1. (a) False

(b) $O(\ln n)$

(c) False

(d) False

(e) $2^k-1$

(f) False

2.

(a)

$f(n)=O(g(n))$ iff $\exists C>0 \wedge \exists n_0>0$ such $\forall n \geq n_0$, $f(n) \leq cg(n)$

(b)

since $f_1(n)=O(g_1(n)) \Rightarrow \exists C_1>0 \wedge \exists n_1>0$ such $\forall n \geq n_1$, $f_1(n) \leq C_1 g_1(n)$

since $f_2(n)=O(g_2(n)) \Rightarrow \exists C_2>0 \wedge \exists n_2>0$ such $\forall n \geq n_2$, $f_2(n) \leq C_2 g_2(n)$

let $C_3 = C_1 \times C_2$, $n_3 = \underline{\max(n_1, n_2)} \Rightarrow \forall n \geq n_3$, $f_1(n) \times f_2(n) \leq \underbrace{C_1 \times C_2}_{} g_1(n) g_2(n) = C_3 g_1(n) g_2(n)$

$\Rightarrow \exists C_3>0 \wedge \exists n_3>0$ such $\forall n \geq n_3$, $f_1(n) f_2(n) \leq C_3 g_1(n) g_2(n)$

$\Rightarrow \underline{f_1(n) f_2(n) = O(g_1(n) \cdot g_2(n))}$ #

(c)

$O(\log_2 n) < O(n) < O(\log_2(n) \cdot n) < O(n^2) < O(2^n)$

$\Rightarrow 4 < 2 < 3 < 1 < 5$

(d)

$O(\log_2 n)$

(c) 3. $(a+b)/c*(d-e)$

(a)

$*/+abc-de$ #

(b)

$ab+c/de-*$ #

(c)

read from the tail:

① get 'e' $\Rightarrow$ stack $\boxed{e \mid \; \mid \; \mid \; \mid \; \mid \; \mid \;}$

(d)

② get 'd' $\Rightarrow$ stack $\boxed{e \mid d \mid \; \mid \; \mid \; \mid \; \mid \;}$

③ get '-' $\Rightarrow$ pop out 'd', 'e', perform "d-e"

let resault $= r_1$ $\Rightarrow$ push $r_1$ $\Rightarrow$ stack $\boxed{r_1 \mid \; \mid \; \mid \; \mid \; \mid \; \mid \;}$

④ get 'c' $\Rightarrow$ $\boxed{r_1 \mid c \mid \; \mid \; \mid \; \mid \; \mid \;}$

⑤ get 'b' $\Rightarrow$ $\boxed{r_1 \mid c \mid b \mid \; \mid \; \mid \;}$

⑥ get 'a' $\Rightarrow$ $\boxed{r_1 \mid c \mid b \mid a \mid \; \mid \;}$

⑦ get '+' $\Rightarrow$ pop out 'a', 'b', perform "a+b"

let result $= r_2$ $\Rightarrow$ push $r_2$ $\Rightarrow$ $\boxed{r_1 \mid c \mid r_2 \mid \; \mid \; \mid \;}$

⑧ get '/' $\Rightarrow$ pop out '$r_2$', 'c', perform "$r_2/c$", result $= r_3$ $\Rightarrow$ push $r_3$

$\Rightarrow$ $\boxed{r_1 \mid r_3 \mid \; \mid \; \mid \;}$

⑨ get '*' $\Rightarrow$ pop out '$r_3$', '$r_1$', perform "$r_3 * r_1$"

$\Rightarrow$ let result $= r_4$

At this time we reach the head of the expression, so the result $= r_4$.

4.

(a)

```
pb -> next = pa -> next ;
pa -> next = pb ;
```

(b)

```
pa -> next = pb -> next ;
pb -> next -> prev = pb -> prev ;   // pb->prev == pa
delete pb ;
```

5.

