**CO322 – Data Structures and Algorithms**

**Writing Assignment**

* What is dynamic programming?

Dynamic programming is a way to solve problems by using memorization and recursion. The special case in dynamic programming is that a problem is separated into similar sub-problems, so that their results can be reused. Most of the times these algorithms are used for optimization. In dynamic programming before solving the sub-problems, it will try to examine the results of the previously solved sub-problems.

As an example, when you try to find nth Fibonacci number, you will get in trouble if n is a bigger number. Because it takes a lot of time to calculate. If you examine what really happen when it calculates that the same calculation occurs many times. When we use dynamic programming, we can memorize previous calculations and use it when it is needed. Knapsack problem, Tower of Hanoi, Shortest path by Dijkstra also used dynamic programming for optimization.

* What is Kadane’s algorithm?

The idea of Kadane’s algorithm is to check all positive contiguous elements of an array. Then keep track of maximum sum continuous segment among all positive segments. The special case is just going through the array once and calculate the maximum sub array sum.

* Can Kadane’s algorithm be considered as dynamic programming?

Let’s take an array with both positive and negative numbers. We can find the maximum sub-array using three technics.

* Using brute force:

Here what is happened is select a sub array and get the sum of the sub-array and return max.

Max\_sub\_array\_sum(array[], n){

For (select a sub-array) {

For (each sub-array) {

For (sum of selected elements) {

Sum = total of element in sub-array

If (sum > answer) answer = Sum}}} return sum}

When the program starts, the ‘answer’ start should be big minus value. In this program the time complexity is O (n^3).

* Using divide and conquer:

Max\_sub\_array\_sum (array [], int n){

If (n == 1) return array [0]

Int m = n/2

Int left\_max\_subarray\_sum = Max\_sub\_array\_sum(array], m)

Int right\_max\_subarray\_sum = Max\_sub\_array\_sum(array], n-m) // here we are passing address of the mth element of array as start

address and number of elements.

Then using two for loops for left and right find the maximum sum of the sub array and return it. Here the time complexity is O (n \* log (n)).

* Kadane’s Algorithm:

Initially we start with empty sub array. And also we use a variable to store the sum of elements in sub-array in left at any stage. And we keep another variable to store the maximum sum so far.

Maximum\_sub\_array\_sum (array [], int n) {

Answer = 0, Temp\_sum = 0

for (0 to end of the loop incrementing i){

if(sum of Temp\_sum and array[I th position] > 0) Temp\_sum = array[i] + Temp\_sum

else { Temp\_sum = 0, Answer = max of Answer and Temp\_sum}

return Answer

}

Here what happens is we store the sum of selected element and keep it safe until a value that is larger than the sum comes. If it happens sum will be assigned to sum. The time complexity of this program is O(n).

In dynamic programming, the main parts are recursion and memorization, which helps to optimize the program. In this above algorithm, we use memorization but for one state to optimize the program. Therefore **Kadane’s algorithm cannot be considered as dynamic programming**.