Divide and Conquer[1]

1. We discuss the basic theme of DC, and the template.
2. Studied Order Statistics-1, which includes finding min, max-min, max-second max.
3. for Min
4. fmin(A, begin, end){

   If (begin==end) return A[begin];

   Else mid= ((begin+end-1)/2));

   Return min {fmin(begin,mid), fmin(mid+1, end)}; }
5. Recurrence relation for min: $T(n) = 2T(n/2) + 1, T(2) = 1$
6. Similarly, we can design the DnC algorithm for the other two problems.
7. For max-min: $T(n) = 2T(n/2) + 2, T(2) = 1$
8. For max-secondmax: $T(n) = 2T(n/2) + 2, T(2) = 1$
9. For max-min, max-secondmax: check the logic behind "the constant 2" in $T(n) = 2T(n/2) + 2$
10. Binary search: $T(n) = T(n/2) + constant$
11. Merge sort: $T(n) = 2T(n/2) + n, T(2) = 1$
12. solved some problem with respect to merging.

    - Consider the following modification to merge sort algorithm: divide the input arrays into thirds(rather than halves), recursively sort each third and finally combine the results using three way Merge subroutine. What is the running time taken by this successive merging algorithm, as a function of n , ignoring constant factors and lower order terms?
    - Suppose that you are given k sorted arrays, each with n elements, and you want to combine them into a single array of kn elements. One approach is to use Merge subroutine like wise, first merging the first two arrays, then merging the result with the third array, then with the fourth array, and so on until you merge in the k-th array. What is the running time taken by this successive merging algorithm, as a function of n and k, ignoring constant factors and lower order terms?

[1]Prepared by Pawan K. Mishra