13.1 탐색이란?

탐색 여러 개의 자료 중에서 원하는 자료를 찾는 작업

항목 탐색의 단위

ex) 숫자, 구조체

키 항목과 항목을 구별시켜줌

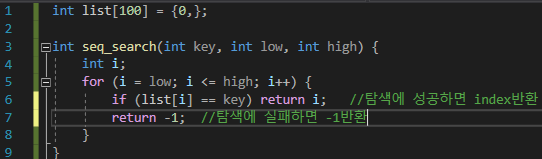
= 탐색키

탐색 = 탐색키와 데이터로 이루어진 여러 개의 항목 중에서 원하는 탐색키를 가지고 있는 항목을 찾는 것

13.2 정렬되지 않은 배열에서의 탐색

순차탐색 탐색방법 중에서 가장 간단, 직접적인 방법

정렬되지 않은 할목들을 처음부터 마지막까지 하나씩 검사

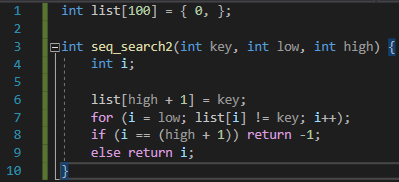
프로그램13.1 

개선된 순차탐색 순차 탐색에서 비교횟수를 줄이는 방법

2번의 비교연산 리스트의 끝을 테스트 🡪 없애기

키값의 비교연산

반복문의 탈출조건을 키 값을 찾을 때로 변경

프로그램13.2 

순차탐색의 시간복잡도 리스트의 처음부터 탐색 시작 🡪 해당항목을 찾거나 모든 항목을 검색할 때까지 항목의 키값 비교

시간복잡도 O(n)

13.3 정렬된 배열에서의 탐색

정렬된 배열에서의 이진탐색 배열의 중앙에 있는 값을 조사 🡪 찾고자 하는 항목이 왼쪽, 오른쪽에 있는지 확인 🡪 범위를 반으로 줄임

비교가 이루어질 때마다 탐색범위가 급격히 줄어듬

이진탐색 진행 시, 배열이 반드시 정렬되어 있어야 함

고정된 데이터 탐색에 적합

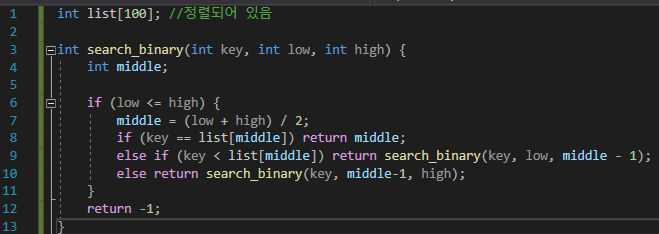
알고리즘13.1 search\_binary(list, low, high) :

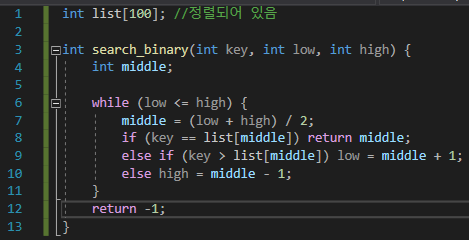
middle = low와 high의 중간값

if (탐색값 = list[middle]) return middle;

else if (탐색값 < list[middle]) return list[0]부터 list[middle-1]까지 탐색

else if (탐색값 > list[middle]) return list[middle+1]부터 list[high]까지 탐색

이진탐색구현(순환호출버전) 프로그램13.3 

이진탐색구현(반복적인 버전) 프로그램13.4 

이진탐색의 시간복잡도 O(log2n)

