**컴퓨터 알고리즘과 실습**

* 3회차 실습 -

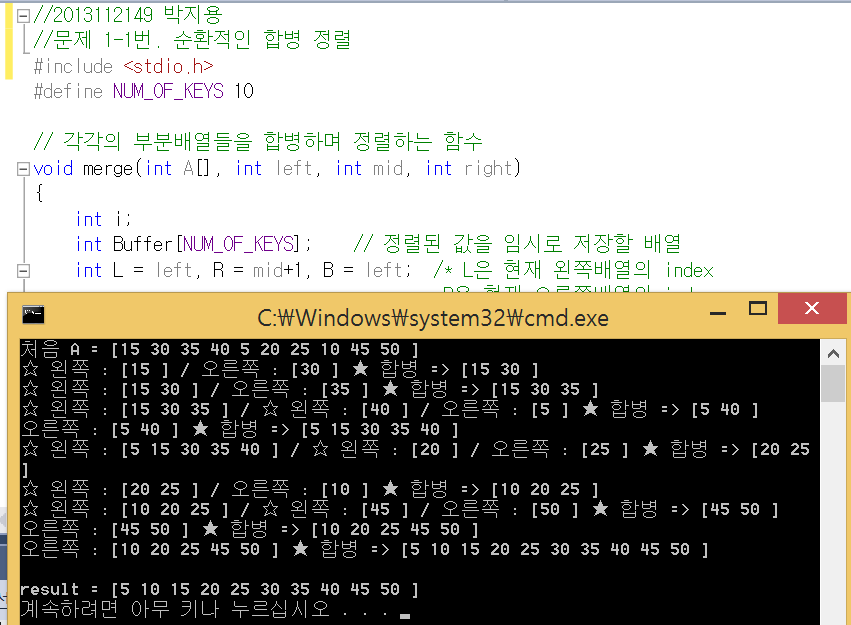
2017.03.22

컴퓨터공학과

2013112149 박지용

* **실습문제 1번 – (1). 순환적 합병 정렬 구현하기**

<출력 결과>



순환적인 합병정렬을 구현한 1-1번 출력화면입니다. 이 문제에 대한 출력을 어떻게 해야하나를 많이 고민했는데 나름대로 알아보기 쉽게 출력한다고 하였으나 너무 복잡하게 되었습니다.

배열을 왼쪽과 오른쪽으로 나눠서 왼쪽 부분배열과 오른쪽 부분배열이 합병되는 과정을 표현하였는데, 이 단위는 흰색 별(☆)과 검은색 별(★)로 묶었습니다. 예를들어 화면의 6번째 줄의 첫부분에 있는 ☆은 11번째 줄의 마지막 ★로 닫히게 됩니다. 그 중간에 있는 과정은 그 내부 과정이라고 보시면 됩니다. 이런 식으로 합병을 진행해서 결과적으로 오름차순으로 올바르게 정렬된 배열(result)을 만들었습니다. 소스코드에 대한 설명은 아래쪽에 있습니다.

<1-1번 소스코드>

//2013112149 박지용

//문제 1-1번. 순환적인 합병 정렬

#include <stdio.h>

#define NUM\_OF\_KEYS 10

// 각각의 부분배열들을 합병하며 정렬하는 함수

void merge(int A[], int left, int mid, int right)

{

int i;

int Buffer[NUM\_OF\_KEYS]; // 정렬된 값을 임시로 저장할 배열

int L = left, R = mid+1, B = left; /\* L은 현재 왼쪽배열의 index

R은 현재 오른쪽배열의 index

B는 현재 Buffer배열의 index

\*/

while (L <= mid && R <= right)

{

if (A[L] < A[R])

Buffer[B++] = A[L++]; // 오른쪽과 왼쪽 중 작은 값을 버퍼에 넣어줌

else

Buffer[B++] = A[R++];

}

if (L > mid) // 왼쪽 배열의 원소가 다 버퍼로 들어갔으면

{

for (i = R; i <= right; i++)

Buffer[B++] = A[i]; // 오른쪽 배열을 다 넣어줌

}

else // 왼쪽 배열의 원소가 다 버퍼로 들어갔으면

{

for (i = L; i <= mid; i++)

Buffer[B++] = A[i]; // 왼쪽 배열을 다 넣어줌

}

for (i = left; i <= right; i++)