(a)

```
int calcBalanceFactors ( Node *root )   // let the return value be the height
                                        //              of the subtree
{
    If( root == Null ) return 0;
    int _left = calcBalanceFactors ( root -> left ); // _left means the height of left subtree
    int _right= calcBalanceFactors ( root -> right); // _right means the height of right subtree
    root -> bfactor = _left - _right;
    return ( max(_left, _righ) + 1 ) ;
}
```

(b) ~~pre-order = (ABD|E|C F H G I)~~

~~in-order = E D B A H F C G I~~

pre-order: ABDECFHGI

in-order: EDB|A|HF C CGI

⇓ ⇓

ED|B| HF|C|GI

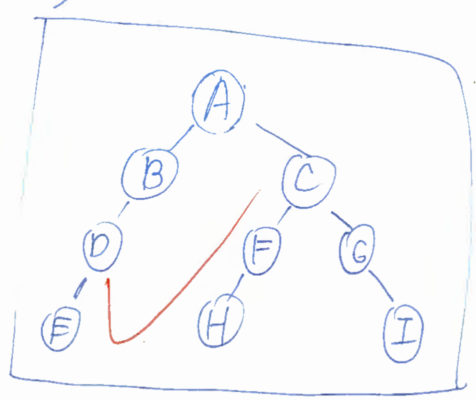
⇓ ⇓ ⇓

E|D| H|F| |G|I

⇓ ⇓ ⇓

E H I

⇒

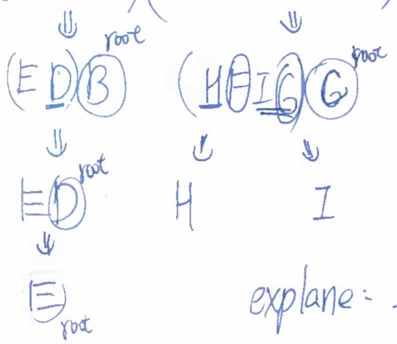



explain: It exists uniquely.

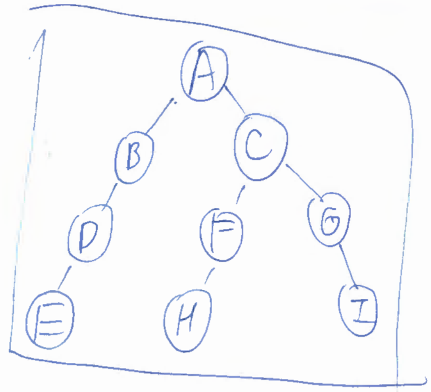
since for each subtree, the first-visited node from pre-order traversal is the root of the subtree, and we can use this fact to tell which nodes are on the right or on the left from in-order traversal. Recursively, we can only build one possible tree.

(c) (c) pre-order = ABDECFHGI

post-order = (ED B)(HFIGC)A




⇒



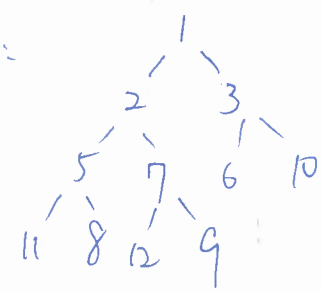
explane: I exists uniquely, because the pre-order traversal helps us to find each root ~~of each subtree, and using this fact on the post-order traversal~~ helps us to distinguish which nodes are on the left or on the right. Recursively, we can only build one possible tree.

+2

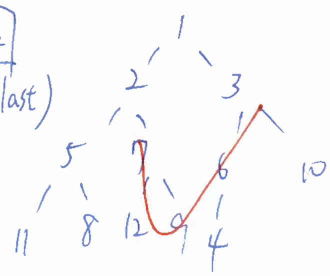
(d)

6.

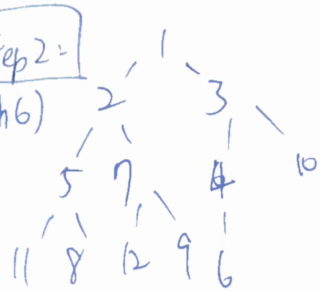
(a) before inserting 4:



$\boxed{\text{Step 1:}}$ (put in the last)



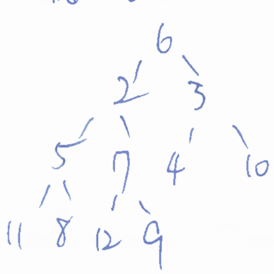
$\boxed{\text{Step 2:}}$ (swap with 6)



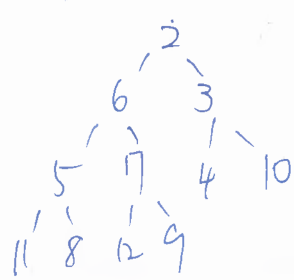
→ since $3<4$, the insertion is done.

