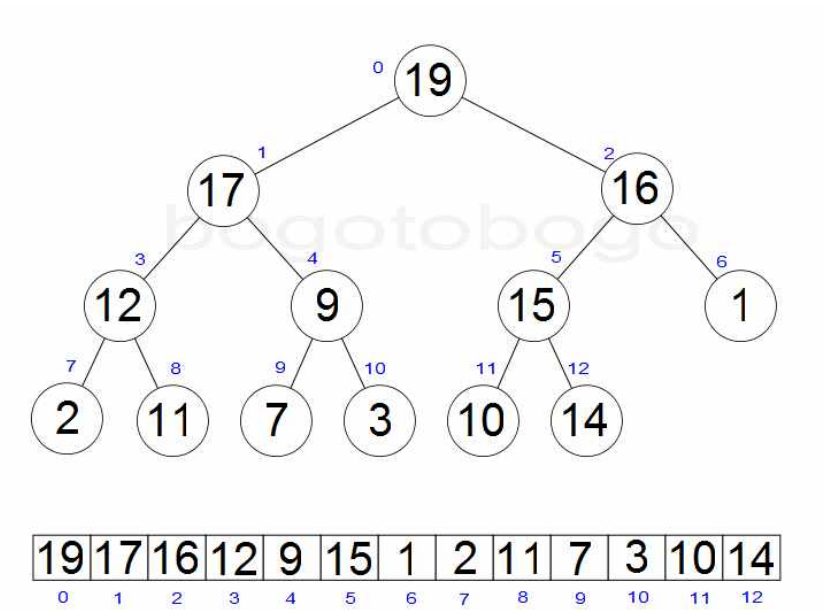
과목명	자료구조 및 알고리즘	분반	x	담당교수	김화성 교수님
학과	전자통신공학과	학번	2016707079	이름	하상천
Heap ADT: K'th element selection					

1, 과제설명 (과제에 대한 설명 및 목표)

Heap ADT를 구현하고(모든 함수를 다 구현 하지 않아도 됨),

강의 자료 (Heap)에 있는 selectK 함수와 이 함수의 동작을 테스트할 수 있는 테스트 함수를 구현하시오.

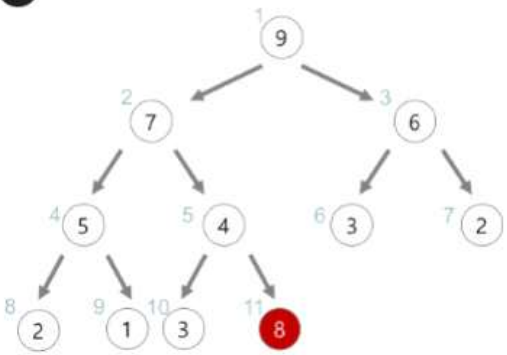
2, 이론 (과제와 관련한 일반적인 내용)



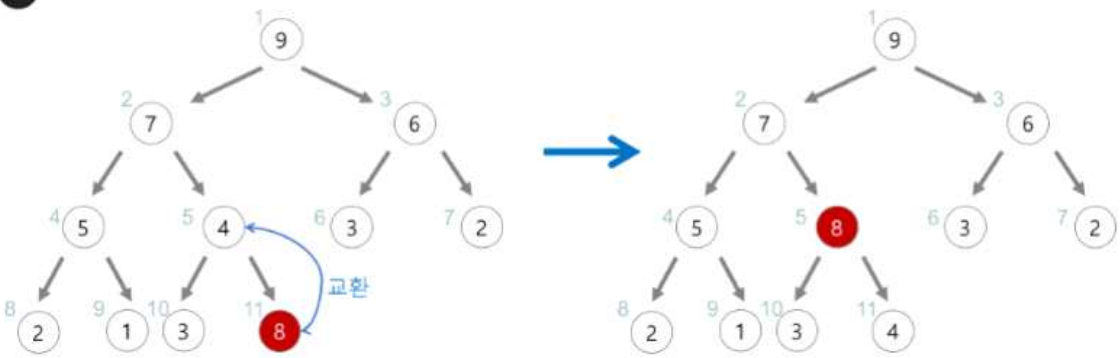
힙(heap)은 최댓값 및 최솟값을 찾아내는 연산을 빠르게 하기 위해 고안된 완전이진트리(complete binary tree)를 기본으로 한 자료구조이다.
heap 중에서 부모노드의 키 값이 자식노드의 키 값보다 항상 큰 Max heap의 구조이다.