A[i] = Buffer[i]; //정렬이 끝난 배열을 A에 옮겨줌

}

//left부터 right까지의 배열 원소를 합병정렬하는 함수

void merge\_sort(int A[], int left, int right)

{

int mid, i;

if (left < right)

{

mid = (left + right) / 2;

merge\_sort(A, left, mid); // 왼쪽 배열 합병정렬

printf("☆ 왼쪽 : [");

for (i = left; i <= mid; i++)

printf("%d ", A[i]);

printf("] / ");

merge\_sort(A, mid + 1, right); // 오른쪽 배열 합병정렬

printf("오른쪽 : [");

for (i = mid + 1; i <= right; i++)

printf("%d ", A[i]);

printf("] ★ ");

merge(A, left, mid, right);

printf("합병 => [");

for (i = left; i <= right; i++)

printf("%d ", A[i]);

printf("]\n");

}

}

int main(void)

{

int i;

int A[] = { 15, 30, 35, 40, 5, 20, 25, 10, 45, 50 };

int left = 0, right = NUM\_OF\_KEYS - 1;

printf("처음 A = [");

for (i = left; i <= right; i++) // 정렬하기 전 배열 출력부분

printf("%d ", A[i]);

printf("]\n");

merge\_sort(A, left, right);

printf("\nresult = [");

for (i = left; i <= right; i++) // 정렬 후 배열 출력부분

printf("%d ", A[i]);

printf("]\n");

return 0;

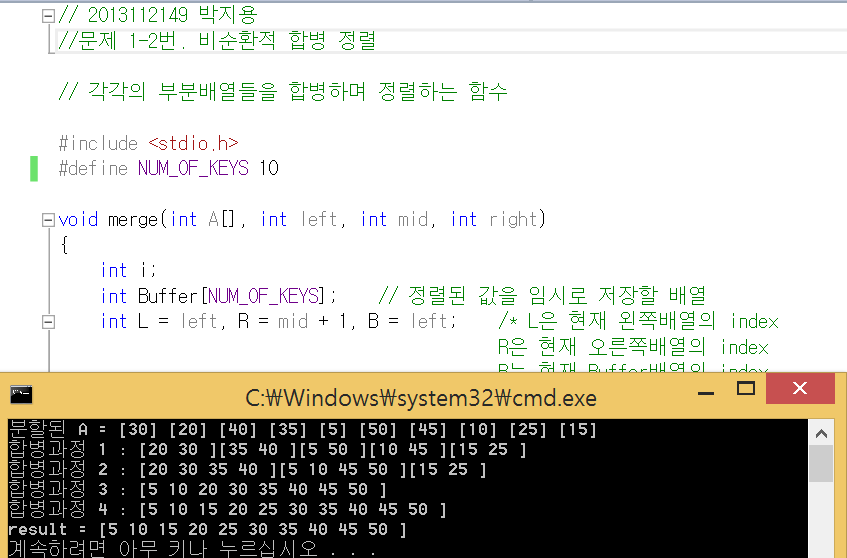
}

<소스코드 설명>

merge\_sort함수를 보면 출력부분은 제외하고 left~mid까지 merge를 한번, mid+1부터 right까지 merge를 한번 하고 merge함수를 호출합니다.. 이는 원래 배열을 반복적으로 왼쪽과 오른쪽 부분배열로 쪼갠 다음 둘이 합병하는 과정에서 대소관계를 비교하여 정렬하겠다는 것입니다. 대소관계 비교 및 합병을 하기위해서는 merge함수가 필요한데, merge함수의 기본 아이디어는 각각의 단계마다 배열을 저장할 Buffer 배열을 둬서 합병을 한 결과를 Buffer 배열에 옮기겠다는 것입니다. 처음에 L과 R, B라는, 각각 left, right, buffer 배열의 index를 나타내는 변수들을 둬서 대소비교를 해가면서 더 작은 값을 Buffer에 넣어서 정렬해주는데, 한쪽이 먼저 끝날 경우(Buffer로 모두 들어갈 경우) 나머지를 Buffer로 옮겨주는 방식으로 수행합니다. 자세한 설명은 소스코드에 주석으로 달아두었습니다.