정렬된 배열에서의 색인순차탐색 파일처리, 데이터베이스 등에서 많이 쓰는 탐색

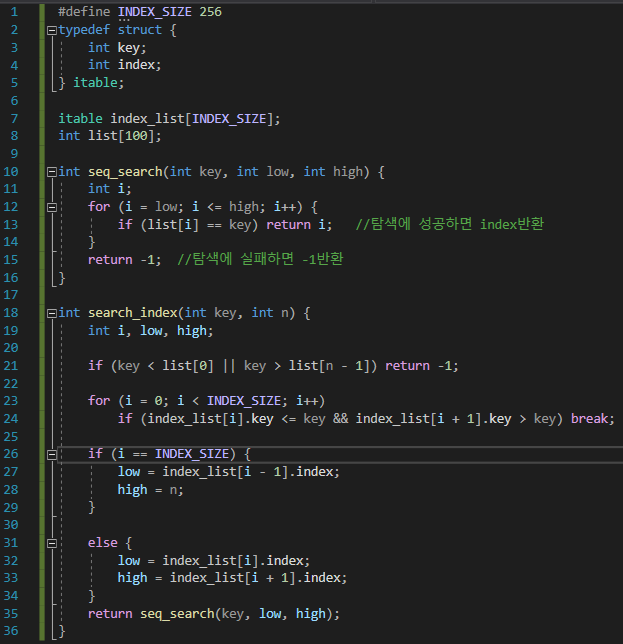
색인순차탐색 인덱스 테이블을 이용해 탐색의 효율을 높임

인덱스 테이블 주 자료 리스트에서 일정 간격으로 발췌한 자료

인덱스테이블에 m개 항목, 주 자료 리스트에 n개 항목 🡪 인덱스 항목은 주자료리스트의 n/m번째 데이터를 가지고 있음

정렬 필요

알고리즘 index[i] < key < index[i+1]을 만족하는 항목부터 주 자료 리스트에서 순차탐색

프로그램13.5 

시간복잡도 인덱스테이블의 크기 m, 주자료리스트의 크기 n 🡪 O(m+n/m)

보간탐색 탐색키가 존재할 위치를 예측해 탐색하는 방법

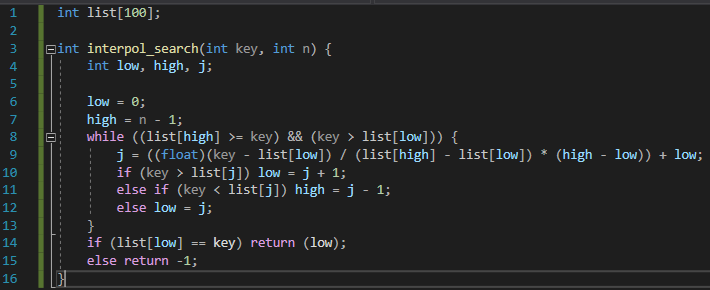
이진탐색과 유사 리스트를 반으로 분할하지 않고 불균등하게 분할하여 탐색

보간탐색의 분할위치 k 찾고자하는 키 값

low 탐색할 범위의 최소 인덱스 값

high 탐색할 범위의 최대 인덱스 값

(k - list[low]) / (list[high] - list[low]) \* (high - low) + low; //소수점이하는 버림

프로그램13.6 

Quiz 1. 리스트변화 1 2 3 5 6 8 9 10 🡪 1 2 3 5 6 8 9 10 4

비교순서 1 🡪 2 🡪 3 🡪 5 🡪 6 🡪 8 🡪 9 🡪 10 🡪 4

2. 리스트변화 1 2 3 4 6 7 8 9🡪 6 7 8 9

비교순서 4 🡪 7 🡪 6

3. 리스트변화 1 2 4 5 7 8 9 10 🡪 1 2 3 5 7 8 🡪 7 8

비교순서 9 🡪 5

13.4 이진탐색트리

이진탐색 자료의 삽입과 삭제가 힘듬

이진탐색트리 비교적 빠른 삽입과 삭제

균형을 유지해야 함 = 높이를 최대한 작게 해야함

13.5 AVL트리

AVL트리 각 노드에서 왼쪽 서브트리의 높이와 오른쪽 서브트리의 높이 차이가 1 이하인 이진탐색트리

비균형 상태가 되면 스스로 노드들을 재배치

AVL트리의 탐색연산 일반적인 이진탐색트리와 동일

시간복잡도 O(log2n)