3. 알고리즘 및 자료구조 설계 내용 기술

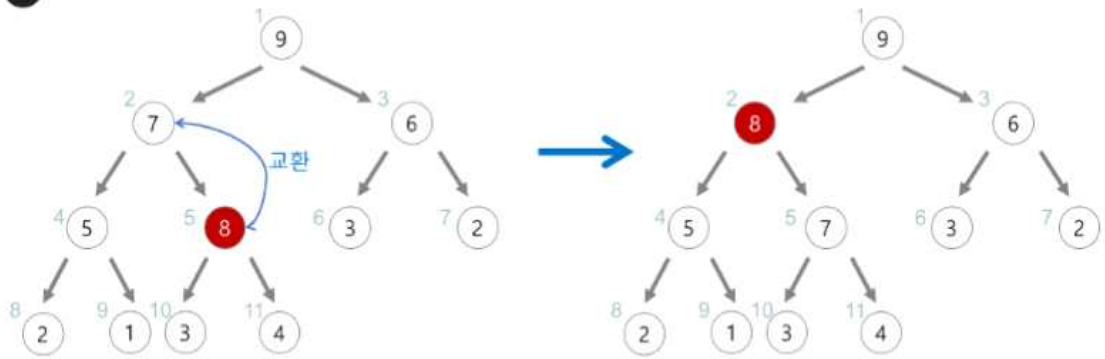
1 인덱스순으로 가장 마지막 위치에 이어서 새로운 요소 8을 삽입



2 부모 노드 4 < 삽입 노드 8 이므로 서로 교환

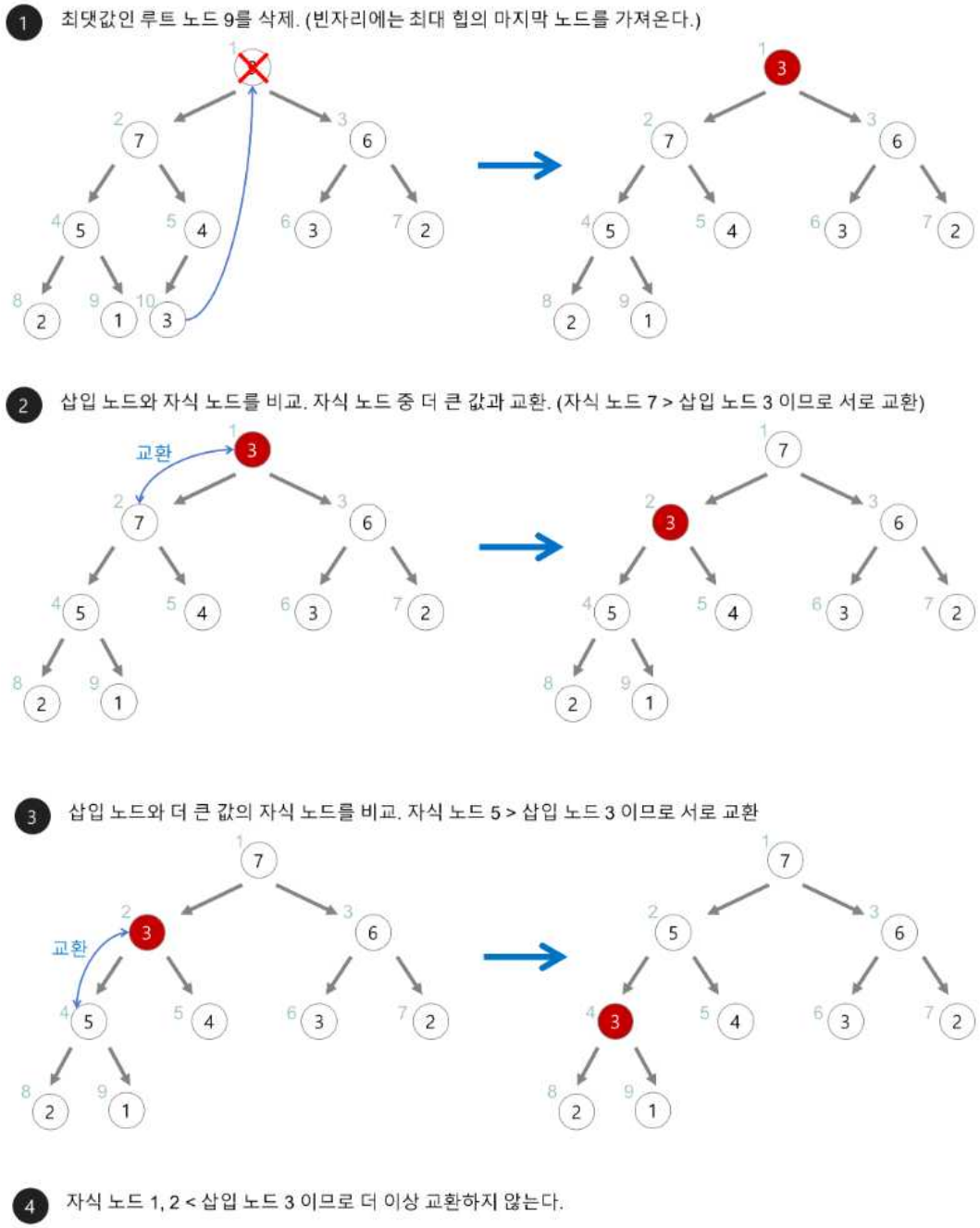


3 부모 노드 7 < 삽입 노드 8 이므로 서로 교환



4 부모 노드 9 > 삽입 노드 8 이므로 더 이상 교환하지 않는다.

Max heap에 새로운 요소 8을 삽입 할 때의 예시이다.



Max heap에서 최대값을 삭제하는 예시이다.

4. 소스코드 설명 (직접 작성한 소스코드 중에 핵심 부분을 발췌하여 설명)

```
Heap* CreateHeap(int size) {
        Heap* pNewHeap = (Heap*)malloc(sizeof(Heap)); //동적 메모리 할당
        if (pNewHeap == NULL) {
                return NULL;
        }
        pNewHeap->heapArray = (Element*)malloc(size * sizeof(Element)); //동적 메모리 할
당
        if (pNewHeap->heapArray == NULL) {
                free(pNewHeap);
                return NULL;
        }
        pNewHeap->maxSize = size;  // 초기화
        pNewHeap->size = 0; // 초기화
        pNewHeap->last = -1; // 초기화
        return pNewHeap;
 }
```

malloc을 통해 Heap size 만큼 동적 메모리를 할당 받고, parameter로 입력받은 size와 4 byte를 곱한 값만큼 동적 메모리를 할당 받아서 heapArray가 가리키게 한다. maxSize를 입력받은 size값으로 초기화하고, size를 0으로 last를 -1로 초기화한다.

```
void ReheapDown(Heap *pHeap, int index)
{
        if (index < 0 || index >= pHeap->size)
                return;
        if (index * 2 + 1 < pHeap->size) {      // left subtree가 존재하는지 확인하고 없으면
바로 함수 종료
                //right subtree를 확인하지 않는 이유는 Heap이 complete binary tree 이기 때문
에 왼쪽에 subtree가 없으면 오른쪽에도 subtree가 없기 때문이다.
                int maxChild = index * 2 + 1;       // maxchild를 left로 대입
                if (maxChild + 1 < pHeap->size && pHeap->heapArray[maxChild + 1] >
pHeap->heapArray[maxChild])
                        // right subtree가 존재하는지 확인하고, right subtree의 값이 left
subtree의 값보다 크다면 maxChild++을 통해 right로 바꿔준다.
                        maxChild++;
                if (pHeap->heapArray[maxChild] > pHeap->heapArray[index]) { //true:
general case, false: base case
                        // parent와 child의 data를 바꾸는 과정
                        Element temp = pHeap->heapArray[index];
                        pHeap->heapArray[index] = pHeap->heapArray[maxChild];
                        pHeap->heapArray[maxChild] = temp;
                        ReheapDown(pHeap, maxChild); //recursion
                }
        }
}
```