* **실습문제 1번 – (2). 비순환적 합병 정렬 구현하기**

<출력 결과>



비순환적 합병 정렬을 구현한 1-2번 문제의 출력화면입니다. 처음 분할된 A배열에서 merge함수를 이용하여 각 원소들이 합병되는 과정을 순차적으로 출력하였습니다. 다만 합병과정 3과 같이 이미 합병이 되어있는 부분([15 25])은 출력하지 않고 합병과정 3 -> 4로 넘어갈 때 3에서 합병되었던 배열이 왼쪽 배열, [15 25]가 오른쪽 배열이 되어 합병됩니다. 알고리즘에 대한 간략한 설명은 아래쪽에 있습니다.

<1-2번 소스코드>

// 2013112149 박지용

//문제 1-2번. 비순환적 합병 정렬

// 각각의 부분배열들을 합병하며 정렬하는 함수

#include <stdio.h>

#define NUM\_OF\_KEYS 10

void merge(int A[], int left, int mid, int right)

{

int i;

int Buffer[NUM\_OF\_KEYS]; // 정렬된 값을 임시로 저장할 배열

int L = left, R = mid + 1, B = left; /\* L은 현재 왼쪽배열의 index

R은 현재 오른쪽배열의 index

B는 현재 Buffer배열의 index

\*/

while (L <= mid && R <= right)

{

if (A[L] < A[R])

Buffer[B++] = A[L++]; // 오른쪽과 왼쪽 중 작은 값을 버퍼에 넣어줌

else

Buffer[B++] = A[R++];

}

if (L > mid) // 왼쪽 배열의 원소가 다 버퍼로 들어갔으면

{

for (i = R; i <= right; i++)

Buffer[B++] = A[i]; // 오른쪽 배열을 다 넣어줌

}

else // 왼쪽 배열의 원소가 다 버퍼로 들어갔으면

{

for (i = L; i <= mid; i++)

Buffer[B++] = A[i]; // 왼쪽 배열을 다 넣어줌

}

printf("[");

for (i = left; i <= right; i++)

{

A[i] = Buffer[i]; //정렬이 끝난 배열을 A에 옮겨줌

printf("%d ", Buffer[i]);

}

printf("]");

}

// 비순환적 합병정렬 - 분할/정복 방식이 아닌 정복 방식

void merge\_sort(int A[], int left, int right)

{

int mid, i, j, n = NUM\_OF\_KEYS;

static int count = 0;

for (i = 1; i < n; i \*= 2)

{

left = 0; // left index는 한 사이클(while)에 1씩 커지므로 결국 n이 될텐데, n이 될때마다 0으로 돌아와야한다.

printf("합병과정 %d : ", ++count);

while (left < n)

{

right = left + 2 \* i - 1; // right는 2\*i단위로 커져나가야 함

if (right >= n)

right = n-1;

mid = left + i - 1; /\* 전체 데이터의 개수(NUM\_OF\_KEYS)가 2의 제곱수일(4, 8, ...) 때는 mid가 (left+right)/2 여도 상관없지만 데이터가 2의 제곱수로 떨어지지 않고 남을 때는 병합을 반복적으로 수행하게 된다. \*/

if (mid < n) // 따라서 mid가 n보다 작을 때만 병합을 해줘야 한다. ((left+right)/2는 절대로 n을 넘을 수가 없기 때문)

merge(A, left, mid, right);

left = right + 1;

}

printf("\n");

}

}

int main(void)

{

int A[] = { 30, 20, 40, 35, 5, 50, 45, 10, 25, 15 };

int right = NUM\_OF\_KEYS-1, left = 0, i;

printf("분할된 A = ");

for (i = 0; i <= right; i++) // 처음 배열 A를 분할한 것 출력

printf("[%d] ", A[i]);

printf("\n");

merge\_sort(A, left, right);

printf("result = [");

for (i = left; i <= right; i++) // 정렬 결과 출력

{

printf("%d ", A[i]);

}

printf("]\n");

return 0;

