Homework #3

1. (10 points) **Exercise 4.1-2: Brute-force method of sovling the maximum-subarray problem.**

Ans:

The brute-force method to find the maximum-subarray of a given array *A[1..n]* is to check all the possible sub-array A[i..j] where $1 \leq i \leq j \leq n$and record the result with the maximum sum.

FIND-MAXIMUM-SUBARRAY (A)

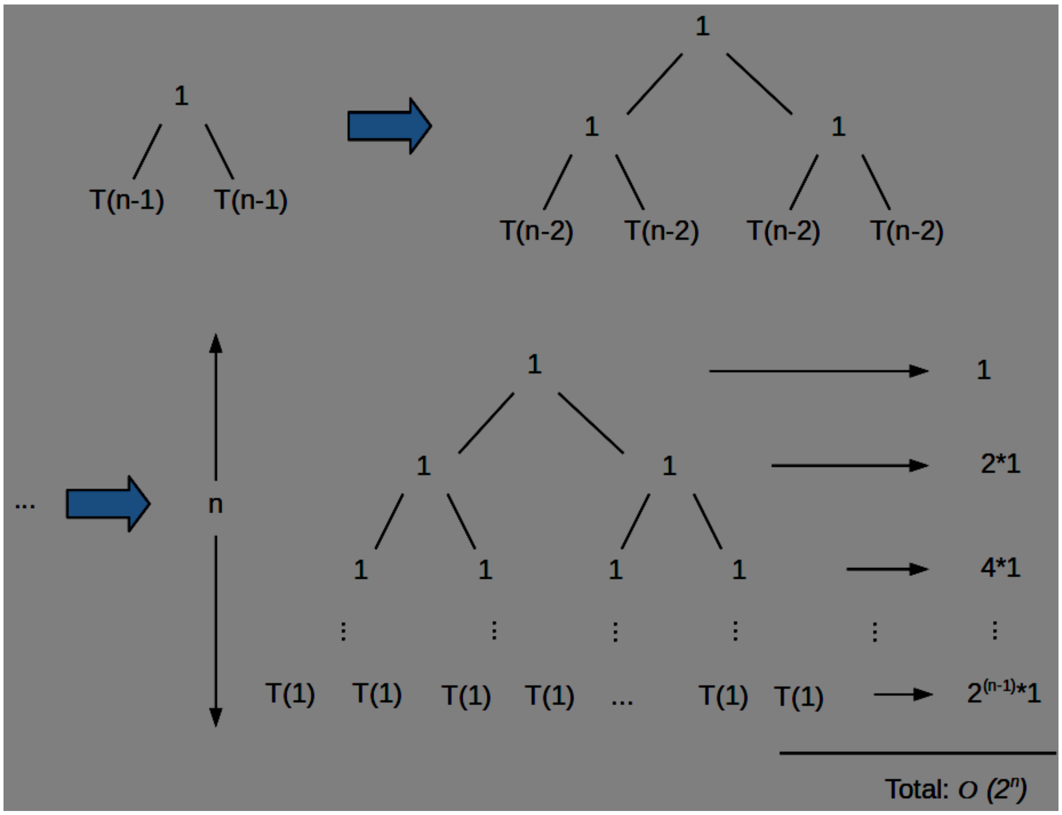
```
1.  n = A.length,  max_sum = −∞
2.  for i = 1 to n
3.      sum = 0
4.      for j = i to n
5.          sum = sum + A[j]
6.          if sum > max_sum:
7.              low = i
8.              high = j
9.              max_sum = sum
10. return (low,  high, max_sum)
```

The operations in line 7~11 only cost constant time. Therefore, in the worst-case, the running time is: $T(n) = c(n + (n-1) + \cdots + 1) = \frac{c}{2}n(n+1) = \Theta(n^2)$,where n is the length of the array and c is a constant.

2. (10 points) **Exercise 4.4-4: Use a recursion tree to determine a good asymptotic upper bound on the recurrence *T(n) = 2T(n-1) + 1*. Use the substitution method to verify your answer.**

Ans:

The recursion tree for *T(n) = 2T(n-1) + 1* is:



According to the recursion tree, the total cost is:

$$T(n) = \sum_{i=1}^{n} 2^{i-1} = \frac{1 \cdot (2^{n-1} - 1)}{2 - 1} = 2^{n-1} - 1 = O(2^n)$$

To verify our answer, we use substitution method: we try the solution *T(n)=O(2ⁿ)* and guess $T(n) \leq c2^n - d$,then we have:$T(n) \leq 2(c2^{n-1} - d) + 1 = c2^n - 2\text{d} + 1 \leq c2^n - d$,as long as $d \geq 1$.Then for$n \geq 1$and$c \geq d$we can prove that *T(n)=O(2ⁿ)*.