(b) $\boxed{\text{Step 1:}}$

(delete 1 and move the last to the top)



$\boxed{\text{Step 2:}}$ (swap with $\min(2,3)=2$)



Step 3 ( swap with min(5,7) = 5 )

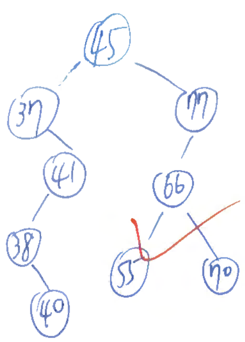

$\Rightarrow$ since $\min(11,8) > 6$, it's done. #

(c)

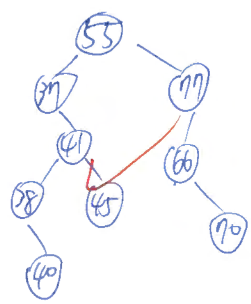
We can find the node with min key in $O(1)$ $\because$ It's the root node.
They, delete the min key from the heap in $O(\log_2 n)$.
Loop this process, since it has $n$ nodes totally, we need to loop $n$ times
Consequently, the time complexity $= O(\log_2 n) \times n = O(n \log_2 n)$ #

7.
(a)

1.

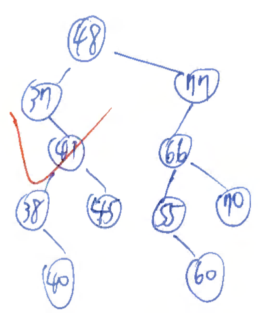



2.

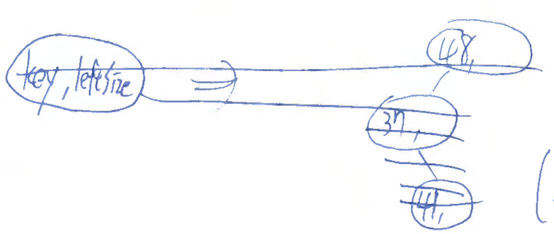



(b)

only one possible :



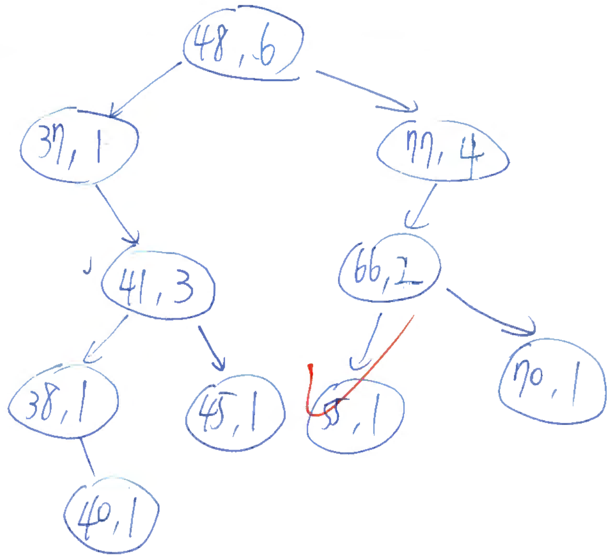

(c)




(換下一頁重寫)

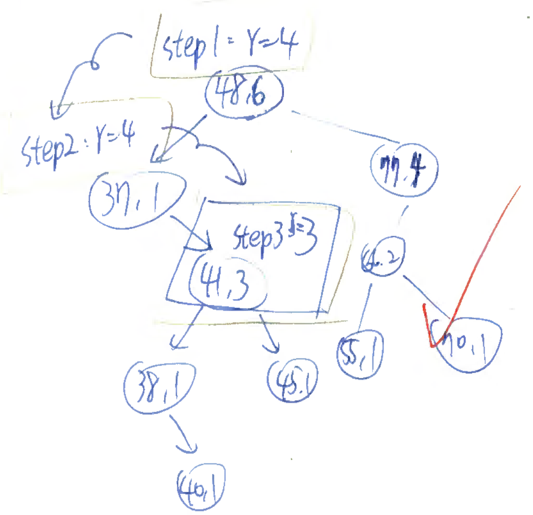
(c)

key, leftSize ⇒




(d)

r=4, firstly, find which node has r=4




⇒ (41,3) is the node with r=4

Then, delete (41,3) (using the smallest node on the right subtree of (41,3))