AVL트리의 삽입연산 삽입, 삭제연산 시 트리의 균형상태가 깨짐

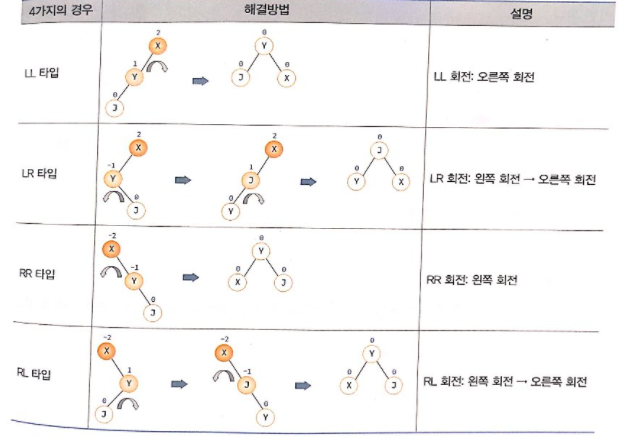
깨진균형 삽입연산 시, 삽입되는 위치부터 루트까지 경로의 노드들의 균형 인수에 영향을 줌

불균형상태로 변한 가장 가까운 조상노드의 서브트리에 대해 다시 균형을 잡아야 함

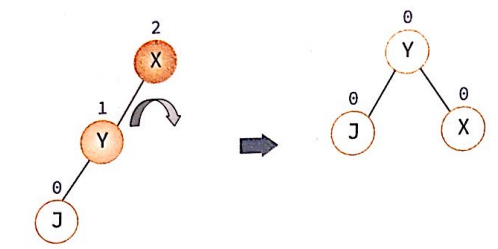
균형잡기 새로운노드부터 균형이 깨진 조상노드까지의 트리를 회전

4가지의 경우 J 새롭게 삽입된 노드

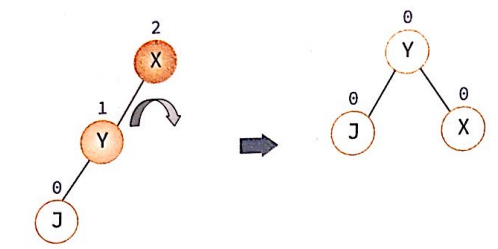
X 균형인수가 ±2가 된 조상노드



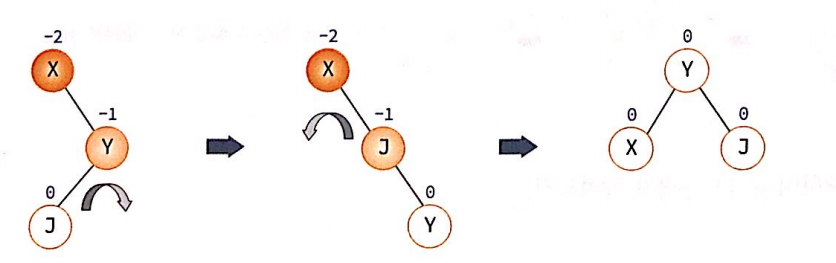
LL타입 노드Y의 왼쪽 자식으로 노드가 추가됨으로서 발생 🡪 노드들을 오른쪽으로 회전시키면 됨



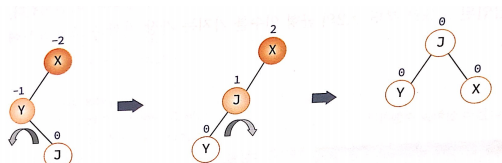
RR타입 노드Y의 오른쪽 자식으로 노드가 추가됨으로써 발생 🡪 노드들을 왼쪽으로 회전시키면 됨



