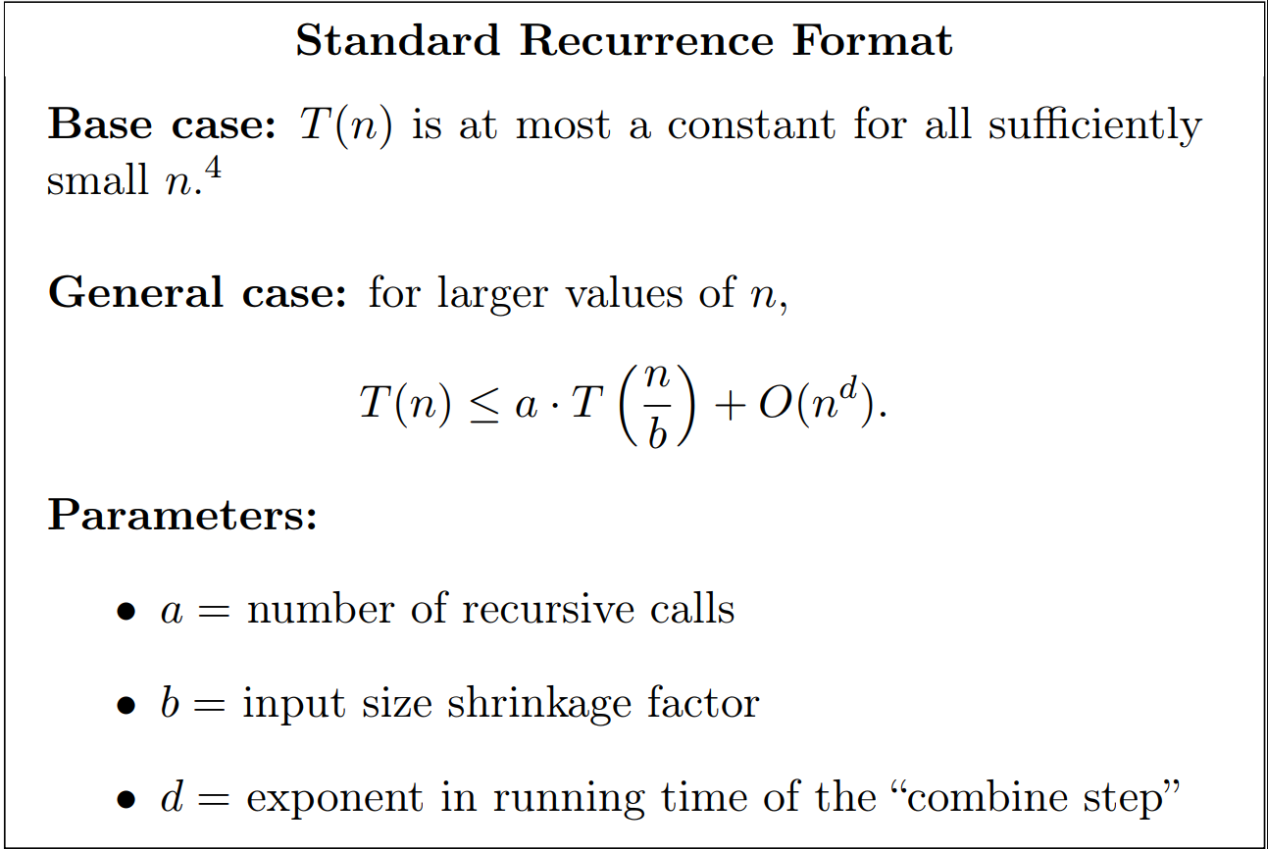
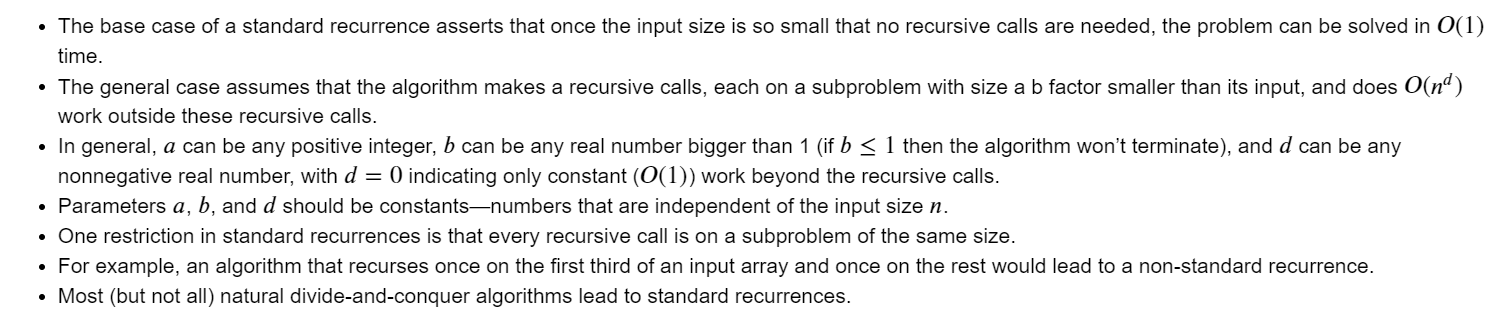
