실습과제 7.2

[과제 설명]

최대 힙 트리(Max Heap Tree)는 완전 이진 트리로 구성된 트리 종류 중 하나로, 부모 노드의 키 값은 자식 노드의 키 값과 크거나 같다. 단, 좌우 자식 노드의 키 값은 크기를 구분하지 않는다.

이런 최대 힙 트리에서 새로운 키 값을 삽입하려면, 마지막에 노드를 추가한 뒤, 부모 노드와 비교해서 더이상 자기보다 작은 부모 노드가 없을 때까지 계속 Swap(교환)해주면 된다.

최대 힙 트리에서 기존 노드를 삭제하는 방법은, 최상단 노드를 삭제하고 나서 마지막 노드를 삭제했던 노드의 자리에 넣는다. 그러고 나서 삽입 때처럼 부모 노드와 비교해서 더이상 자기보다 작은 자식 노드가 없을 때까지 계속 Swap해주면 된다.

최대 힙 트리 함수를 만들 때, 알고리즘과 앞서 나온 개념에 충실한다면 Swap을 사용해서 만들어야 하지만, Swap은 생각보다 리소스를 많이 사용하게 된다. 따라서 삽입/삭제 함수는 Swap을 사용하지 않고 구현할 수 있다.

아래의 코드들은 Swap을 사용하지 않는 방법 2를 이용해서 구현했다.

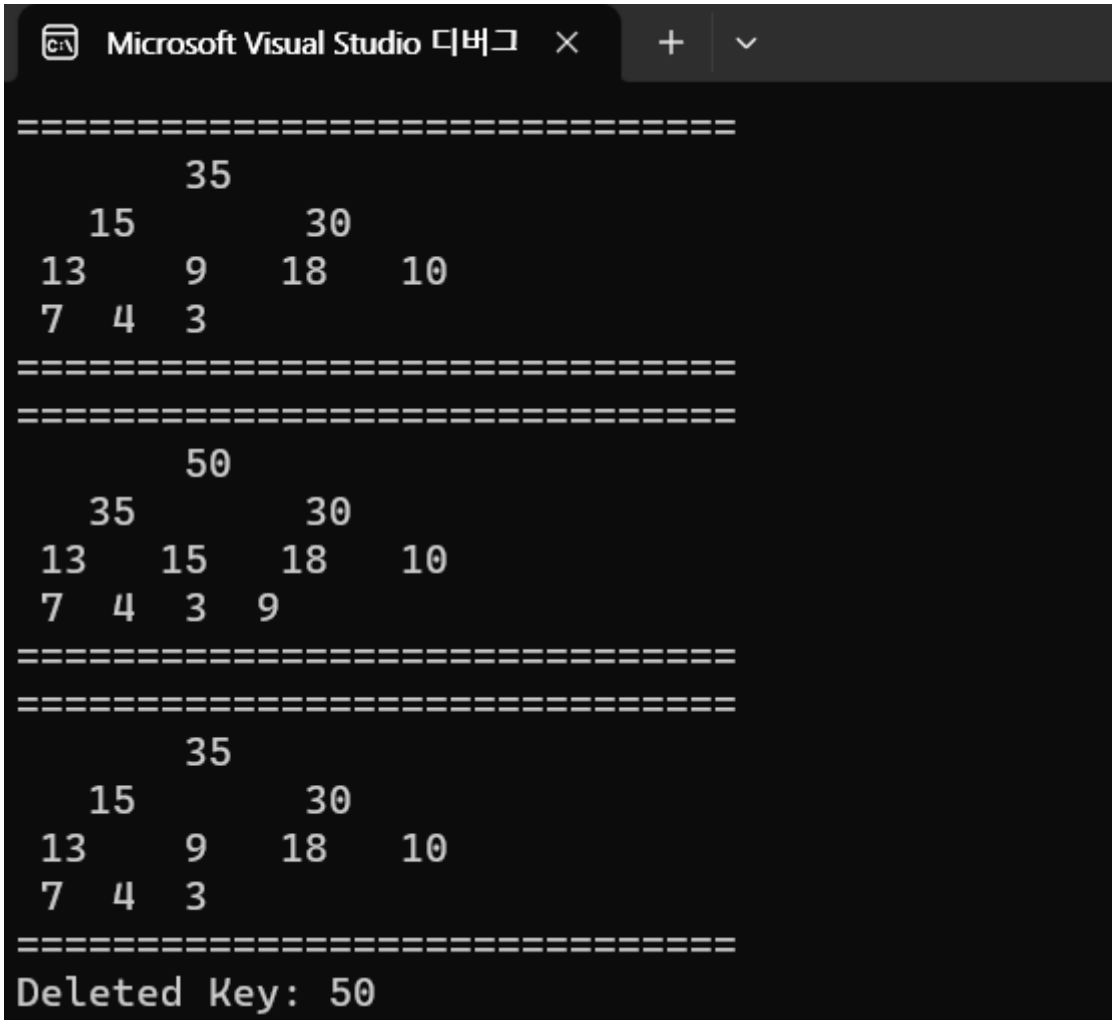
[코드]