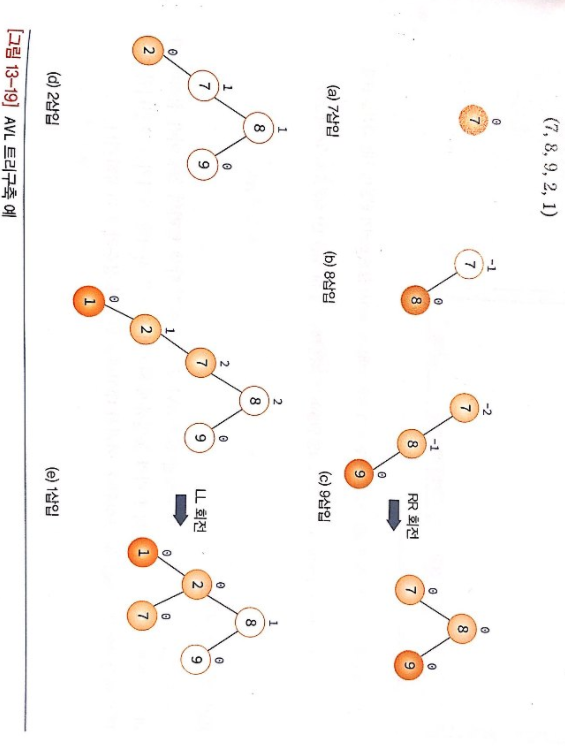
RL타입 노드 X의 오른쪽 자식의 왼쪽에 노드가 추가됨으로써 발생 🡪 2번의 회전필요



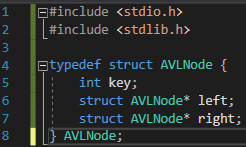
LR타입 노드 X의 왼쪽자식의 왼쪽에 노드가 추가됨으로써 발생 🡪 2번의 회전 필요

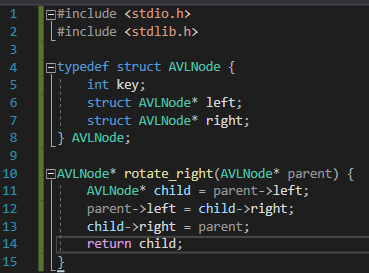


AVL트리 예제 데이터가 삽입되고 균형이 깨지면 회전연산 실행

ex)

AVL트리 정의 AVL트리 = 이진트리의 일종

프로그램13.7 

rotate\_right()함수 구현 프로그램 13.8 

rotate\_left()함수 구현 AVLNode\* rotate\_left(AVLNode\* parent) {

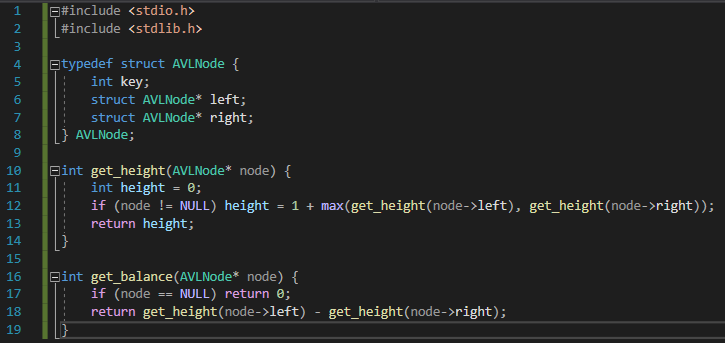
AVLNode\* child = parent->right;

parent->right = child->left;

child->left = parent;

return child;

}

트리의 높이 계산 프로그램 13.9 

새로운 노드 추가 함수 AVLNode\* insert(AVLNode\* node, int key) {

if (node == NULL) return (create\_node(key));

if (key < node->key) node->left = insert(node->left, key);

else if (key < node->key) node->right = insert(node->right, key);

else return node;

int balance = get\_balance(node);

if (balance > 1 && key < node->left->key) return rotate\_right(node);

if (balance < -1 && key > node->right->key) return rotate\_left(node);

if (balance > 1 && key > node->left->key) {

node->left = rotate\_left(node->left);

return rotate\_right(node);