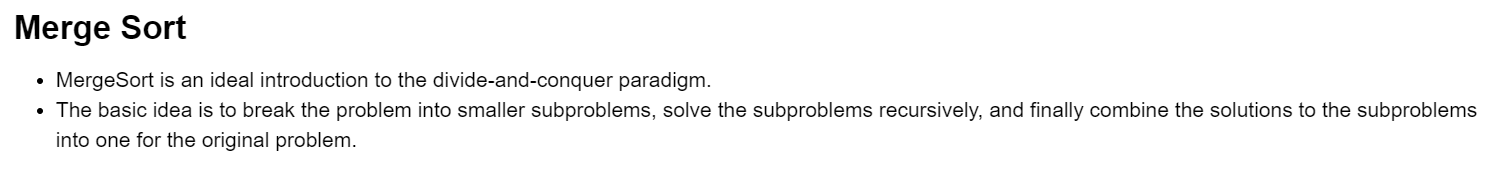
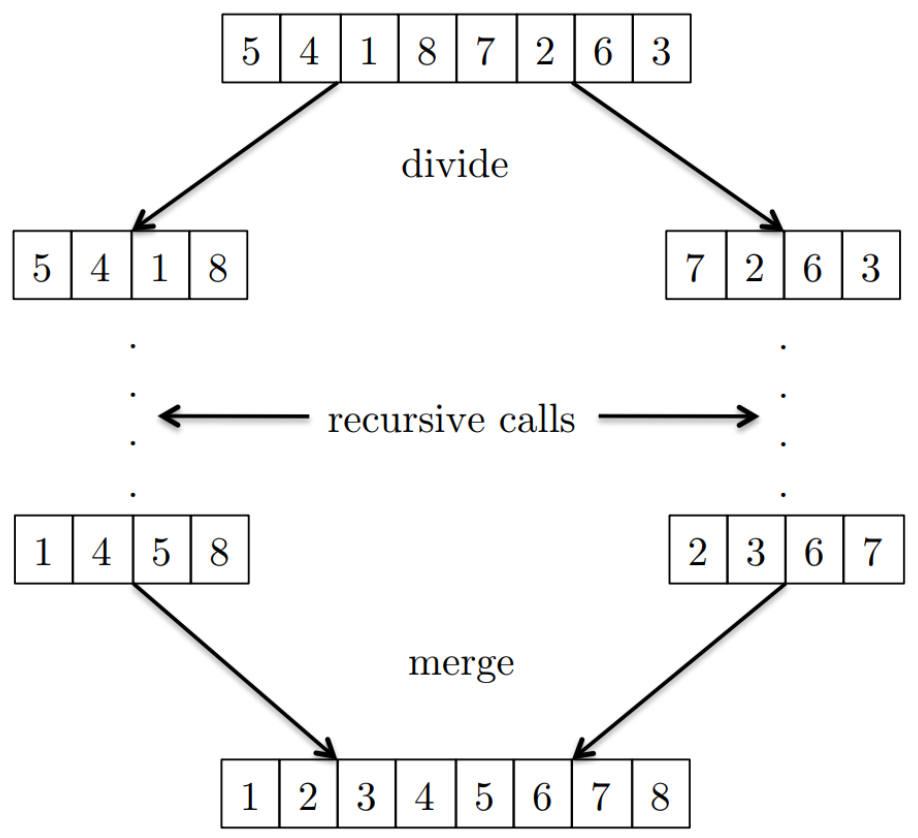
