For sorting an array of n elements of which k are distinct values, the worst-case would have a recursion tree of depth k, wherein the array is either sorted in ascending or descending order. Thus, the lower bound for the worst case would be proportional to k.n, i.e, c.k.n. Therefore, worst case running time $\geq ckn$.

Question 2

The given array A has n elements, and k distinct elements. Let us assume that each of the k distinct elements occurs p_1, p_2,.., p_k times respectively. Then:
$E(X) = E(\sum_{i<j} X_{i,j})$

$\implies E(X) = \sum_{i<j} E(X_{i,j})$

$\implies E(X) = \sum_{i<j} P(X_{i,j} = 1)$

$X_{i,j} = 1$ if x_i, x_j are in the same sublist, and, either of them is chosen as pivot. In the given array, x_i repeats p_i times, and x_j repeats p_j times. Thus,

$P(X_{i,j} = 1) = \frac{p_i+p_j}{|L|}$, where, $|L|$ is the size of sublist

$|L| \geq (j - i + 1)$

$\implies E(X) = \sum_{i<j} P(X_{i,j} = 1)$

$\implies E(X) \leq \sum_{i<j} \frac{p_i+p_j}{j-i+1}$

$\implies E(X) \leq \sum_{i=1}^{k-p_a} \sum_{j=p_b}^{k} \frac{p_i+p_j}{j-i+1}$

$\implies E(X) \leq \sum_{i=1}^{k-p_a} \sum_{l=p_b}^{k} \frac{p_i+p_{l+i}}{l+1}$

$\implies E(X) \leq \sum_{l=1}^{k} \frac{2(p_1+p_2+p_3+...+p_k)}{l+1}$

$\implies E(X) \leq \sum_{l=1}^{k} \frac{2n}{l+1}$

$\implies E(X) \leq 2n. \sum_{l=1}^{k} \frac{1}{l+1}$

$\implies E(X) \leq 2n. \sum_{l=1}^{k} \frac{1}{l}$

$\implies E(X) \leq c.n.log_2(k)$

$\implies E(X) = O(n.log_2(k))$

Therefore, (c).

Question 3

When using a decision tree method for sorting using a comparison-based model, the length of the longest simple path from the root of a decision tree to any of its reachable leaves represents the worst-case number of comparisons that the corresponding sorting algorithm performs. Thus, to find the worst-case number of comparisons, it suffices to determine the height of the decision tree in which each permutation appears as a reachable leaf. Thus, the worst case number of comparisons in a nXn matrix, which is either row-sorted or column-sorted would be $\geq (log_2(n)+n)$, i.e, $\geq log_2(n)$. Therefore, when the matrix is either row-sorted or column-sorted, the worst case number of comparisons is given by $\Omega(n) = log_2(n)$.

When the nXn matrix is unsorted, the problem can be broken down into finding the element from n unsorted lists. The decision tree in this case has $n^2 + 1$ distinct leaves, and the height of the tree would be $log_2(n^2)$. Thus, the worst case number of comparisons, is $\geq log_2(n^2)$. Therefore, $p \geq 2.log_2(n)$, i.e, $p \geq c.log_2(n) \implies p = \Omega(log_2(n))$
Thus, irrespective of sorting or non-sorting of a nxn matrix (and in whichever order), the lower bound for the worst number of comparisons is $\Omega(log_2(n))$

Question 4

(a)
```
Input: Array 'A', length 'n'
Output: Sorted bit-array
BIT-SORT (A, p, r)
    i <- p - 1
    for j <- p to r
        do if A[j] < MAX(A, n)
            i += 1
            exchange A[i] <-> A[j]
    return A
end function

Input: Array 'A'
Output: Maximum element of array
MAX (A, n)
    max <- A[1]
    for i <- 2 to n
        do if A[i] > max
            max <- A[i]
    return max
end function
```

(b) Let $< a_1a_2a_3...a_n >$ denote the n-bit binary string. Each internal node denotes $(a_i : a_j)$ with two outcomes, i.e, $\leq$ and $>$ for the left edge and right edge respectively. Each path corresponds to one possible outcome of the algorithm. When using a decision tree method for sorting using a comparison-based model, the length of the longest simple path from the root of a decision tree to any of its reachable leaves represents the worst-case number of comparisons that the corresponding sorting algorithm performs. The height of the decision tree for the worst-case when sorting a n-bit binary string would be $\geq n$. Thus, $\Omega(n) = n$, i.e, the lower bound for the problem is n.

Question 5

Of the three groups formed, the number of elements greater than x, and the number of elements less than x is:
$x \geq 2(\frac{1}{2}[\frac{n}{3}] - 2)$

$\implies x \geq \frac{n}{3} - 4$
Thus, in the worst-case scenario, SELECT recursively calls on at most $(\frac{2n}{3} + 4)$ elements.

$$\text{T(n)} = \begin{cases} T(1) & if n \leq n_0 \\ T(\frac{n}{3}) + T(\frac{2n}{3} + 4) + O(n) & if n > n_0 \end{cases}$$

$T(n) \leq T(\frac{n}{3}) + T(\frac{2n}{3} + 4) + O(n)$

$\implies T(n) \leq T(\frac{n}{3}) + T(\frac{2cn}{3}) + O(n)$

$\implies T(n) = \Omega(nlog_2(n))$

Thus, it doesn't have a linear solution. Therefore, when we make groups of 3 elements, it doesn't lead to linear time complexity.