left subtree가 존재하는지 확인하고 없으면 바로 함수를 종료한다. right subtree를 확인하지 않는 이유는 Heap이 complete binary tree 이기 때문에 왼쪽에 subtree가 없으면 오른쪽에도 subtree가 없기 때문이다. 우선 maxChild를 left로 대입한다. right subtree가 존재하는지 확인하고, right subtree의 값이 left subtree의 값보다 크다면 maxChild++을 통해 right로 바꿔준다.
pHeap->heapArray[maxChild] > pHeap->heapArray[index]에서 true이면 general case이고, false이면 base case이다. true이면 parent와 child의 data를 바꿔주고 ReheapDown 함수를 recursion한다.

```
int SelectK(Heap *pHeap, int k) {
	int dataOut;
	int holdOut;
	int originalSize;

	if (k > pHeap->size) {
		return false;
	}
	originalSize = pHeap->size;
	for (int i = 0; i < k; i++) { //k번 반복하여 root data를 삭제
		DeleteHeap(pHeap, &dataOut);
		pHeap->heapArray[pHeap->size] = dataOut; //reconstruct heap을 위해서 root data를 heapArray의 대입
	}
	holdOut = pHeap->heapArray[0]; //우리가 원하는 data를 holdOut에 대입

	while (pHeap->size < originalSize) {
		pHeap->size++;
		pHeap->last++;
		ReheapUp(pHeap, pHeap->last);
	}
	return holdOut;
}
```

pHeap의 size를 originalSize에 대입시켜 놓고, k번 root data를 삭제한다. 삭제 할 때 reconstruct를 위해서 root data를 heapArray[pHeap->size]에 대입한다. 인덱스를 pHeap->size로 한 이유는 DeleteHeap함수가 실행 될 때 size가 1 감소했기 때문이다. k번 root data를 삭제하면 우리가 원하는 data 값이기 때문에 holdOut에 저장해놓고 다시 reconstruct를 한다. pHeap->size가 작아졌기 때문에 처음에 대입시켜 놓았던 originalSize 값과 비교하면서 반복문을 실행시켜 reconstruct한다.

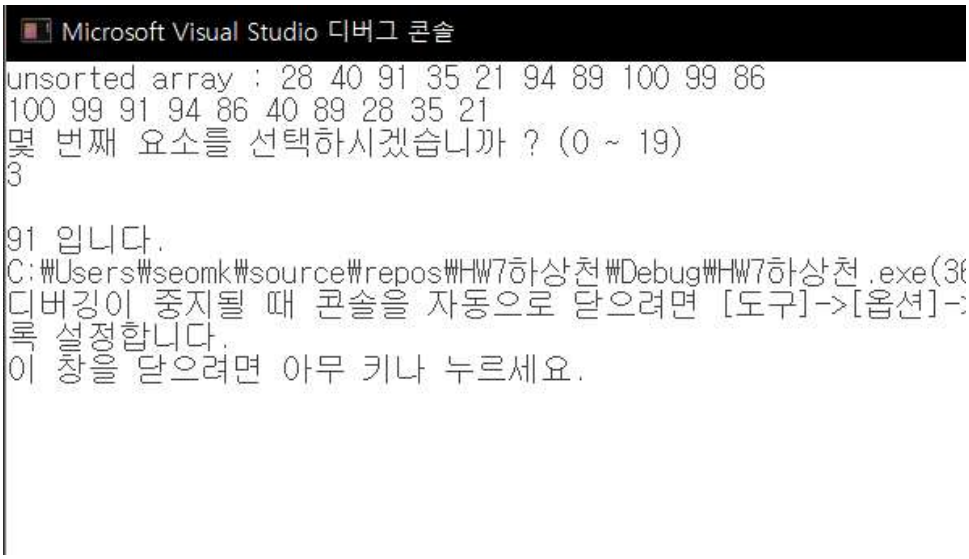
```
void MakeHeap(Heap *pHeap) {
	int num;
	srand(time(NULL)); // 계속 난수가 바뀌도록 한다.
	printf("unsorted array : ");

	for (int i = 0; i < 10; i++) {
		num = (rand() % 100) + 1;
		printf("%d ", num);
		InsertHeap(pHeap, num); // 1 ~ 100 범위의 난수 10개를 heap에 삽입
	}
	printf("₩n");
	for (int i = 0; i < 10; i++) {
		printf("%d ", pHeap->heapArray[i]);
	}
}
```

