4.1-1

What does FIND -MAXIMUM -SUBARRAY return when all elements of A are negative?

Solution:

It will return the least negative position.

Because the sums will always being counted from a positive element, and stop at a positive element, if all the elements are negative, the sum will not start to count until the end, since the algorithm doesn't consider length zero sub arrays, so it must have length 1, thus it will give us the least negative position back.

4.1-2

Write pseudocode for the brute-force method of solving the maximum-subarray problem. Your procedure should run in θ(n^2) time.

Solution:

Brute Force Method on A

```
1.  start = 1
2.  end = 1
3.  max = A[ 1 ]
4.  sum = 0
5.  For i = 0 to A.length – 1
6.      For j = i to A.length – 1
7.          sum += A[ j ]
8.          If max < sum
9.              start = i
10.             end = j
11.             max = sum
12.         End if
13.     End for
14. End for
```

Then the subarray will be the part [start, end], the maximum will be the 'max'

4.3-1

Show that the solution of T(n) = T(n – 1) + n is O(n^2)

Solution:

Assume that T(n) <= cn^2,

Then T(n) = T(n – 1) + n

<= c(n-1)^2 + n

= cn^2 +(1−2c)n+1

<= cn^2 + 2 – 2c

<= cn^2

The first inequality comes from the inductive hypothesis

the second from the fact that n ≥ 1 and 1 − 2c < 0. The last from the fact that c ≥ 1.

4.3-2

Show that the solution of T(n) = T([n/2]) + 1 is O(lgn)

Solution:

Using substitution method we want to prove that T(n) <= c lg(n-b).

Assume this holds for [n/2], We have:

T(n) <= c lg([n/2 – b]) + 1

< c lg(n /2-b+ 1) + 1

= c lg((n -2b+2)/2) + 1

= c lg(n- 2b+ 2) – c lg2 + 1

<= c lg(n – b)

The last inequality requires that b >=2 and c > =1.

4.3-6

Show that the solution to T(n) = 2T([n/2] + 17) + n is O(n lgn)

Solution:

Using substitution method we want to prove that T(n) <= c nlgn -d

Choose n1 such that n≥n1 implies n/2 + 17 ≤ 3n/4. We'll find c and d then suits the equation

T(n) <= c lg([n/2]+17) + n

<= 2(c(n/2) + 17) lg(n/2 + 17 -d) + n

<= cn lg(n/2 + 17) + 17c lg(n/2 + 17) – 2d + n

<= cn lg(3n / 4) + 17c lg(3n/4) -2d + n

= cn lgn -d + cn lg(3/4) + 17c lg(3n /4) -d + n

Take c = -2/lg(3/4) and d = 34. Then we have T(n) <= cnlgn-d + 17clgn – n.

Since lg(n) = o(n), there exsits n2 such that n > n2 implies n >= 17n lgn,

Letting n0 = max{n1, n2}, we have n >= n0 implies T(n) <= cnlgn -d.

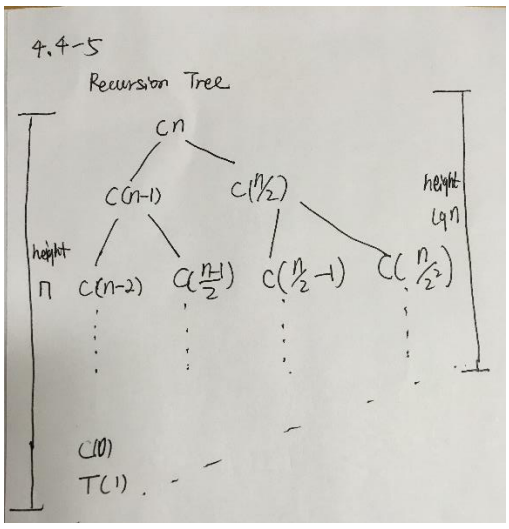
Therefore T(n) = O(nlgn).

4.4-5

Use a recursion tree to determine a good asymptotic upper bound on the recurrence

T(n) = T(n – 1) + T(n /2) + n. Use the substitution method to verify your answer.

Solution:



From the recursion tree we can see that it is not symmetric, and the max height is the left most branch, which is n, min height is the left most branch, which is lgn.

Lets Solve it by substitution.

T(n) = T(n-1) + T(n/2) + n

= T(n-2) + T(n/4) + (n-1) + n

= T(n-3) + T(n/8) + (n-2) + (n-1) + n

= T(n-4) + T(n/16) + (n-3) + (n-2) + (n-1) + n

= ----------

= T(n-k) + T(n/(2^k)) + (n-(k-1)) + (n-(k-2)) + ---------- + (n-2) + (n-1) + n

Now consider n = 2^k, then k = lg n

= T((2^k) - k) + T((2^k)/(2^k)) + (n - k + 1) + (n - k + 2) + ---------- + (n - 1) + n