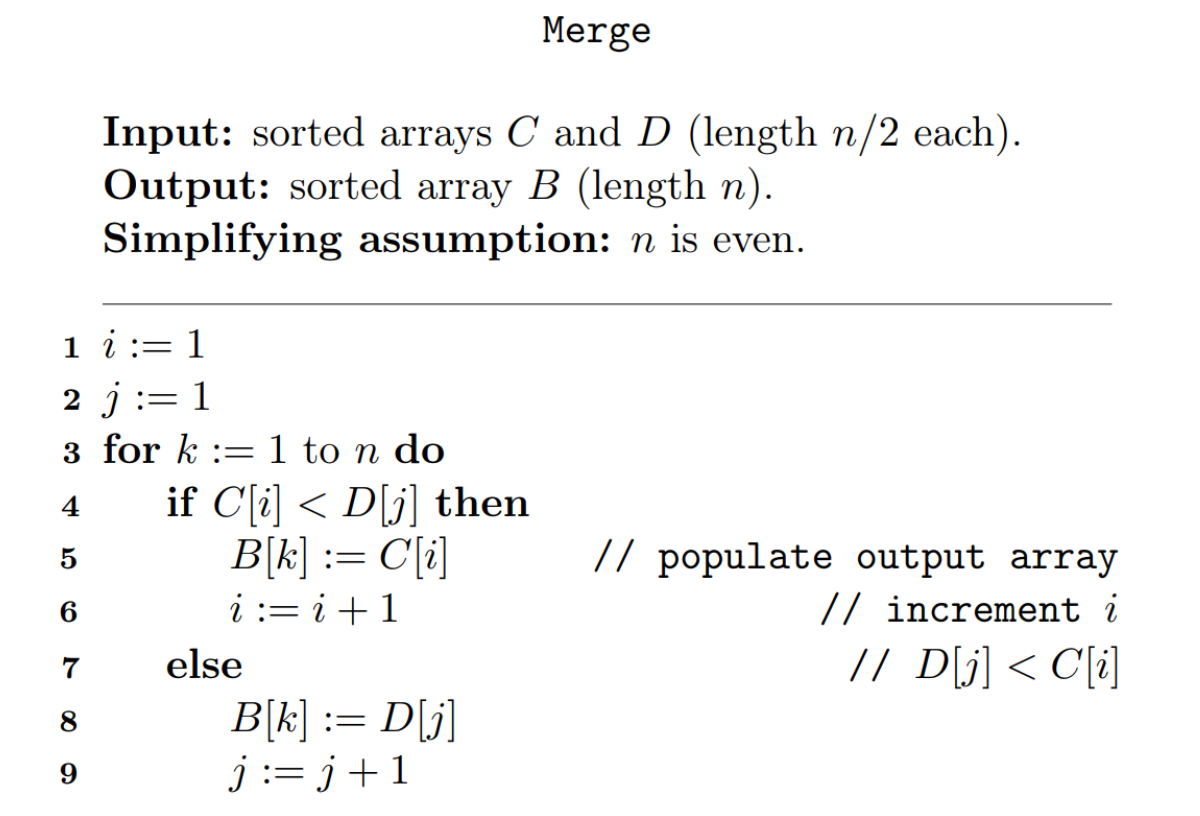
**Standard Recurrence relationships**