}

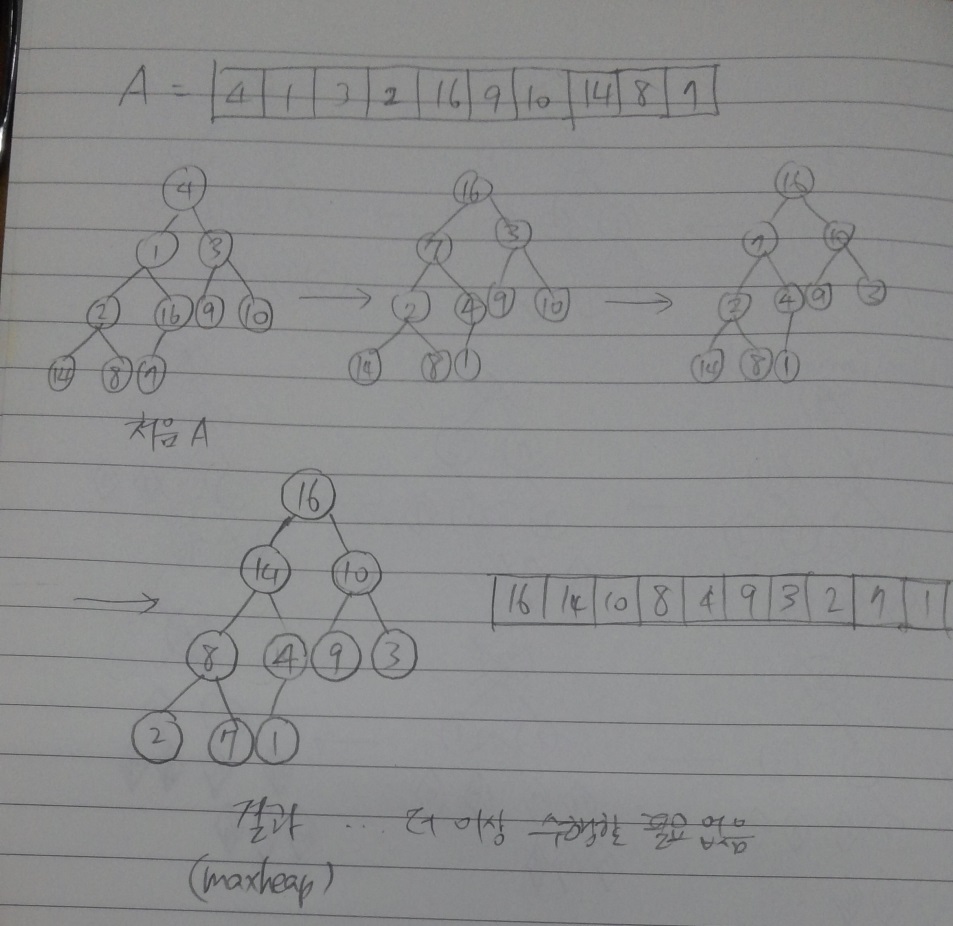
<소스코드 설명>

1-2번 문제에서는 1-1번에서 구현해놓았던 merge함수를 재활용합니다. 어차피 순환 / 비순환의 차이는 merge\_sort함수에서 나타나는 것이지 합병 과정에 대한 아이디어는 동일하기 때문입니다. 따라서 출력부분과 merge함수에 대한 설명은 생략하고 merge\_sort함수를 설명하자면, 주석으로 분할 / 정복방식이 아닌 정복방식이라고 달아두었는데, 이는 순환적인 방식과는 달리, 각 index들이 이미 한개 단위로 분할되었다는 가정 하에서 진행한 것이기 때문입니다. 따라서 분할을 할 필요가 없고 1개 씩에서 2개씩으로, 2개씩에서 4개씩으로, ... 합병해 나가면 됩니다. 그러기 위해서는 right가 한번의 병합마다 2 \* i 씩 커지게 해주고 마지막 index를 초과해버리면 마지막 index가 되게 해주고 left는 그 전 병합때의 right에 +1을 한 값으로 해서 right가 마지막 index일 경우에 다시 0으로 초기화해줘야 합니다.

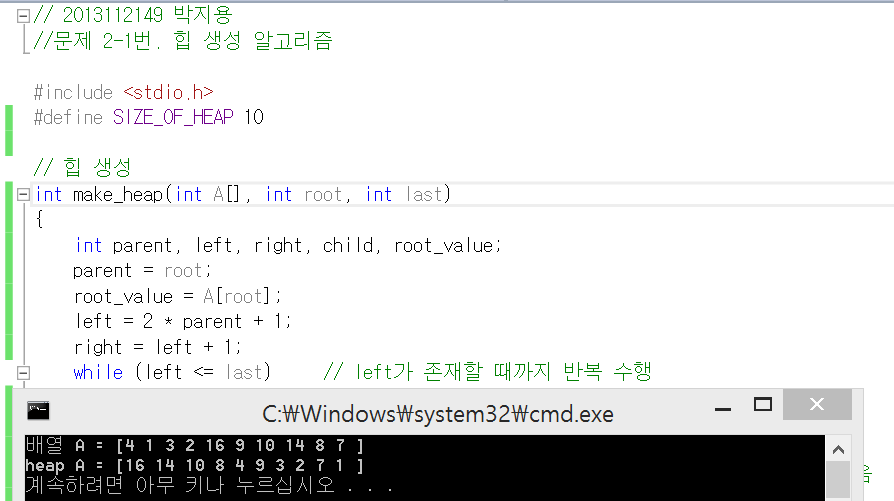
나머지 설명은 주석으로 달아두었습니다.

