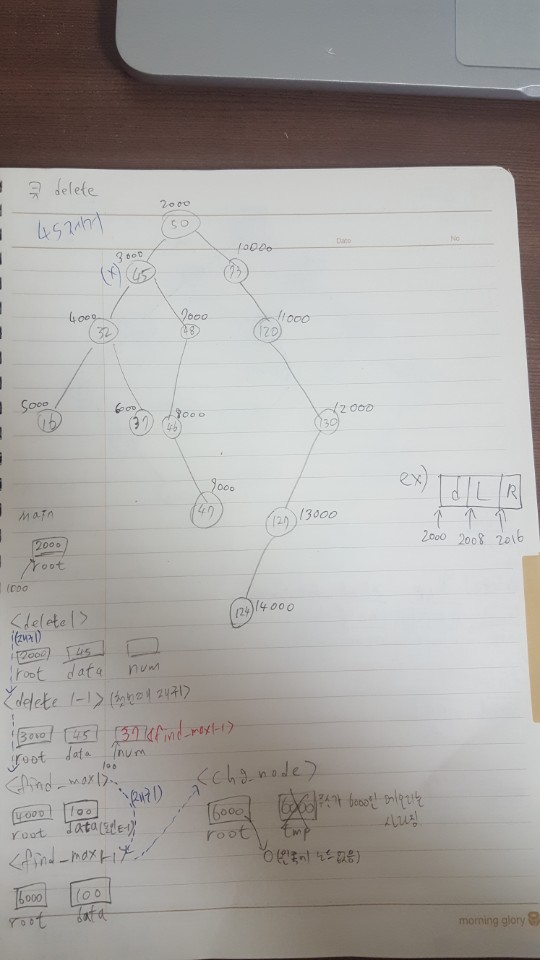
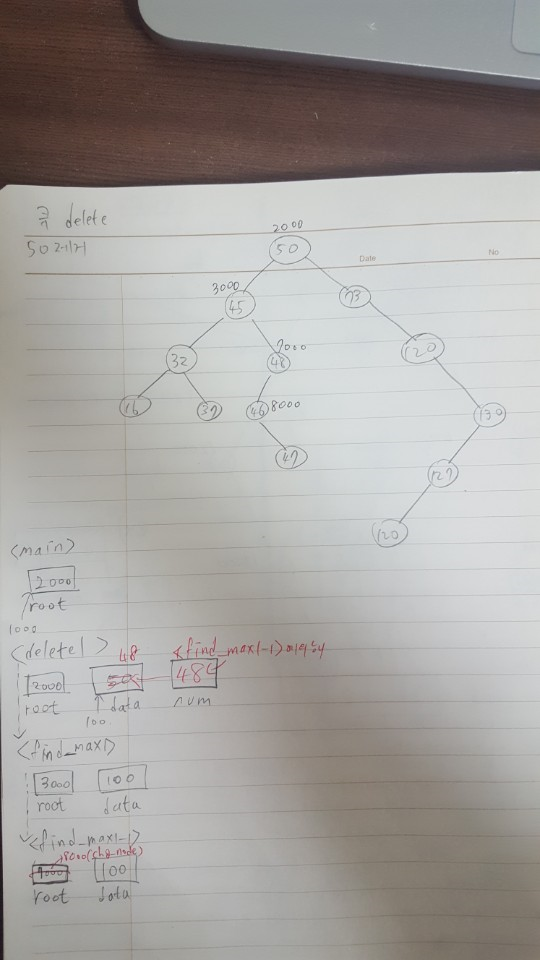
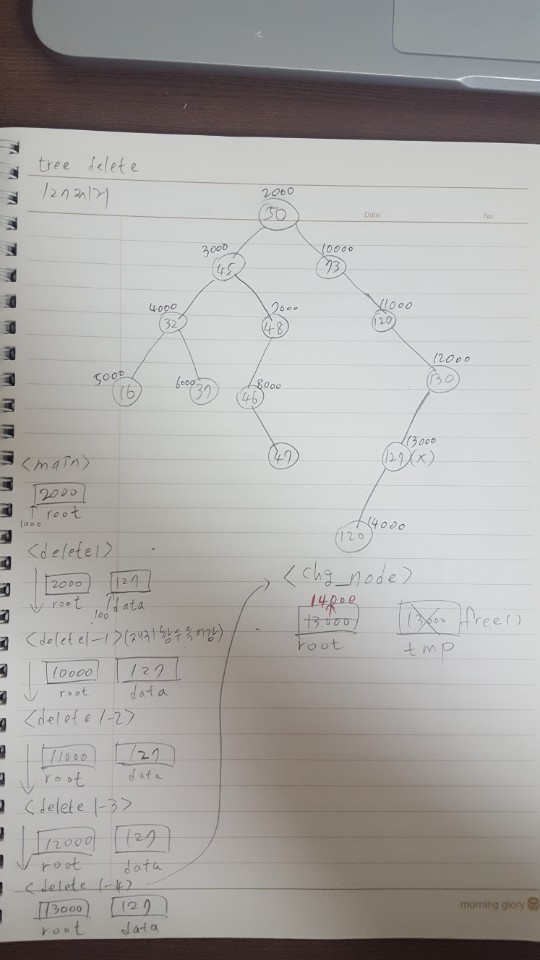
**1) 45제거**