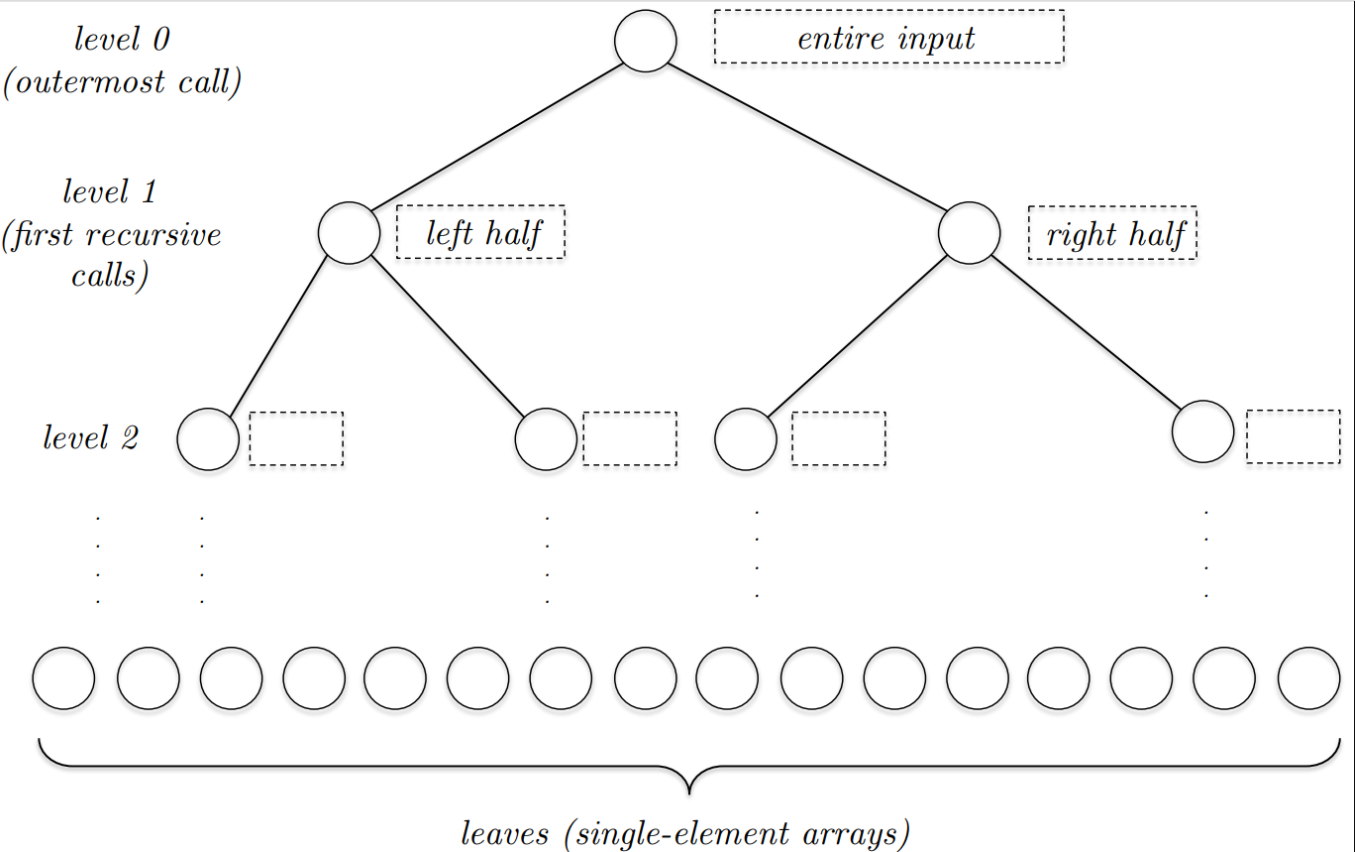


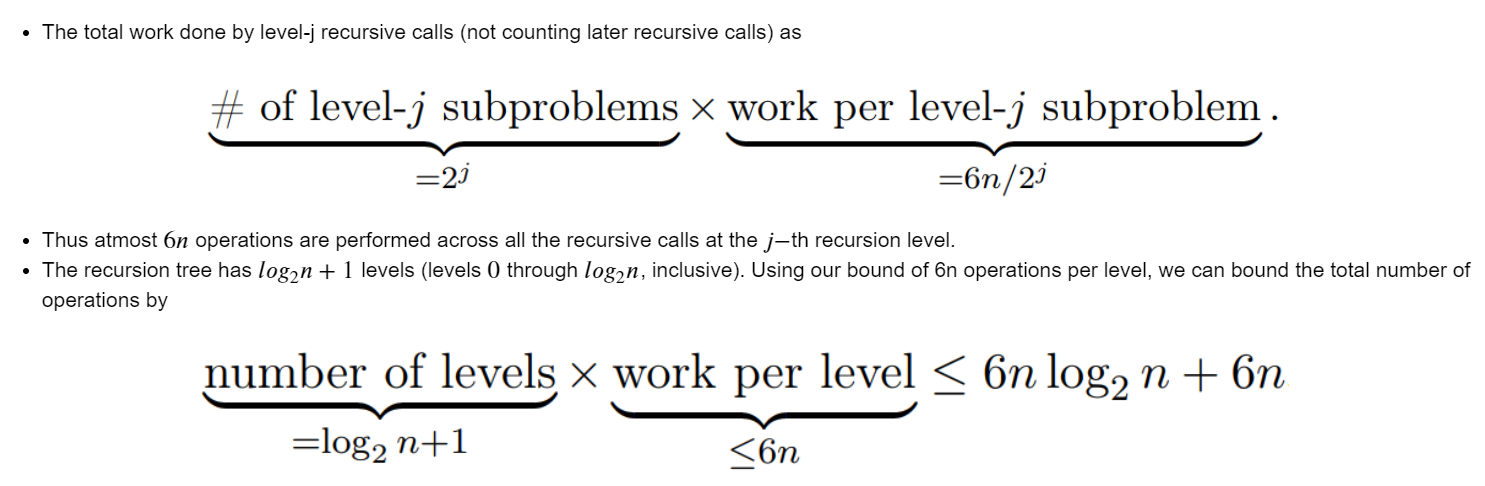