* **실습문제 2번 – (1). 힙 생성 알고리즘 구현하기**

<그림>



<출력 결과>



힙 생성을 구현한 2-1번 문제의 출력화면입니다. 무작위로 값이 주어져있는 배열 A를 힙 생성 함수로 변형하여 maxheap으로 만들었습니다. 이는 주어진 데이터를 가지고 해당 알고리즘을 통해 제가 직접 그려보았던 maxheap의 모습과 일치합니다. 소스코드에 대한 설명은 아래에 있습니다.

<2-1번 소스코드>

// 2013112149 박지용

//문제 2-1번. 힙 생성 알고리즘

#include <stdio.h>

#define SIZE\_OF\_HEAP 10

// 힙 생성

int make\_heap(int A[], int root, int last)

{

int parent, left, right, child, root\_value;

parent = root;

root\_value = A[root];

left = 2 \* parent + 1;

right = left + 1;

while (left <= last) // left가 존재할 때까지 반복 수행

{

if (right <= last && A[left] < A[right])

{

child = right; // 왼쪽과 오른쪽 중 큰놈이 후레자식이 될 자격을 얻음

/\* last에 대한 조건을 준 이유 : 완전 이진트리에서 마지막 노드가

left가 될 수 있기 때문에 right는 존재하지 않을 수도 있다 \*/

}

else child = left;

if (root\_value < A[child])

{

A[parent] = A[child]; // parent 자리에 child 자리의 원소를 넣어줌

parent = child; // child를 부모로 바꿔줌

left = parent \* 2 + 1; // 그에 맞게 자식들도 바꿔줌

right = left + 1;

}

else break;

}

A[parent] = root\_value; // 결국 while문이 다 돌고 남아있는 빈 자리에 root\_value를 넣어줌

}

int main(void)

{

int A[] = { 4, 1, 3, 2, 16, 9, 10, 14, 8, 7 };

int i, j, n = SIZE\_OF\_HEAP;

printf("배열 A = [");

for (i = 0; i < n; i++) // 정렬 전 배열 출력

printf("%d ", A[i]);

printf("]\n");

for (i = 1; i <= n / 2; i++) // n/2으로 해도 값의 변화가 없다는 것을 확인

{

for (j = i; j >= 0; j--)

make\_heap(A, j, n - 1); // 배열을 heap으로 변환해줌(이중 for문 이용)

}

printf("heap A = [");

for (i = 0; i < n; i++) // 결과 배열 출력

printf("%d ", A[i]);

printf("]\n");

return 0;

}

<소스코드 설명>

make\_heap함수는 어떤 root값을 받아 root와 그 자식들을 비교하고 그 자식이 다시 부모가 되서 그 자식들과 비교를 해서 비교당한 대상들끼리 올라갈 대상은 올라가고 내려갈 대상은 내려감으로써 자신의 올바른 자리를 찾는 함수입니다. 기본적인 비교 아이디어는 자식들(코드 상에서 left와 right) 중 더 큰 자식(child)을 찾아 부모(parent)와 비교해서 자식이 더 크면 부모의 자리를 차지하고(주석에서는 후레자식이라고 표현) 그 자식의 위치가 다시 부모가 되어 자신의 자식들과 같은 비교를 반복하게 됩니다. 여기서 자식의 index값이 계속 커지게 되는데 배열의 마지막 index보다 커지면 비교를 수행하지 않습니다.