**2) 50제거**



**3) 127제거**



**4) 트리 삭제 코드 설명**

tree\* chg\_node(tree \*root) **// 자식이 한 개만 있거나 없을 때, 현재 노드 지우고, 그 자리에 자식노드를 끌어올리는 함수**

{

tree \*tmp = root; **// root를 지우면 아예 트리가 끊어지므로, tmp라는 포인터에 주소를 옮김. (tmp를 통해서 노드를 없애기 위해)**

if(!root->right) **// find\_max 함수에서 이미 오른쪽 끝까지 갔으므로 무조건 if 로 들어감.**

{

root = root->left; **// 지우려는 노드에 왼쪽 자식 노드를 끌어올림.**

}

else if(!root->left)

{

root = root->right;

}

free(tmp); **// 노드 없앰.**

return root; **// 왼쪽 자식 노드를 리턴함.**

}

tree\* find\_max(tree \*root, int \*data) **// 원래 지우려고 했던 노드에서 왼쪽 서브트리의 데이터 중 최댓값을 찾는 함수.**

{

if(root->right != 0) **// 오른쪽 끝까지 갈때까지 재귀함수로 들어감.**

{

root->right = find\_max(root->right, data);

}

else **// 오른쪽 끝까지 가면 그 노드의 data를 delete\_tree의 num으로 포인터를 통해 보냄.**

{

\*data = root->data;

root = chg\_node(root); **// 맨 오른쪽 노드를 삭제하고 왼쪽 자식 노드를 끌어올림.**

}

return root; **// 재귀함수가 계속 리턴되면, 결국 최종적으로 delete\_tree함수에서의 root->left 노드가 리턴됨.**

}

tree\* delete\_tree(tree \*root, int data)

{

int num;

if(root == 0)

{

printf("Not found\n");

return 0;

}

else if(root->data > data) **// 삭제하려는 data가 해당 노드의 data보다 작으면 왼쪽으로 이동함.**

{

root->left = delete\_tree(root->left, data);

}

else if(root->data < data) **// 삭제하려는 data가 해당 노드의 data보다 크면 오른쪽으로 이동함.**

{

root->right = delete\_tree(root->right, data);

}

else if(root->left && root->right) **// 삭제하려는 data의 노드를 찾았고 해당 노드의 왼쪽,오른쪽 자식 노드 모두 있는 경우.**

{

root->left = find\_max(root->left, &num); **// 삭제하려는 노드의 왼쪽 서브트리에서 최댓값을 찾고, 그 최댓값의 노드를 지우고 왼쪽 자식 노드를 끌어올림.**

root->data = num; **// 왼쪽의 맨 오른쪽 노드의 data를 삭제하려는 노드의 data로 옮김. (이렇게 함으로써**

**실질적으로 노드를 삭제하지 않아도 마치 data가 삭제한 것처럼 보임.)**

}

else // 한쪽만 있거나 모두 없는 경우

{

root = chg\_node(root); **// 해당 노드를 삭제하고 자식 노드를 끌어올림.**

}

return root; **//** **위의 재귀함수에서 현재의 노드 주소가 역순으로 계속 리턴됨.**

}