메인 함수와 header 파일, 출력 함수는 교안에서 미리 구현된 것을 사용했으므로 보고서에는 생략하고 구현된 최대 힙 트리 삽입/삭제 함수 세부 내용만 첨부한다.

```
//최대 힙 트리 삽입 함수
//방법 2: Swap에 사용되는 리소스 절약
void Max_Heap_Insert(int* heap, int* h_size, int key) {
  *h_size = *h_size + 1;
  int idx = *h_size;
  //포인터로 받았기 때문에 대입, 연산을 할 때 *를 붙여야 한다.

  while ((idx != 1)) {
    if (key > heap[idx / 2]) { //조상과 비교해서, 조상보다 key값이 크다면,
      heap[idx] = heap[idx / 2]; //조상의 값을 아래로 내림
      idx = idx / 2; //조상으로 이동
    }
    else
```

```
      break;
  }
  heap[idx] = key; //최종적으로 찾은 index에 key 값을 삽입
}

//최대 힙 트리 삭제 함수
//방법 2: Swap에 사용되는 리소스 절약
int Max_Heap_Remove(int* heap, int* h_size) {
  int deleted_key = heap[1]; //삭제할 값(root node의 key값)을 저장
  int temp = heap[*h_size]; //마지막 노드를 임시 변수에 저장
  *h_size = *h_size - 1;
  int parent = 1;
  int idx = 2;

  while (idx <= *h_size) {
    if ((idx < *h_size) && (heap[idx + 1] > heap[idx]))
      //오른쪽 자식이 더 크다면
      idx = idx + 1;
    if (temp >= heap[idx]) //temp의 값(마지막 노드 값)이 크다면 반복 종료
      break;

    heap[parent] = heap[idx]; //더 큰 자식을 부모로
    parent = idx; //부모(parent) 값을 업데이트
    idx = idx * 2; //자식으로 이동
  }

  heap[parent] = temp;
  return deleted_key; //삭제가 완료된 key 값을 반환
}
```

[실행 결과]



[과제에 대한 고찰]

코드를 테스트 하면서 삽입 노드에 루트 노드에 있는 값보다 큰 값 대신, 전체 트리의 중간 쯤 위치할 데이터를 넣어주어도 함수가 잘 작동이 되었다. 이렇게 최상단의 값이 아닌데도 정상 작동하는 이유는, 마지막에 추가해서 비교하다가 중간에 조건이 false가 되면 별 문제 없이 반복문이 종료되고 함수가 종료되기 때문일 것이다.

반면에 삭제 함수는 노드 하나를 삭제 후, 마지막 노드를 맨 위에 올려주는 것을 시작으로 한다. 중간에 있는 노드를 삭제해버리면, 그 노드 전에 있는 노드들의 인덱스를 하나씩 다 밀어주는 코드가 없기 때문에 제대로 실행되지 않는다. 또, 함수 인수로 특정 노드를 받는 형태가 아니라 무조건 최상단 루트 노드만 삭제할 수 있게 알고리즘이 만들어져 있다. 이것을 개선하는 방법에는 뭐가 있을지 고민해봐야겠다는 생각이 들었다.