HW2

1.

Our objective is to remove the auxiliary variables A and B, specifically in Algorithm 2.2 of the book Step 2 of B(h, j) of the prefix sum and Step 1 A(j) initial set up. We can do this by making A the only array of values with the prefix sums stored in the children nodes. We essentially do backward and forward transversal at the same time with the prefix sum, since the results of the forward transversal B(h = j) = A(h, j)

Input: Array A of size $n = 2^k$, k some positive integer. C holds
Output: Array A of same size that holds the prefix sum of input elements.

begin

1. for 1<= j <= n pardo
   Set C(0, j) := C(j)
2. for h = 1 to log n do
   For 1<= j <= $n/2^h$ pardo
   Set C(h, j) := C(h – 1, 2j – 1) * C(h – 1 , 2j)

end

2.

This is essentially algorithm 2.10 in the book. Implementation is given here. We partition the general graph into a sorted array of vertices and color a vertex different from its neighbors.

Input: index i of the color c(i) of the node (total nodes n) in the graph S
Output: 3-color of the vertices for a cycle of the graph S

begin

1. for 1<= i <= n pardo
   Set C(i):= i
2. for 1<= i <= n pardo

   Set k to the LSB position where the color of the current vertex c(i) and the color of the vertex next to it c(S(i)) differ by XOR

   Set the color of the current vertex, c(i), to c'(i):= $2k + c(i)_k$
3. Sort the vertices by their colors using radix sort, which pads all binary representation of the colors, looks at the LSB position, and moves the color to the correct index.

4. for i = 3 to 2 ceil(log n) do
   for vertices v of color i pardo

color vertex c(i) with the color from the set {0, 1, 2} using the smallest color that differs from its two neighbors.

end

This algorithm takes O(log n) time in O(n) operations. Steps 1 and 2 take O(1) time with the check of the adjacent vertex color (thus break symmetry) but O(n) operations since we have to use it on all the vertices. Step 3 is done in O(log n) time since we sort the array and independently go through all vertices to recolor them taking O(1) time in Step 4.

3.

Input: Problem P of size n, some integer n.
Output: Solution S

begin

for 1 <= i <= n pardo

Divide n into log(n) blocks
Do algorithm A on log(n) blocks

end

We partition the problem n into log(n) blocks. This is input is n=log(n) into O(log(n)/loglog(n)), or O(loglog(n)/logloglog(n)) which just cancels one log of each numerator and denominator. Doing algorithm A by dividing and partitioning A should separate the problem space into O(n) total operations across all log (n) blocks, if problem P is independent in its factors to create solution S.