Problem 1 (2 points) Solving Recurrence Relations
Draw the recursion tree for $T(n)$=4T($\lfloor n/2 \rfloor$) + cn, where c is a constant, and provide a tight asymptotic bound on its solution. Verify your bound by the substitution method.

Recurrence Tree: A recurrence tree is a tree where each node represents the cost of a certain recursive sub-problems. Then we sum up the numbers in each node to get the cost of the entire algorithm.

T(n) cn

n/2 n/2 n/2 n/2 4/2 cn

n/4 n/4 n/4 n/4 …… n/4 n/4 n/4 n/4 $(4/2)^2$ cn

n/8 n/8 n/8 n/8 …………………………… $(4/2)^3$ cn

. . . .

. . . .

T(1) T(1) T(1) T(1) …………………………… T(1) Total Θn^2

Height of tree= log n
Cost of each level = 2^i cn

By using Master Theorem we calculated asymptotic bound:

$T(n)= a\,T(n/b) + \Theta(n^k \log^p n)$ where $a\geq1$, $b>1$, $k\geq0$, P is real number

here a=4, b=2, k=1, p=0
$a>b^k$ so $T(n)= \Theta(n\log^4 2)$
$T(n)= \Theta(n^2)$

Proof:
T(n) = 4T(n/2) + cn equation - 1
T(n/2) = 4T(n/4) + cn/2 equation - 2

$T(n/4) = 4T(n/8) + cn/4$ equation – 3

Substitute equation 2 in 1 we will get

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

$= 16T(n/4) + 3cn$

$= 4^2T(n/2^2) + 3n$

Substitute equation 3 in above equation

$T(n) = 16T[4T(n/8) + cn/4] + cn$

$= 64T(n/8) + 7cn$

$= 4^3T(n/2^3) + 7n$

By observing pattern, we found out

$T(n)=4^iT(n/2^i) + tcn$ Let $n/2^i = 1 \Rightarrow n=2^i \Rightarrow \log_2 n = i$

$= 4^{\log_2 n} T(1) + tcn$

$= n\log^4{}_2 + tcn$

$= n^2 + tcn$

$T(n) = O(n^2)$

Another method:

Guess: $T(n) \le a(n^2 - n)$ a is constant

$T(n) = 4T(\lfloor n/2 \rfloor) + cn \le 4T(n/2) + cn \le 4a((n/2)^2 - n/2) + cn$

$= an^2 - 2an - cn$

$= an^2 – (2a-c)\, n$ when $a>c$

$T(n) \le a(n^2 - n)$

Problem 2 (2 points) Solving Recurrence Relations

Give asymptotic upper and lower bounds for $T(n)$ in each of the following recurrences:

(1) $\boldsymbol{T(n)=8T(n/3) + n^2}$

By using Master Theorem: $T(n)= aT(n/b) + \Theta(n^k \log^p n)$

$a=8, b=3, k=2, P=0$

we found out that $a<b^k$ and $p = 0$ so $T(n) = O(n^k)$

$T(n) = O(n^2)$

(2) $T(n)$=$T(n-1)$ + logn

Solution: T(n) = T(n-1) + logn
T(n-1) = T(n-2) + log(n-1)
T(n-2) = T(n-3) + log(n-2)

By substituting the values of T(n-1) in T(n) we get:
T(n)=T(n-2) + log(n-1) + logn

By substituting the values of T(n-2) in above equation we get:
T(n)= T(n-3) + log(n-2) + log(n-1) + logn
= T(n-3) + log((n.(n-1).(n-2))
We found out the pattern and get

T(n)=T(1) + logn!
= $\log(n/e)^n$
= n(logn-loge)
= n(logn-1.44n)
T(n)= Θ(nlogn)

Problem 3 (4 points) Building a Heap using Insertion (Problem 6 – 1, p. 166 –167)
We can build a heap by repeatedly calling MAX-HEAP-INSERT to insert the elements into the heap.