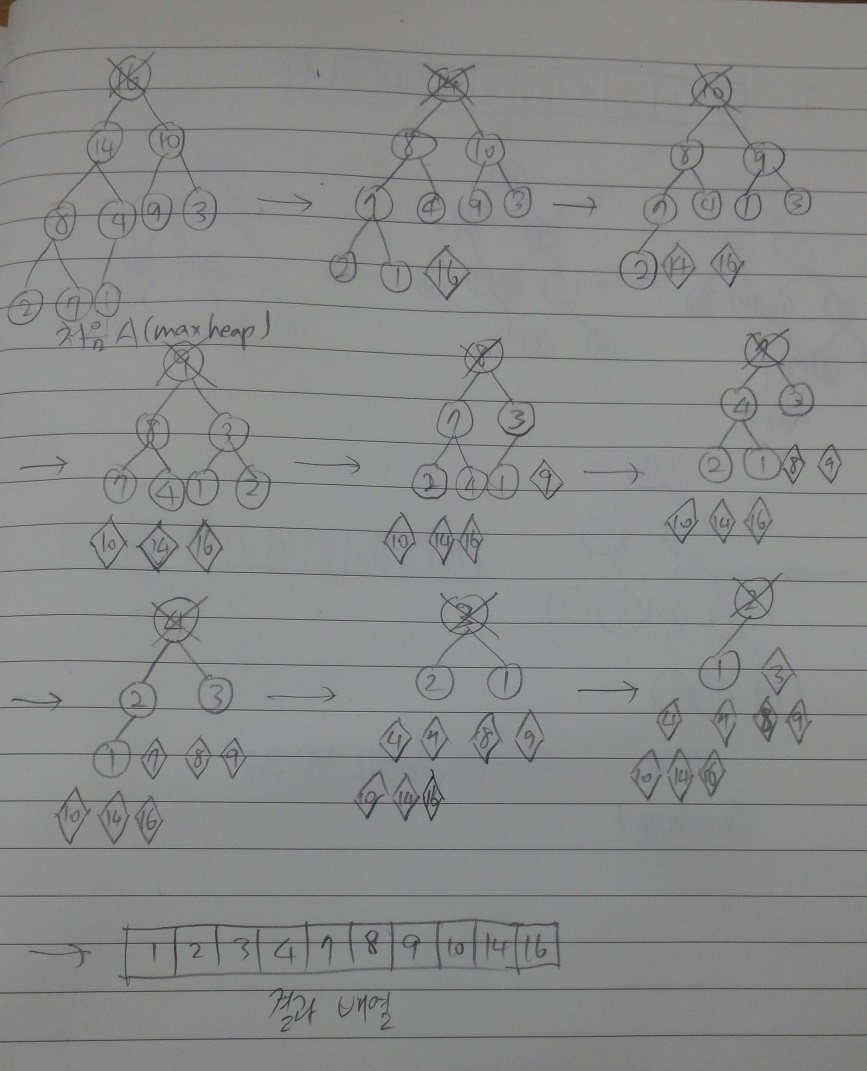
물론 결국 만들고자 하는 대상은 maxheap이며 이것은 make\_heap함수를 한번만 호출해서는 만들 수가 없습니다. 따라서 main함수에서 이중 for문을 돌려서 반복적으로 호출합니다. root의 index를 바꿔가면서 호출을 해야 배열의 모든 대상에 대한 비교가 가능해지기 때문입니다.

참고로 책과 과제 ppt에 나와있는대로 for문을 한개만 돌려 호출하면 예를 들어서 말하면 i=3에서 만들어진 A배열의 정보를 i=2에서는 기억하지 못합니다. i = 3에서 수행한 결과로 A배열은 바뀌었지만 i = 2에서 make\_heap(A, 2, n-1)에는 바뀌지 않은 A배열이 들어가기 때문입니다. 따라서 root\_value값도 엉망이 되어버리고 결국 나오는 결과도 의도치 않은 결과가 나오게 됩니다.

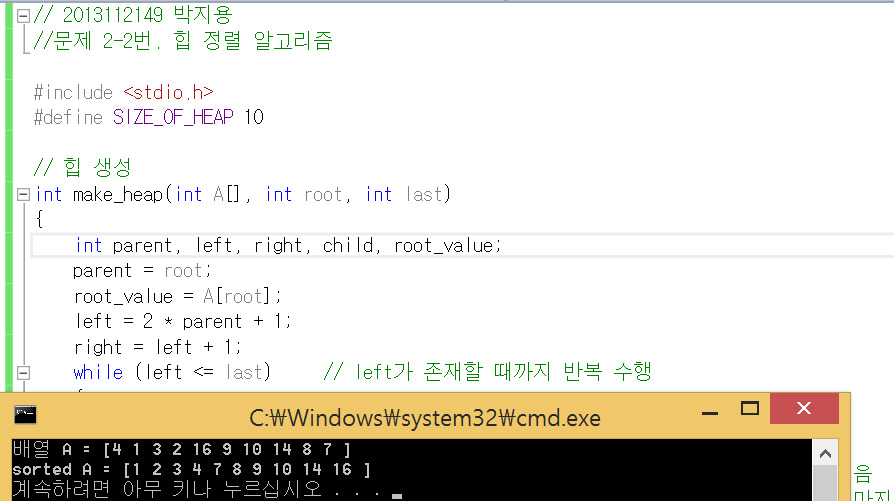
나머지 부수적인 내용은 주석으로 달아놓았습니다.