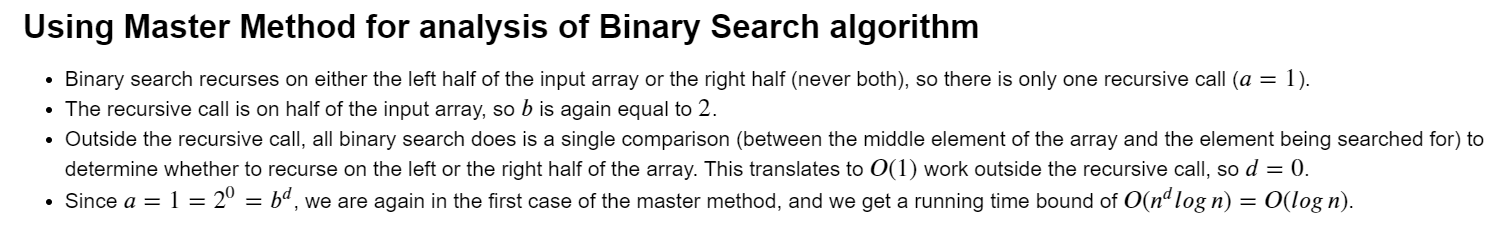
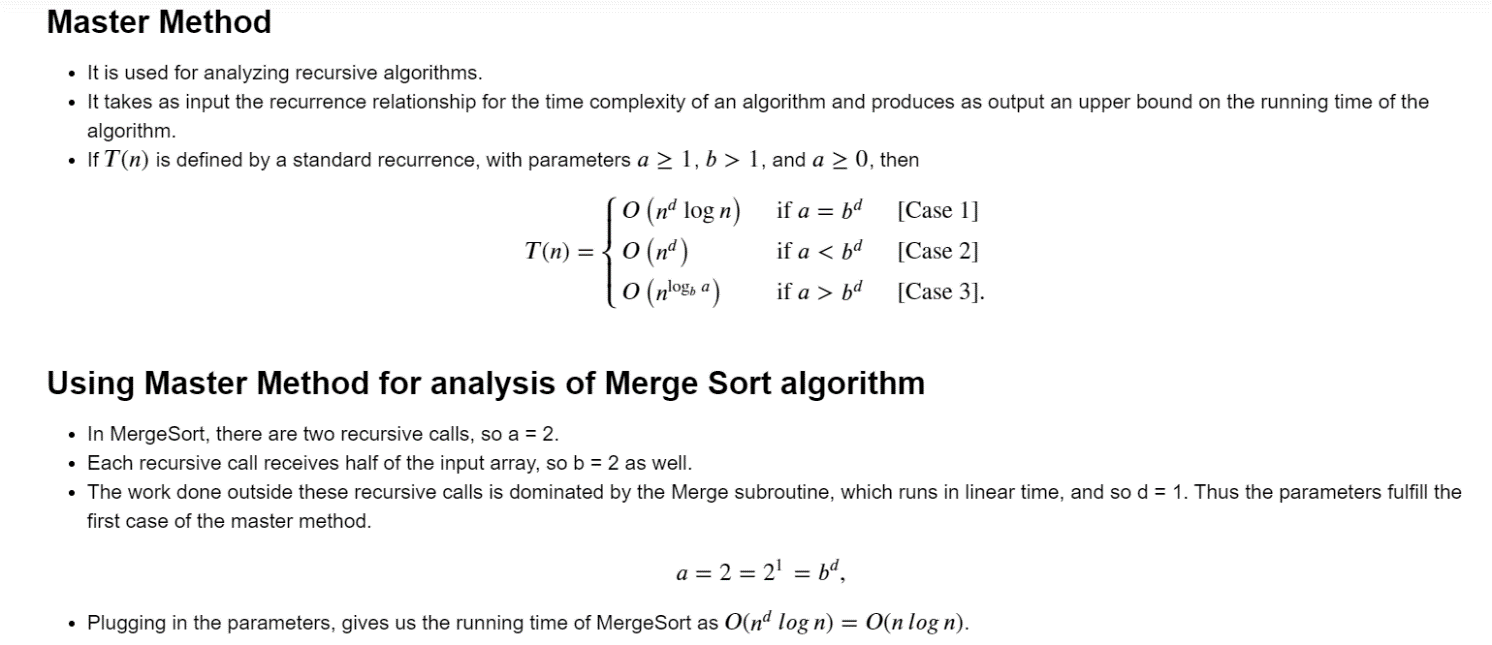


