Homework #1

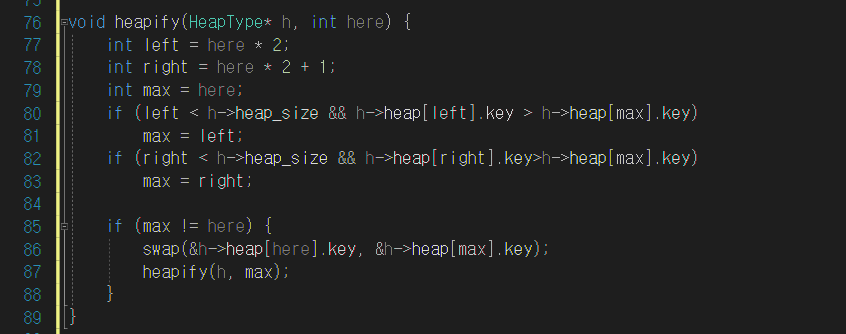
**Code Explanation**

1. void swap(int\* a, int\* b)



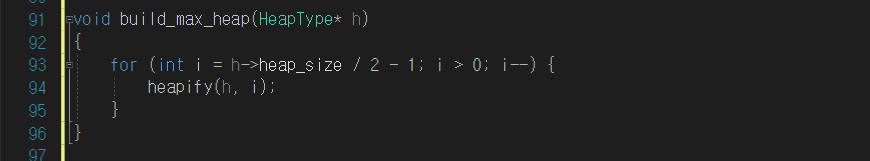
heap에서 자리를 바꿀 때 사용할 swap 함수를 정의하였다.

1. void heapify(HeapType\* h, int here)



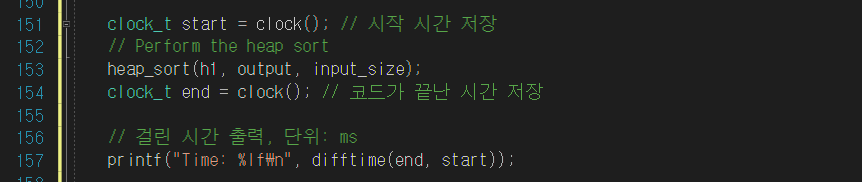
heap의 부모 노드와 자식 노드를 비교했을 때 자식 노드가 크면 위치를 바꾸고 그렇지 않으면 그대로 유지한다.

1. void build\_max\_heap(HeapType\* h)

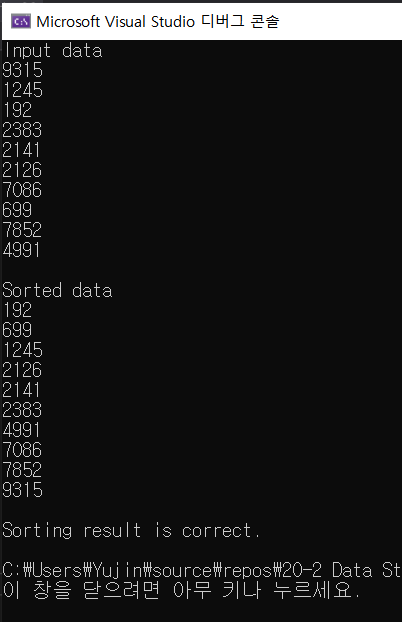


위에서 정의한 Heapify 함수를 n/2-1 에서 1까지 반복하여 maxheap을 만든다.

1. Heap\_sort의 실행시간을 출력하는 함수

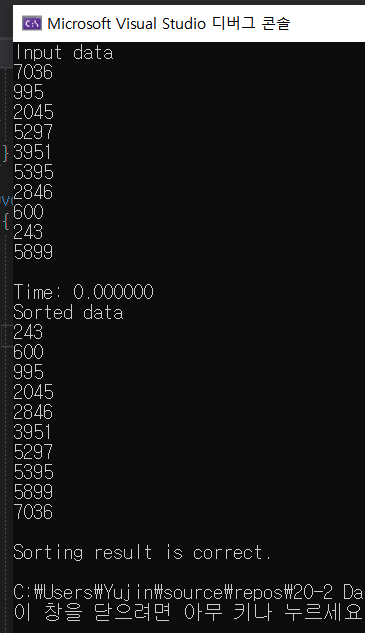


**실행결과**

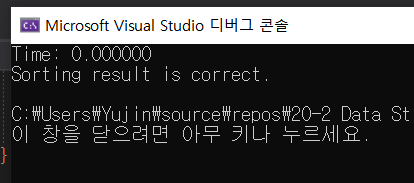


**실행시간 비교**

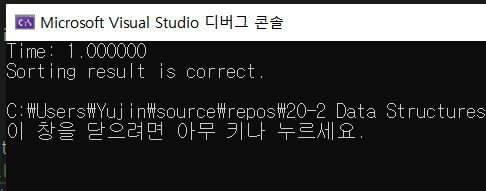
1. int input\_size = 10;



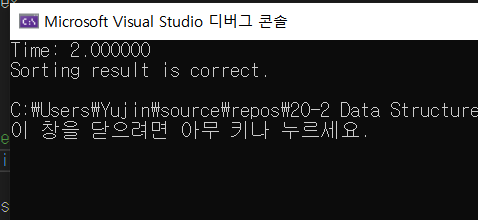
1. int input\_size = 100;



1. int input\_size = 1000;



1. int input\_size = 5000;



Homework #2

**Code Explanation**

1. void huffman\_traversal(TreeNode\* node,int level)

다음 두가지 경우로 나누어 생각해볼 수 있다.

1. Leafnode일 경우 : 현재까지 arr배열에 저장된 binary code를 M\_LUT 테이블에 옮겨담는다.
2. Leafnode가 아닐 경우
   1. 왼쪽 노드 탐색 : arr배열에 0을 저장하고, 재귀함수를 호출한다.
   2. 오른쪽 노드 탐색 : arr배열에 1을 저장하고, 재귀함수를 호출한다.

이후 huffman\_traversal 함수를 통해서 m\_LUT와 m\_bit\_size에 알맞은 값이 저장되면, 이를 이용하여 encoding과 decoding을 진행할 수 있다.

1. void huffman\_encoding (char\* str, bits\_stream\* bits\_str)

: string->bit로 만드는 함수. 각 char에 맞는 bit를 stream에 저장하고 해당 bit\_size를 length에 저장한다.

1. int huffman\_decoding (bits\_stream\* bits\_str, TreeNode\* node, char\* decoded\_str)

: encoding된 string을 decode하는 함수. 허프만 트리에 따라서 해당 비트의 자식 노드로 이동하여 마지막 노드일 경우 해당 data를 output에 저장하고 새로운 search를 다시 시작한다.

**실행결과**