* **실습문제 2번 – (2). 힙 정렬 구현하기**

<그림>



<출력 결과>



힙 정렬을 구현한 2-2번 문제 출력화면입니다. 무작위로 값이 부여되어 있는 배열 A를 넣어서 힙 정렬을 해서 오름차순으로 정렬한 결과를 정상적으로 출력하고 있습니다. 소스코드에 대한 설명은 아래에 있습니다.

<2-2번 소스코드>

// 2013112149 박지용

//문제 2-2번. 힙 정렬 알고리즘

#include <stdio.h>

#define SIZE\_OF\_HEAP 10

// 힙 생성

int make\_heap(int A[], int root, int last)

{

int parent, left, right, child, root\_value;

parent = root;

root\_value = A[root];

left = 2 \* parent + 1;

right = left + 1;

while (left <= last) // left가 존재할 때까지 반복 수행

{

if (right <= last && A[left] < A[right])

{

child = right; // 왼쪽과 오른쪽 중 큰놈이 후레자식이 될 자격을 얻음

/\* last에 대한 조건을 준 이유 : 완전 이진트리에서 마지막 노드가

left가 될 수 있기 때문에 right는 존재하지 않을 수도 있다 \*/

}

else child = left;

if (root\_value < A[child])

{

A[parent] = A[child]; // parent 자리에 child 자리의 원소를 넣어줌

parent = child; // child를 부모로 바꿔줌

left = parent \* 2 + 1; // 그에 맞게 자식들도 바꿔줌

right = left + 1;

}

else break;

}

A[parent] = root\_value; // 결국 while문이 다 돌고 남아있는 빈 자리에 root\_value를 넣어줌

}

// 배열의 index1과 index2를 입력해주었을 때 서로의 원소를 바꿔주는 함수

void swap(int A[], int index1, int index2)

{

int temp = A[index1];

A[index1] = A[index2];

A[index2] = temp;

}

// 힙 정렬 알고리즘

void heap\_sort(int A[], int n)

{

int i, j;

for (i = 1; i <= n / 2; i++)

{

for (j = i; j >= 0; j--)

make\_heap(A, j, n - 1); // 정렬할 배열을 heap으로 변환해줌

}

for (i = n - 1; i > 0; i--)

{

swap(A, 0, i); // heap의 root원소를 A[i]와 교환해줌

make\_heap(A, 0, i - 1); // 남은 것들로 새로운 heap을 생성

}

}

int main(void)

{

int i, n = SIZE\_OF\_HEAP;

int A[] = { 4, 1, 3, 2, 16, 9, 10, 14, 8, 7 };

printf("배열 A = [");

for (i = 0; i < n; i++) // 정렬 전 배열 출력

printf("%d ", A[i]);

printf("]\n");

heap\_sort(A, n);

printf("sorted A = [");

for (i = 0; i < n; i++) // 결과 배열 출력

printf("%d ", A[i]);

printf("]\n");

return 0;

}

<소스코드 설명>

make\_heap함수에 대해서는 2-1번 문제에서 설명했으니 생략하겠습니다. heap\_sort함수에서 2-1번 문제의 main함수에서처럼 올바른 maxheap을 생성해주고나서 heap의 root원소가 가장 큰 값이므로 배열 index 두개를 input으로 받아 그 원소값을 서로 바꿔주는 swap함수를 써서 root원소를 배열의 가장 뒤(n-1번째 index)로 옮겨준 다음 다시 나머지를 가지고 힙을 정렬시키고 정렬된 힙에서의 root원소를 n-2번째 index로 옮겨주고, ..., 를 정렬되지 않은 원소가 1개 남을 때까지 반복해줍니다. 그럼 결국 맨 뒤부터 가장 큰 순으로 정렬되게 됩니다.

나머지 부수적인 내용은 주석으로 달아놓았습니다.