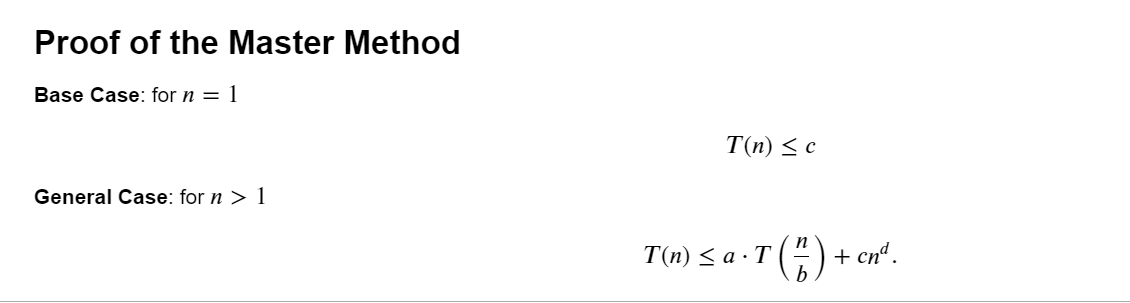
* The Merge subroutine performs at most 4𝑙+2 operations to merge two sorted arrays of length 𝑙/2 each.
* For 𝑙≥1,4𝑙+2≤6𝑙. That is, 6𝑙 is also a valid upper bound on the number of operations performed by the Merge subroutine.
* For every pair of sorted input arrays C, D of length 𝑙/2, the Merge subroutine performs at most 6l operations.

