4.1)

The choice of k can significantly impact the asymptotic time complexity of this variant of merge sort. Let's analyze the impact of different values of k on the best-case, worst-case, and average-case asymptotic time complexities:

1. Best-case: The best-case scenario for merge sort is when the input sequence is already sorted, and this variant will perform similarly to the regular merge sort algorithm since the subsequences will be larger than k and the insertion sort will not be used. Thus, the best-case time complexity of this variant of merge sort is O(n log n) regardless of the value of k.
2. Worst-case: The worst-case scenario for merge sort is when the input sequence is reverse sorted. In this case, the algorithm will repeatedly divide the sequence into smaller subsequences until the size of each subsequence is equal to k, at which point the insertion sort will be applied. The worst-case time complexity of the insertion sort algorithm is O(k^2). So if k>n, the overall time complexity will be O(n^2)
3. Average-case: The average-case time complexity of this variant of merge sort depends on the distribution of the input data. If the input data is uniformly distributed, then the performance of this variant will be similar to the regular merge sort algorithm. However, if the input data has some special characteristics, such as having many small or nearly sorted subsequences, then the performance of this variant can be better or worse than the regular merge sort algorithm depending on the value of k. ( Dependency described below )

In summary, for small values of k, this variant of merge sort can perform better than the regular merge sort algorithm in some scenarios, but for larger values of k, the worst-case time complexity can be worse than the regular merge sort algorithm. Therefore, the choice of k should be based on the characteristics of the input data and the desired trade-off between average-case and worst-case performance.

For instance, for k = 3, algorithm performs in the best way, because it takes less time to sort sequence of length 3 using insertion sort algorithm instead of using merge sort. (According to their time complexities)

4.2)

(a) (2 points) T(n) = 36T(n/6) + 2n,

A = 36

B = 6

Nlogb(a) = N^2

f(n) = 2n

f(n) = O(n2-1)=O(n)

T(n) = Θ(N^2)

(b) (2 points) T(n) = 5T(n/3) + 17n^1.2

A = 5

B = 3

N^logb(a) = N^log3(5) = N^1.465

F(n) = 17n^1.2

F(n) = O(n^1.465-0.265) = O(n^1.2)

T(n) = Θ(N^1.465)

(c) (2 points) T(n) = 12T(n/2) + n^2 lg(n)

a = 12, b=2, f(n) = n^2 lg(n)  
n^logb(a) = n^3.58494   
T(n) = Θ(n^2 lg(n))

(d) (2 points) T(n) = 3T(n/5) + T(n/2) + 2^n

Recursion tree method:  
Total: n^2 ( of recursion ), but 2^n is bigger, so  
  
T(n) = Θ(2^n)

(e) (2 points) T(n) = T(2n/5) + T(3n/5) + Θ(n).

We have log5/2(n) levels in the recursion tree, each contributes Θ(n) to the total cost, so overall  
T(n) = Θ(n \* log(n))