}

if (balance < -1 && key < node->right->key) {

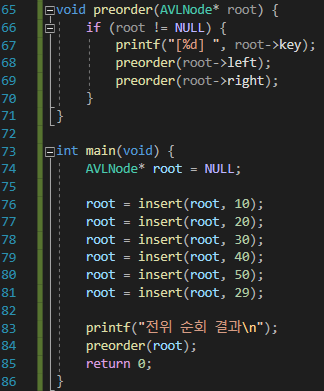
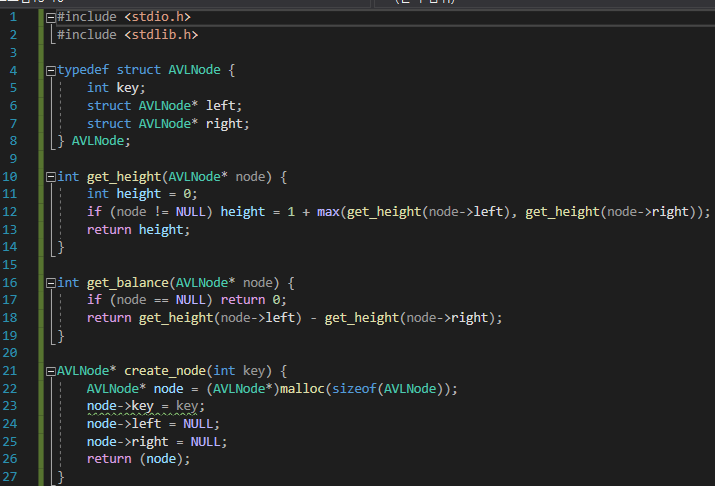
node->left = rotate\_right(node->right);

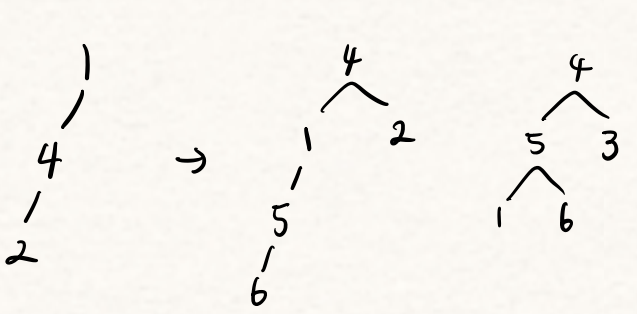
return rotate\_right(node);

}

return node;

}

전체 프로그램 프로그램13.10 

Quiz 1. 