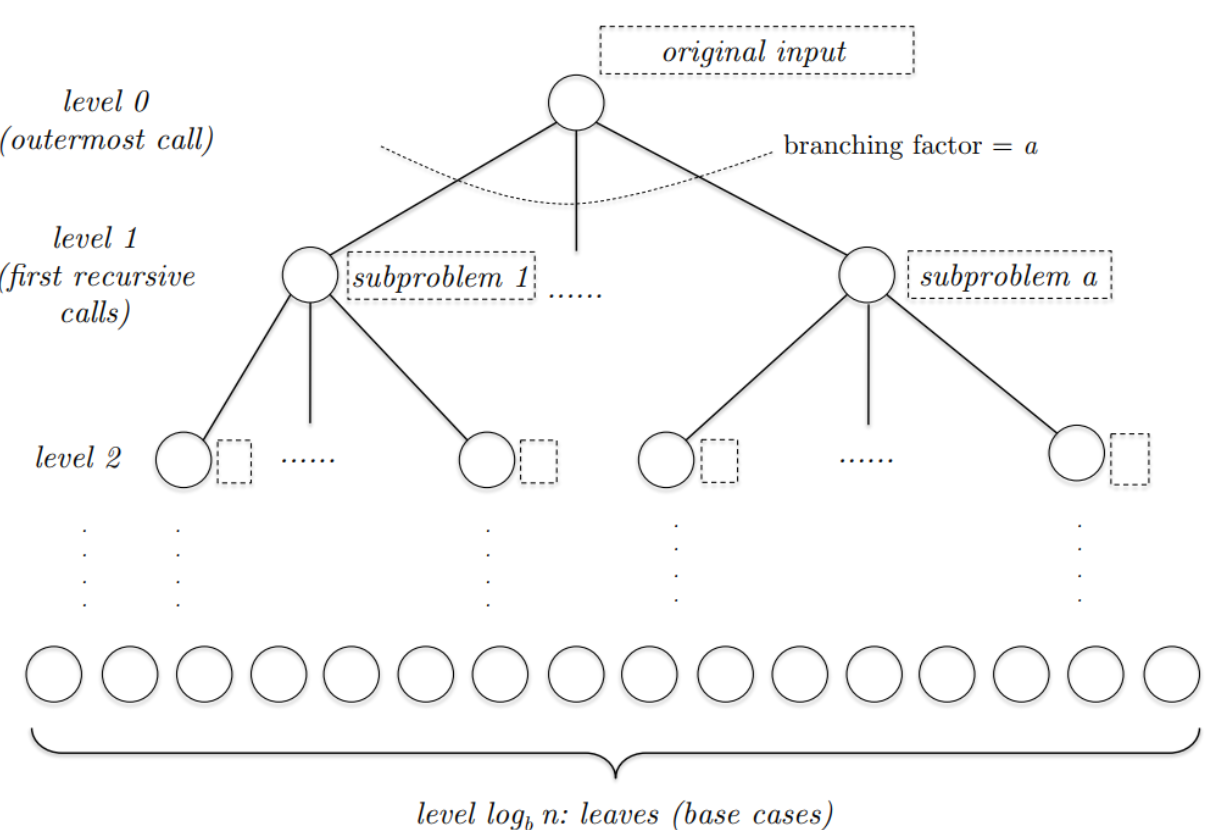
## Running time of Merge Sort



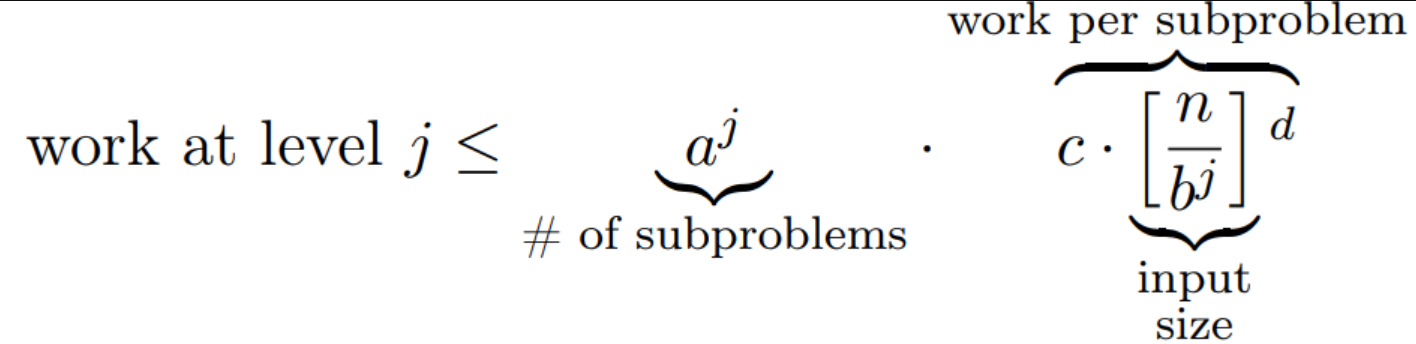


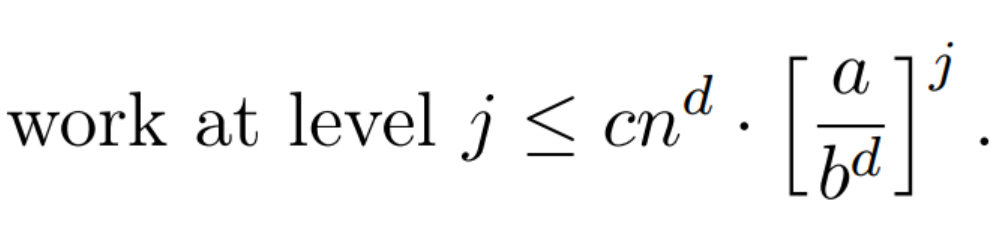


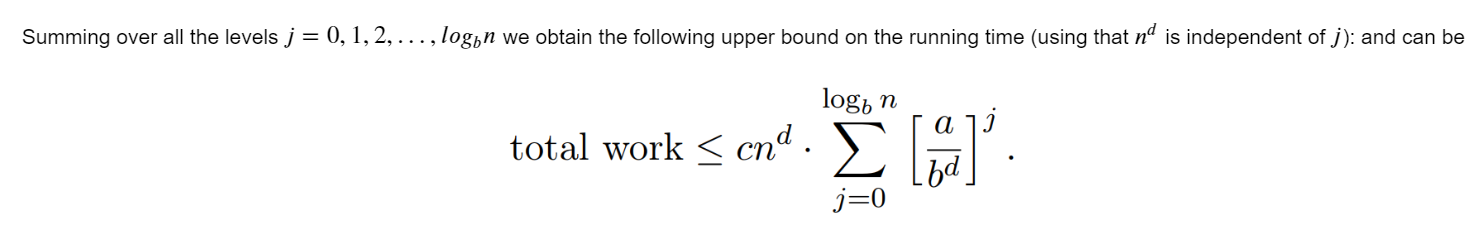


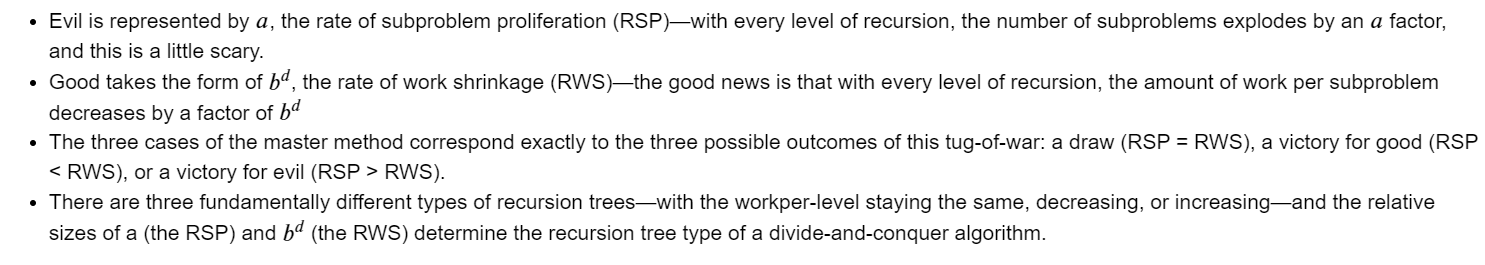


There are 𝑎𝑗 different subproblems at level 𝑗, each with an input size of 𝑛/bj



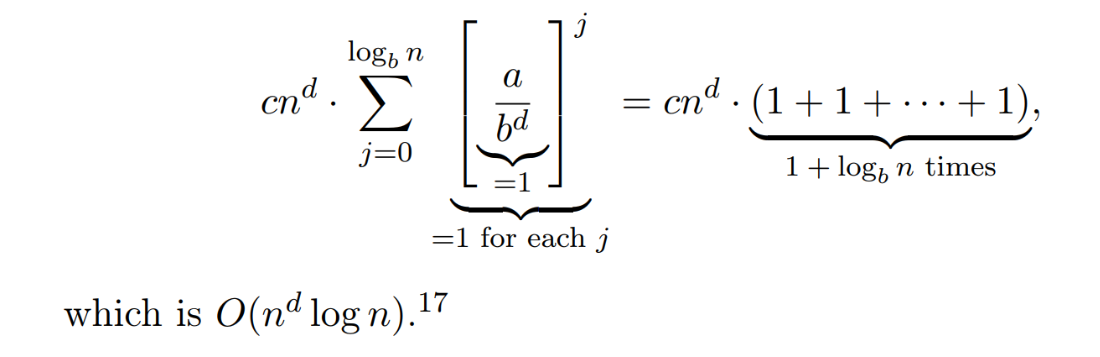






## Case-I

* Consider the first case, when 𝑎=bd and the algorithm performs the same amount of work at every level of its recursion tree.
* We certainly know how much work is done at the root, in level 0 — 𝑂(𝑛𝑑), as explicitly specified in the recurrence.
* With O(nd) work per level, and with 1+logbn=O(log n) levels, we should expect a running time bound of O(nd log n) in this case



## Case-II

* In the second case, a<bd and the forces of good are victorious— the amount of work performed is decreasing with the level.
* Thus more work is done at level 0 than at any other level.
* The simplest and best outcome we could hope for is that the work done at the root dominates the running time of the algorithm.
* Since O(nd) work is done at the root, this best-case scenario would translate to an overall running time of O(nd)
* Set a/bd; since a, b, and d are constants (independent of the input size n), so is r.
* The upper bound becomes:

