2.4. (1) $T(n) = 5T(n/2) + O(n)$. Using master theorem, $T(n) = O(n^{\log_2 5})$.

(2) $T(n) = 2T(n-1) + O(1)$. Expanding it, we have $T(n) = 4T(n-2) + 2O(1) + O(1) = 2^n T(0) + O(2^{n-1} + ... + 1)$. Assuming that $T(0) = 0$, $T(n) = O(2^n)$.

(3) $T(n) = 9T(n/3) + O(n^2)$. Using master theorem, $T(n) = O(n^2 \log n)$.

2.14 Sort the array in $O(n \log n)$ time and then scan the array in $O(n)$ to find duplicated elements. The total complexity is $O(n \log n)$.

2.16 There is a $O(\log n)$ method to find the position $n$ of the last non-infinity element in $A$: we increase the guessed location by 2 times each time until we first hit an infinity element. The takes $O(\log n)$ time. We then do a binary search in the interval of the current guess and the previous guess. Note that the interval’s length is less than $n$. So, the complexity of the binary search that locates an element who’s right neighbor is infinity is $O(\log n)$. After the boundary of $A$ is determined, searching for arbitrary element is $O(\log n)$. Therefore, the overall complexity of searching is $O(\log n)$.

2.17 We use a similar method to binary search. The idea is that we recursively check each middle element in an array: if $A[i] = i$, we find the element; if $A[i] > i$, then we can discard everything on the right of $A[i]$, and similarly if $A[i] < i$, we can discard everything on the left of $A[i]$. (Use the fact that $A$ contains distinct integers).

2.19 (a) The complexity of sequentially merging is $O(2n + 3n + 4n + ... + kn) = O(k^2 n)$.

(b) Using a divide and conquer method, we partition the arrays into left half group and right half group and recursively merge them. We thus have $T(k) = 2T(k/2) + O(kn)$. The complexity is thus $O(kn \log k)$.

2.20 We first scan the elements in $x[1..n]$ and put them into bins $1, 2, .., M + 1$ that correspond to numbers from $\min(x)$ to $\max(x)$. Then we scan the bins can sequentially put the numbers in the sorted order. The whole procedure takes time $O(n + M)$.

2.23 (a) We split the array $A$ into two halves. If $A$ has a majority element, at least one of the halves has a majority element. This can be proved
by contradiction: if none of the two halves have a majority element, the original array must not have a majority element. However, that the halves have majority element does not guarantee $A$ has one. If we find a majority element in one half, we need to further find out the number of this element in the other half and test whether the total of this number is the majority in $A$. The checking procedure is a linear method. Thus $T(n) = 2T(n/2) + O(n)$ and therefore $T(n) = O(n \log n)$. 