|  |  |
| --- | --- |
| **Master:**  T(n) = aT(n/b) + f(n) where, a ≥ 1, b > 1, and f(n) > 0  **case 1:** if f(n) = O(nlogba -ε) for some ε > 0, then: T(n) = Θ(nlogba)  **case 2:** if f(n) = Θ(nlogba), then: T(n) = Θ(nlogba lgn)  **case 3:** if f(n) = Ω(nlogba +ε) for some ε > 0, and if  af(n/b) ≤ cf(n) for some c < 1  and all sufficiently large n, then:  T(n) = Θ(f(n))  **Muster:**  **Recurrence:**T(n) = 4T(n-1) + c  By the Muster Method: T(n) = a T(n-b) + f(n)  *a* = 4, b= 1, f(n) = c so d = 0. f(n) = Θ(n0)  if f(n) is O(n^d)    Therefore, T(n) is Θ(n0 4n/1) = Θ(4n).  **MergeSort Iteration**  **T(n) = n + 2T(n/2)**  T(n) = n + 2T(n/2)  = n + 2(n/2 + 2T(n/4))  = n + n + 4T(n/4)  = n + n + 4(n/4 + 2T(n/8))  = n + n + n + 8T(n/8)  … = in + 2iT(n/2i) stop at i = lgn  = nlgn + 2lgnT(1)  = nlgn + nT(1)  = Θ(nlgn)  **Recursion Tree:**  T(n) = T(n/4) + T(n/2) + n2  n2 n2  (n/4)2, (n/2)2 ++++ 5/16\*n 2  (n/16)2, (n/8)2, (n/8)2, (n/4)2 +++ 25/256\*n 2  total <= n2 (1 + 5/16 + (5/16)2 + (5/16)3 etc…  for parenth vals less than 1, T(n) = O(total) = O(n2)  for parent vals (x) more than 1, T(n) = (1-xn+1)/(1-x) \* total | **Log**  LogbA + logbC = logb(A\*C)  LogbA - logbC = logb(A/C)  A\* LogbC = logb(CA)  LogbA = logcA/logcB  2lgn = n  lg(2n) = n  **Summation**  **n**  **E i =** n(n+1) / 2  **i = 1**  n  E (i + 1) = theta(n2)  i = 0  **Order of alg growth**  Constant  sqrt(n)  N  The above summation  n2n/2  2n  n!  BinarySearch:  In order to find whether two elements exist whose sum is x, we can cycle through each index of the array, subtract x from the value in that index, and BinarySearch the array for the result.  **min\_max (array)**  **if array.length == 1**  **return [array[0], array[0]]**    **mm1 = min\_max(array[0]…(array.length/2)-1)**  **mm2 = min\_max(array.length/2…array.length-1)**    **if (mm2[0] < mm1[0])**  **mm1[0] = mm2[0]**  **if (mm2[1] > mm1[1])**  **mm1[1] = mm2[1]**    **return mm1** |

|  |  |
| --- | --- |
| **Longest common subsequence: (elements don’t have to be consecutive)**  LCS-Length(X, Y)  1. m = length(X) // get the # of symbols in X  2. n = length(Y) // get the # of symbols in Y  3. for i = 1 to m c[i,0] = 0 // special case: Y0  4. for j = 1 to n c[0,j] = 0 // special case: X0  5. for i = 1 to m // for all Xi  6. for j = 1 to n // for all Yj  7. if ( Xi == Yj )  8. c[i,j] = c[i-1,j-1] + 1  9. else c[i,j] = max( c[i-1,j], c[i,j-1] )  10. return c  theta(mn) rather than theta(2n)  **Longest Increasing Subsequence:**  L[n] = 1 + max{L[i]: for all j such that Aj < An  L0 = 0 (base case)  length of LIS = max{L[1], L[2], …, L[n]}  theta(n2)  **Longest Palindromic Subsequence:**  if(input[i]) == input[j])  T[i][j] = T[i+1][j-1] + 2  else  T[i][j] = max (T[i+1][j], T[i][j-1]  **knapsack 0-1: (or w = d and r = f for days and fee)**  Base case  0 limit = 0 reward fill in top row  0 items = 0 reward fill in left column  for i = 1 to limit (row)  for j = 1 to limit (col)  if wi > j  sol[i,j] = sol[i-1, j]  else  sol[i,j] = max{sol[i-1,j], sol[i-1, j-wi] + ri}  time is theta(mn) | **Coin change:**  min\_coins (V, A)  coins[][] is 2d array of size V.length down, A + 1 acr  for col = 0 to A  for row = 0 to length of V  if col == 0, coins[row, col].num = 0*init left col*  else if row == 0, coins[row, col].num = col ^*init top row*  else if V[row] <= col,  option0 = 1+coins[row, col - V[row]].num  option1 = coins[row - 1, col].num  if option0 <= option1  coins[row, col].num = option0  coins[row, col].op = 0  else  coins[row, col].num = op1  coins[row, col].op = 1  else, coins[row, col] = coins[row - 1, col]  time is theta(mn)  **Car with penalty**  Let S[j] be the min total penalty when you go hotel j  minpen[0] = 0  for i = 1 to n  minpen[i] = inf  for j = 0 to (i-1)  minpen[j] = min{minpen[j] + (200-xi-xj)2, minpen[i]}  **CanoeSequence**  n = number of rows in R  C[1] = 0, P[1] = 0  for i = 2 to n  min = R[1, i]  P[i] = 1  for k = 2 to i – 1  if C[k] + R[k,i] < min  min = C[k] + R[k,i]  P[i] = k  C[i] = min  return P  A problem is said to have [optimal substructure](http://en.wikipedia.org/wiki/Optimal_substructure) if the globally optimal solution can be constructed from locally optimal solutions to subproblems. |