a. We know that insertion sort on an array of size k takes $\Theta(k^2)$ time, because it takes $c_1k^2 + c_2k + c3$ steps to sort the array. The number of steps required to sort n/k arrays would simply be $c_1nk + c_2n + c3(n/k)$, which means it takes $\Theta(nk)$ time to sort the sub-arrays.

b. We can define a function MODIFIED-MERGE, that takes as arguments: A - the original array being referenced, n - the total number of elements in the array, k - the length of each sub-array, and p - the pointer at which the array starts. It also uses the MERGE function as it is normally defined.

```
MODIFIED-MERGE($A, n, k, p$)
1. $numberOfSubarrays = n/k$
2. **if** $numberOfSubarrays = 1$
3.     **return**
4. $leftCount = \lfloor n/2k \rfloor$
5. MODIFIED-MERGE($A, k \times leftCount, k, p$)
6. MODIFIED-MERGE($A, k \times (n/k - leftCount), k, p + k \times leftCount$)
7. MERGE($A, p, p + k \times leftCount - 1,\ p + n - 1$)
```

The worst running time of this algorithm can be formulated like this:

$$T(n,k) = \begin{cases} c_1 & n = k \\ 2T(n/2,k) + n & n = mk, m \in \mathbb{Z} \end{cases}$$

It is not difficult to see that this recursion equation gives us the required running time.

c. With some manipulation, the term can be rewritten as:

$$\Theta(n(k - \lg k) + n \lg n)$$

Thus, the largest value of k is given by

$$k - \lg k = \lg n$$

$$\frac{2^k}{k} = n$$

The solution to this is the function

$$f(n) = -\frac{W(\frac{-\ln 2}{n})}{\ln 2}$$

where W is the Lambert W Function (basically, $W(y)$ gives the solution to the equation $xe^x = y$)

d. In practical scenarios, it might be better to just test and see what value of k tends to fit better because of depends on how fast the machine is able to actually run different sorts of instructions, or what sorts of optimisations the compiler may do for your code. However, I am not sure of this answer. If you have a better answer, let me know!