Now Put value of K

= (2^(lg n) - lg n) + 1 + (n - lg n + 1) + (n - lg n + 2) + ---------- + n

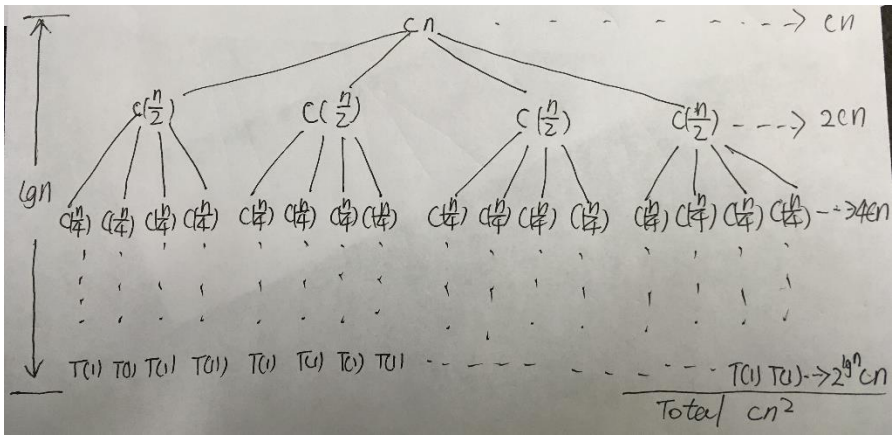
Here series {(n - lg n + 1) + (n - lg n + 2) + ---------- + (n - 1) + n} are in AP with (lg n) element then Hence Sn = ((lg n)/2) *(n -lg n + 1 + n) = O(n.log n)

Hence, T(n) = 2^(log n).

4.4-7

Draw the recursion tree for T(n) = 4 T([n / 2]) + cn, where c is a constant, and provide a tight asymptotic bound on its solution. Verify your bound by the substitution method.

Solution:



From the recursion tree, we could assume that T(n) is θ(n^2).

Then we could test it by substitute method as follows:

Suppose that T(n) <= dn^2.

Then T(n) = 4T([n / 2]) + cn

<= 4 d(n/2)^2 + cn

= dn^2 + cn

Then it <= dn^2, whenever d + c / n <= 1, and this is true for a large enough n as long as d < 1.

4.5-3

Use the master method to show that the solution to the binary-search recurrence T(n) = T(n/2) + θ(1) is T(n) = θ(lgn).

Solution:

In the binary search case, when we apply the master method to it,

a = 1,b = 2

=> n ^(logb(a)) = n ^ (log2(1)) = n^0 = 1

So θ(n ^(logb(a))) = θ(1), this is the second situation.

We have a final result of θ(n ^ (log2(1))) = θ(lg(n)).

4.5-4

Can the master method be applied to the recurrence T(n) = 4T(n / 2) + n^2 lgn ?

Why or why not? Give an asymptotic upper bound for this recurrence.

Solution:

The master method cannot be applied here. Observe that lg b(a) = lg 2(4) = 2, and f(n) = n^2 lgn.

It is clear that cases 1 and 2 do not apply. Furthermore, although f is asymptotically larger than n^2 , it is not polynomially larger, so case 3 does not apply either. We'll show T(n) = O(n^2 (lgn)^2).

To do this, we'll prove inductively that T(n) ≤ n^2 (lgn)^2.

T(n) = 4T(n/2) + n^2 lgn

$\leqslant$ 4((n/2)^2 (lg(n/2))^2) + n^2 lgn

= n^2 (lgn − lg2)^2 + n^2 lgn

= n^2 (lgn)^2 – n^2 (2lgn − 1 − lgn)

= n^2 (lgn)^2 – n^2 (lgn − 1)

$\leqslant$ n^2 (lgn)^2

provided that n ≥ 2.

4-1 Recurrence examples

Give asymptotic upper and lower bounds for T(n) in each of the following recurrences. Assume that T(n) is constant for n<=2. Make your bounds as tight as possible, and justify your answers.

T(n) = 2T(n/2) + n^4

Solution:

a = 2, b = 2, n^(logb(a)) = n^(log2(2)) = n < f(n), which is n^4, this is the third case in master theorem.

so T(n) = θ(f(n)) = θ(n^4)

T(n) = T(7n/10) + n

Solution:

a = 7, b = 10, n^(logb(a)) = n^(log10(7)) < f(n), which is n, this is the third case in master theorem.

so T(n) = θ(f(n)) = θ(n)

T(n) = 16T(n/4) + n^2

Solution:

a = 16, b = 4, n^(logb(a)) = n^(log4(16)) = n^2 = f(n), which is n^2, this is the second case in master theorem. so T(n) = θ(n^(logb(a)) lgn) = θ(n^2 lgn)

T(n) = 7T(n/3) + n^2

Solution:

a = 7, b = 3, n^(logb(a)) = n^(log3(7)) < n^2, this is the third case in master theorem.

so T(n) = θ(f(n)) = θ(n^2)

T(n) = 7T(n/2) + n^2

Solution:

a = 7, b = 2, n^(loga(b)) = n^(log2(7)) > f(n), which is n^2, this is the first case in master theorem.

so T(n) = θ(n^(loga(b)) = θ(n^(lg(7)))

T(n) = 2T(n/4) + n^(1/2)

Solution:

a = 2, b = 4, n^(logb(a)) = n^(log4(2)) = n^(1/2) = f(n), which is n^(1/2), this is the second case in master theorem. so T(n) = θ(n^(logb(a)) lgn) = θ(n^(log4(2)) lgn) = θ(n^(1/2) lgn)

T(n) = T(n-2) + n^2

Solution:

This is not suitable for master theorem, let d = m mod 2, we can easily see that the exact value of T(n) is n (n + 1) (n + 2) / 6 + n (n+ 2) d / 2 + d^2 n / 2, for which its increasing rate is dominated by its leading term, which is n^3/ 6, so T(n) = = θ(n^3)