13.6 2-3트리

차수가 2또는 3인 트리

삽입, 삭제 알고리즘이 AVL트리보다 더 간단

2-노드 1개의 데이터와 2개의 자식노드를 가짐

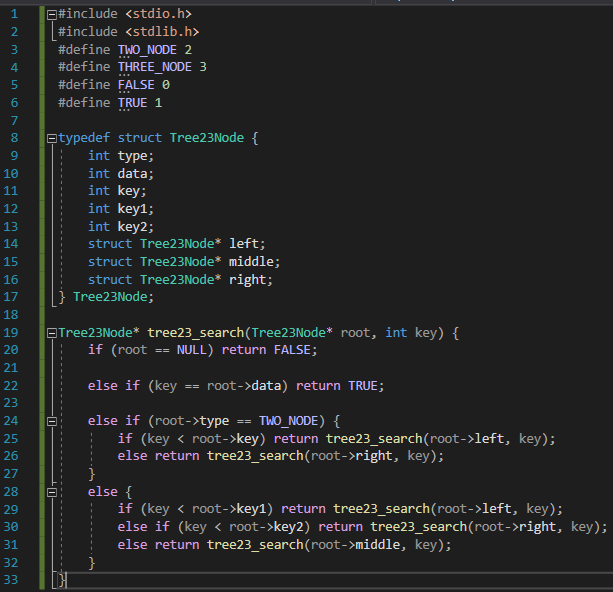
3-노드 2개의 데이터와 3개의 자식노드를 가짐

데이터 k1, k2 왼쪽 서브트리 < k1

k1 < 가운데 서브트리 < k2

오른쪽 서브트리 > k2

2-3트리의 탐색 연산 2-노드, 3-노드에 따라 탐색 진행

프로그램13.11 

2-3트리의 삽입 연산 2-3트리의 노드는 2개의 데이터값을 저장할 수 있음

더 이상 노드안에 데이터값을 저장할 수 없는 경우, 노드를 분리함

노드 분리 중간값을 1단계위로 올림

제일 작은 값을 왼쪽으로 내림

제일 큰 값을 오른쪽으로 올림

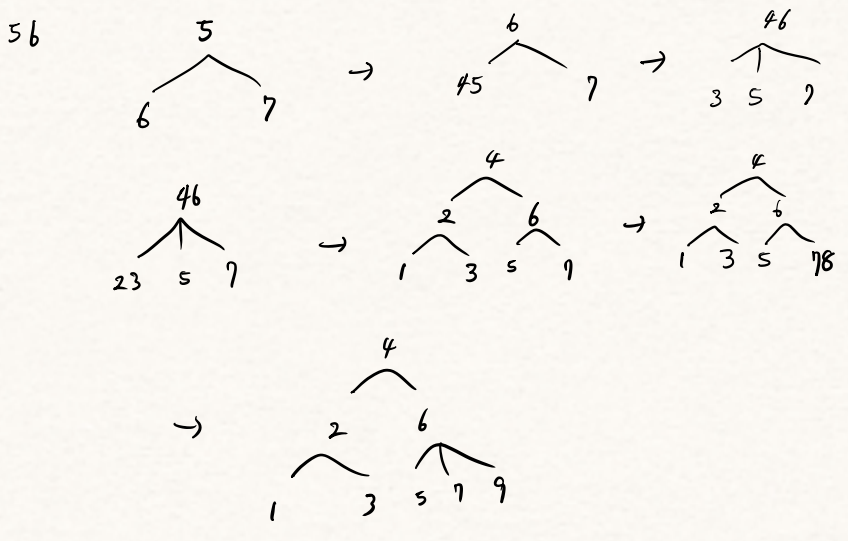
단말노드를 분리하는 경우 단말노드의 중간값은 부모노드로 올라감

작은값, 큰값은 새로운 노드로 분리됨

비단말노드(루트노드)를 분리하는 경우 단말노드의 중간값은 부모노드(윗단계)로 올라감

작은값, 큰값은 새로운 노드로 분리됨

작은값, 큰값에 딸린 트리들도 분리됨

Quiz 1. 

13.7 2-3-4트리

노드가 4개의 자식까지 가질 수 있도록 2-3트리 확장

4-노드 4개의 자식, 3개의 데이터

데이터 small, middle, large

자식 T1 < small

small < T2 < middle

middle < T3 < large

T4 > large

2-3-4트리의 삽입연산 삽입할 노드가 2-노드, 3-노드면 동일하게 진행

but 4-노드면 후진분할 연산이 필요함 🡪 후진분할 연산을 피하기위해 삽입노드를 찾는 순회 시에 4-노드를 만나면 미리 분할 수행 🡪 삽입되는 4-노드의 부모는 무조건 4-노드가 아님

3가지 경우 1. 4-노드가 루트인 경우

2. 4-노드의 부모가 2-노드인 경우

3. 4-노드의 부모가 3-노드인 경우

연습문제

1. 3

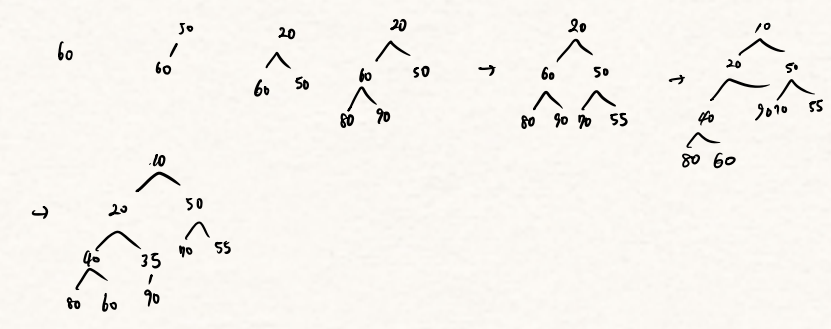
2. (1) 8 🡪 11 🡪 12 🡪 15 🡪 16 🡪 19 🡪 20 🡪 23 🡪 25 🡪 28, 10번

