**Question 1:**

Design a linear-time, non-recursive algorithm for the Maximum-subarray problem

discussed in our lecture. Given an array A[1,…,n], start at the left end of the array, and

progress toward the right, keeping track of the maximum subarray seen so far. Suppose

a maximum-subarray of A[1,…,j] is known, the maximum-subarray ending at index j+1

should be either a maximum-subarray of A[1,…,j] or a subarray A[i,…,j+1], for some 1 ≤ 𝑖 ≤ 𝑗 + 1 . A key observation: determine a maximum-subarray of A[i,…,j+1] costs a

constant time if a maximum subarray of A[i,…,j] is known. Hint: you may want to

construct a table to store the key information of a maximum-subarray of A[1,…,j], as j

progresses from 1 to n. Please write down the pseudo-code that describes the

algorithm.

**Solution:**

Pseudo-code is as follows

Step1: Input array

Explain:

e.g: array=[-2,1,3,-2,-8,9,-5,-7]

Step2: Initialize variables

0

explain:

maxSum is a variable that calculate the sum of Maximum-subarray

currentSum is a variable that calculate the current sum of Maximum-subarray

start\_subarray is means the position occupied by the value of the first element of the sub-array in the given array.

end\_subarray is means the position occupied by the value of the last element of the sub-array in the given array.

Step 3: calculate the maxSum, start\_subarray, end\_subarray by using “for loop”

  Explain:

“max(currentSum+array[i], array[i]) = array[i]” means array[i] > currentSum, so start point of maximum-subarray must be updated.

“ max(maxSum, currentSum) is not maxSum” means that currentSum > maxSum, so maxSum must be updated. Therefore last point of maximum-subarray must be updated

Step 4: print the results

Explain:

This means get elments from start\_subarray to endsubarray of array

Explain:

This means the print the subarray calculated.

**Question 2:**

Prove by Mathematical Induction:

a) Given 𝑇(𝑛) = 𝑇(𝑛 − 1) + 𝑛 , 𝑇(𝑛) ∈ 𝑂 (𝑛^2 ) .

b) Given 𝑇(𝑛) = 𝑇(⌈𝑛/2⌉) + 1 , 𝑇(𝑛) ∈ 𝑂 (𝑙 𝑜𝑔𝑛 ).

c) Given 𝑇(𝑛) = 2𝑇(⌊𝑛/2⌋) + 𝑛, 𝑇(𝑛) ∈ Ω(𝑛 𝑙𝑜𝑔 𝑛).

**Solution:**

𝑇(𝑛) = 𝑇(𝑛 − 1) + 𝑛 = [𝑇(𝑛 − 2) + (𝑛 – 1)] + 𝑛 = 𝑇(𝑛 − 2) + [𝑛 + (𝑛 − 1)]=

= [𝑇(𝑛 – 3) + (𝑛 – 2)] + [𝑛 + (𝑛 − 1)] = 𝑇(𝑛 – 3) + [𝑛 + (𝑛 − 1) + (𝑛 – 2)]=

== 𝑇(1)+ = 𝑇(1) + = 𝑇(1) + .

By the way

Therefore

Hence Proved

b)

𝑇(𝑛) = 𝑇([𝑛/2])+1=[𝑇([𝑛/]) + 1]+1= 𝑇([𝑛/])+2=[𝑇([𝑛/]) + 1]+2= 𝑇([𝑛/])+3==

= 𝑇([𝑛/])+

=

Hence Proved

c) Given 𝑇(𝑛) = 2𝑇(⌊𝑛/2⌋) + 𝑛, 𝑇(𝑛) ∈ Ω(𝑛 𝑙𝑜𝑔 𝑛)

𝑇(𝑛) = 2𝑇([𝑛/2])+=2{𝑇([𝑛/])+]}+ = 𝑇([𝑛/])+𝑇([𝑛/])+2n=

= 𝑇([𝑛/])+𝑇([𝑛/])+3n

= 𝑇([𝑛/])+

**Question 3:** Using the master method to show that the solution to the recurrence T(n)=4T(n/3)+n

is T(n) ∈ 𝜃() .

**Solution:**

the master theorem is as follows

condition:

T(n) = aT(n/b) + f(n),

Where

n = size of input

a = number of subproblems in the recursion

n/b = size of each subproblem. All subproblems are assumed to have the same size.

f(n) = cost of the work done outside the recursive call, which includes the cost of dividing the problem and cost of merging the solutions

Here, a ≥ 1 and b > 1 are constants, and f(n) is an asymptotically positive function.

In this case, the following facts are established:

1) , then

2) , then

3) , then

is a constant.

From the above theorem, we substitute a=4, b=3, , .

Then .

Therefore from the 1) of the master theorem,

Hence Proved

**Question 4:**

Given the solution to problem 3, try to prove the same by using the substitution method.

In particular, show that a substitution proof with the assumption T(n) ≤ 𝑐𝑛 log3 4 fails.

Then show how to subtract off a low-order term to make a substitution proof work.

**Solution:**

𝑇(𝑛) ===

=

From (1), the term is calculated as following

From (2), ,

(4)

From (3) (5), the follow disequilibrium expression holds as

Hence Proved

**Question 5:**

Use a recursion tree to determine a good asymptotic upper bound on the recurrence T(n)=T(n-1)+T(n/2)+n. Then, use the substitution method to verify your answer.

**Solution:**