Consider the following variation on the BUILD-MAX-HEAP procedure:

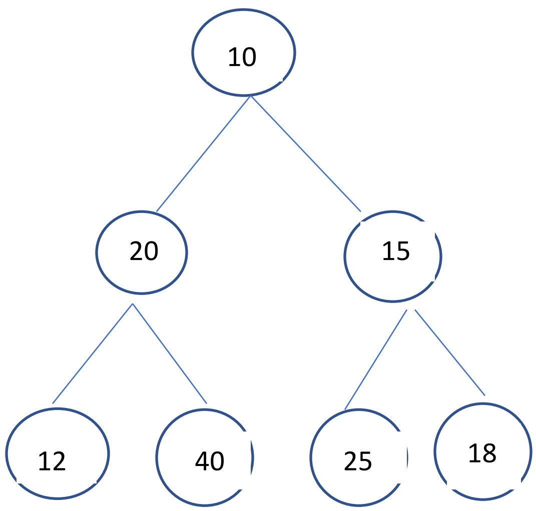
BUILD-MAX-HEAP'(*A*)
1 *A.heap-size* = 1
2 **for** *i* = 2 **to** *A.length*
3 MAX-HEAP-INSERT(*A*, *A*[*i*])

a. Do the procedures BUILD-MAX-HEAP and BUILD-MAX-HEAP' always create the same heap when run on the same input array? Prove that they do or provide a counterexample.

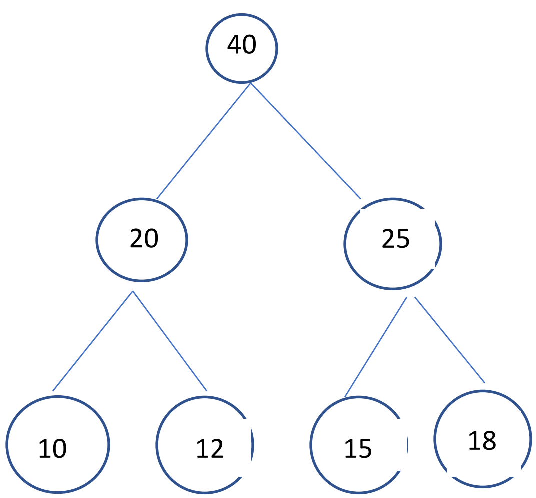
Answer: The procedures BUILD-MAX-HEAP and BUILD-MAX-HEAP’ do not always create the same heap when run on the same input array.
Consider the following counterexample:

10	20	15	12	40	25	18

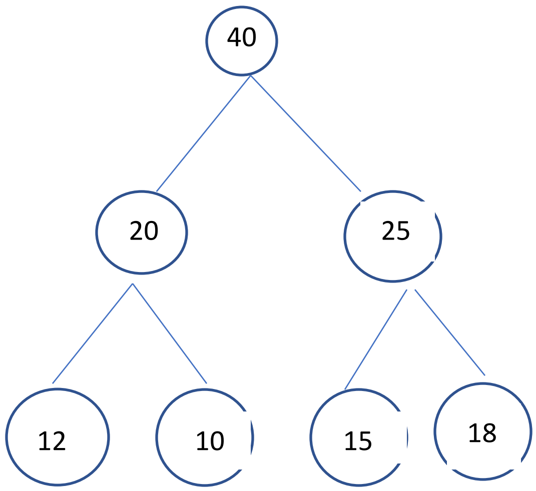
Binary Tree:



BUILD-MAX-HEAP:



BUILD-MAX-HEAP’:



b. Show that in the worst case, BUILD-MAX-HEAP’ requires Θ(*n* lg *n*) time to build an *n*-element heap.

Answer: The initial heap size is 1 and the size is incremented by one after each call to MAX-Heap-Insert. The worst-case running time for Max-Heap-Insert is Θ(lg(heap-size)). Thus, the worst-case running time T(n) for Build-Max-Heap’ is

$$T(n) = \sum_{i=1}^{n-1} \Theta(\log i)$$

$$= \Theta\left(\sum_{i=1}^{n-1} (\log i)\right)$$

$$= \Theta(\log(n-1)\,!)$$
$$= \Theta\big(\Theta(n\log n)\big)$$
$$= \Theta(n\log n)$$