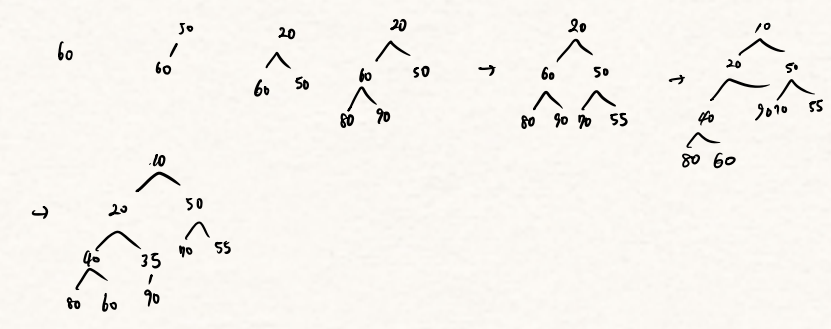
(2) 23 🡪 31 🡪 28, 3번

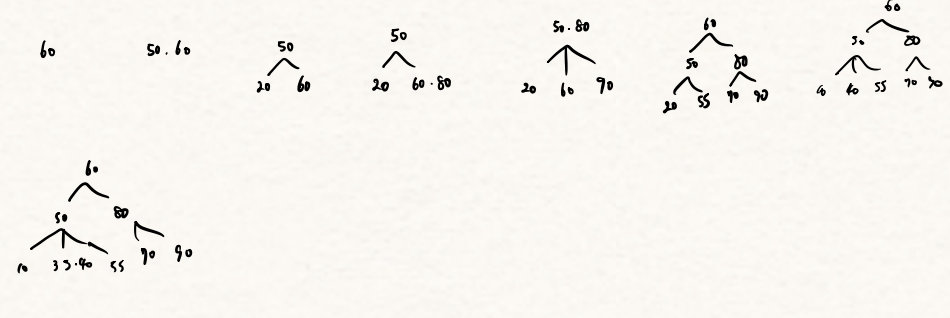
(3) 28, 1번

3. 2

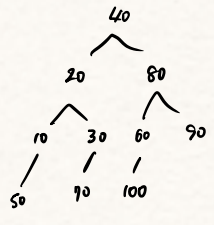
4. 100000000개

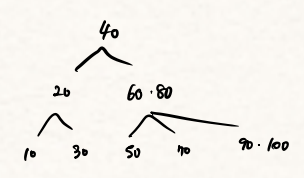
5. (1) 

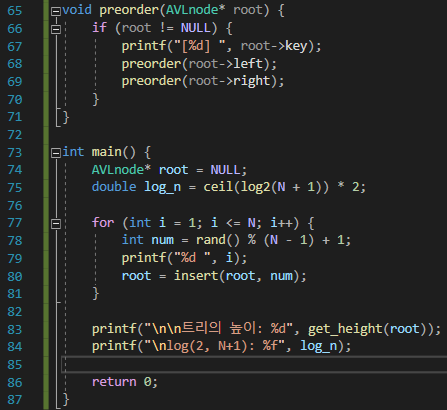
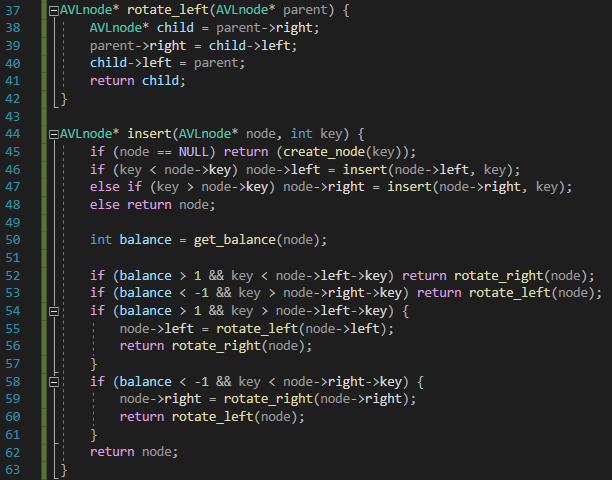
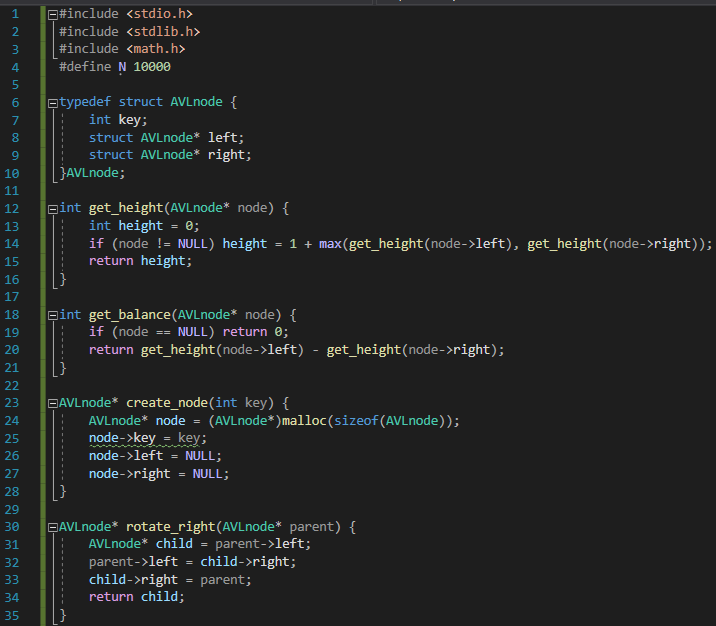
(2) 