3. (10 points) **Exercise 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.**

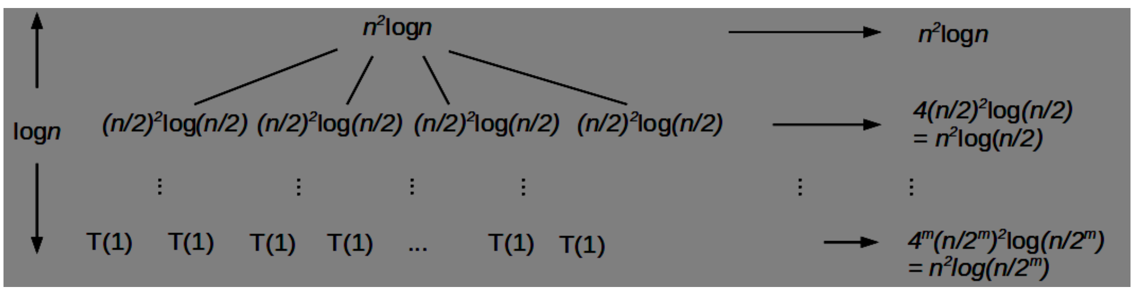
Ans:

According to the master theorem we know that *a=4*, *b=2*, and *f(n)* = $n^2 lgn$, so *f(n)* seems to be larger than $n^{\log_b a} = n^2$.However, we can NOT apply the master method for the following reasons:

1) *f(n)* is not polynomially larger than $n^{\log_b a}$,where it requires $f(n) = \Omega(n^{\log_b a + \varepsilon})$.

2) The condition (*for some constant c<1 and all sufficiently large n)* is NOT satisfied since:

However, we can use the recursion tree method:



The total cost is:

$$T(n) = n^2 \sum_{i=1}^{\log n} \log\frac{n}{2^i} = n^2(\log^2 n - \sum_{i=1}^{\log_2 n} i) = n^2(\log^2 n - \frac{1}{2}\log n(\log n + 1))$$
$$= O(n^2 \log^2 n).$$

4. (70 points) **Problem 4-1: Recurrence examples**

Ans:

a. *T(n) = 2T(n/2) +* n^4: using the master theorem, *a=2, b=2*, and *f(n)=*n^4, then we know

$f(n) = \Omega(n^{1+\varepsilon})for\varepsilon = 1 > 0and\frac{n^4}{8} \leq cn^4forc = 0.5 < 1$and all sufficiently large *n*. So, $T(n) = \Theta(n^4)$according to the case 3.

b. *T(n)=T(7n/10) + n:* using the master theorem we have *a=1, b=10/7* and *f(n)=n*, then$f(n) = \Omega(n^{0+\varepsilon})for\varepsilon = 0.5 > 0and0.7\text{n} \leq cnforc = 0.8 < 1$and all sufficiently large *n*. According to the case 3, we have $T(n) = \Theta(n)$.

c. *T(n) = 16T(n/4) +* n^2: using the master theorem we have *a=16, b=4* and *f(n)=*n^2, then$f(n) = \Theta(n^{2+\varepsilon})$.According to the case 2, we have $T(n) = \Theta(n^2 lgn)$.

d. *T(n) = 7T(n/3) +* n^2: using the master theorem we have *a=7, b=3* and *f(n)=*n^2, then$f(n) = \Omega(n^{1.8+\varepsilon})for\varepsilon = 0.1 > 0and\frac{7}{9}n \leq cnforc = \frac{8}{9} < 1$ and all sufficiently large *n*. According to the case 3, we have $T(n) = \Theta(n^2)$.

e. *T(n) = 7T(n/2) +* n^2: using the master theorem we have *a=7, b=2* and *f(n)=*n^2, then$f(n) = O(n^{2.8-\varepsilon}) for \varepsilon = 0.3 > 1$.According to the case 1, we have $T(n) = \Theta(n^{\log_2 7})$.

f. *T(n) = 2T(n/4) +* $n^{1/2}$: using the master theorem we have *a=2, b=4* and *f(n)=*$n^{1/2}$, thenAccording to the case 2, we have

g. *T(n) = T(n-2) +* n^2: using the substitution method, we guess $T(n) = \Theta(n^3)$.

Upper bound: assuming that$T(n) \leq c_1 n^3$then

$$
\begin{aligned}
T(n) &\leq c_1(n-2)^3 + n^2 \\
&= c_1 n^3 - ((6\text{c}_1 - 1)n^2 + 4\text{c}_1 n + 8\text{c}_1) \\
&\leq c_1 n^3,
\end{aligned}
$$

where the last step holds as long as $c_1 > 1/6$.

Lower bound: assuming that $T(n) \geq c_2 n^3$then

$$
\begin{aligned}
T(n) &\geq c_2(n-2)^3 + n^2 \\
&= c_1 n^3 + ((1 - 6\text{c}_2)n^2 - 4\text{c}_2 n + 8\text{c}_2) \\
&\geq c_2 n^3,
\end{aligned}
$$

where the last step holds as long as $c_2=0.1$ for sufficiently large n.

Therefore, we have $T(n) = \Theta(n^3)$.