(3) 

6. (1) 

(2) 

(3) 

7. 

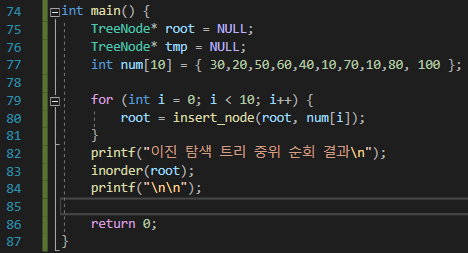
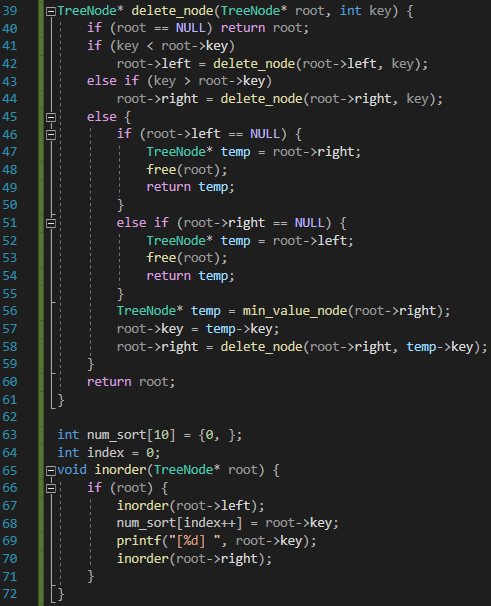
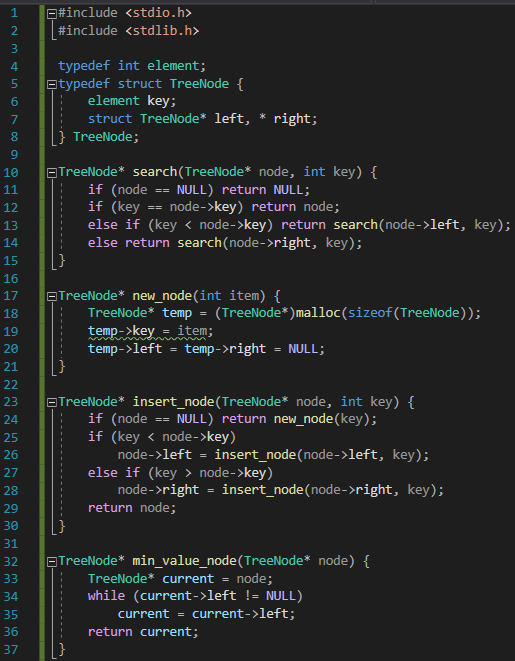
n=100, 높이 7, log값은 14 7

n=500, 높이 9, log값은 18 10

n=1000, 높이 10, log값은 20 11

n=10000, 높이 14, log값은 28 15

n=50000, 높이 16, log값은 18

8. 

이진탐색 시간복잡도 O(log2n)

삽입정렬 시간복잡도 O(n)

히프정렬 시간복잡도 O(nlog2n)

9. 