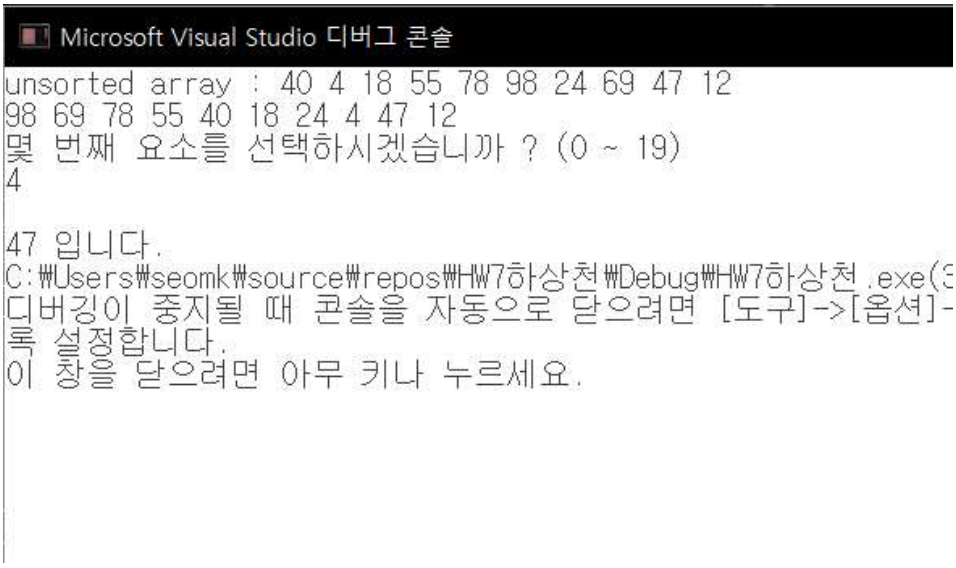
srand(time(NULL))을 통해 계속 난수가 바뀌도록 하고, 10개의 난수를 heap에 삽입한다. 난수만 하나하나 출력 할 때는 unsorted array 이기 때문에 InsertHeap 함수를 통해 heap을 만든다. 그리고 반복문을 통해 array를 출력한다.

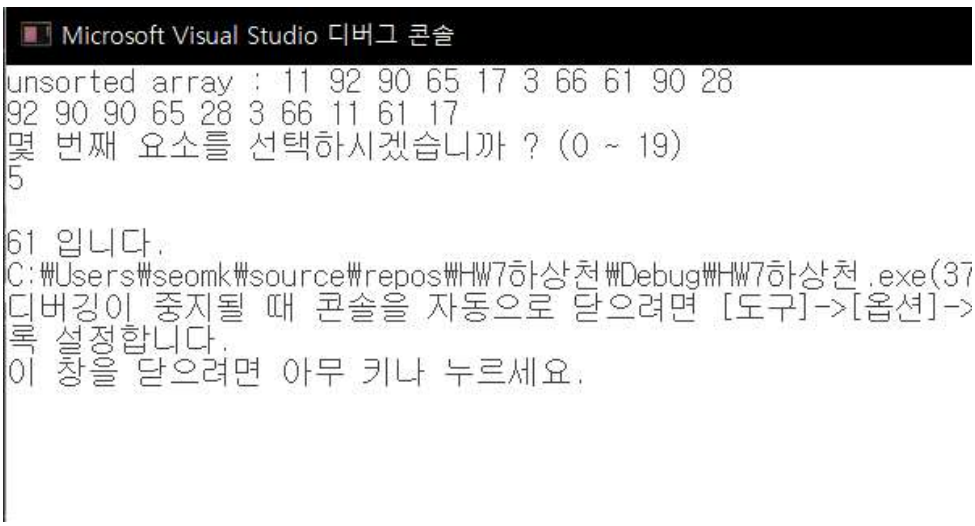
```
void Test(Heap *pHeap) {
	int key;

	printf("₩n몇 번째 요소를 선택하시겠습니까 ? (0 ~ 19)₩n"); //크기가 20인 array를 만들었기 때문이다.
	scanf("%d", &key);

	printf("₩n%d 입니다.", SelectK(pHeap, key));

}
```

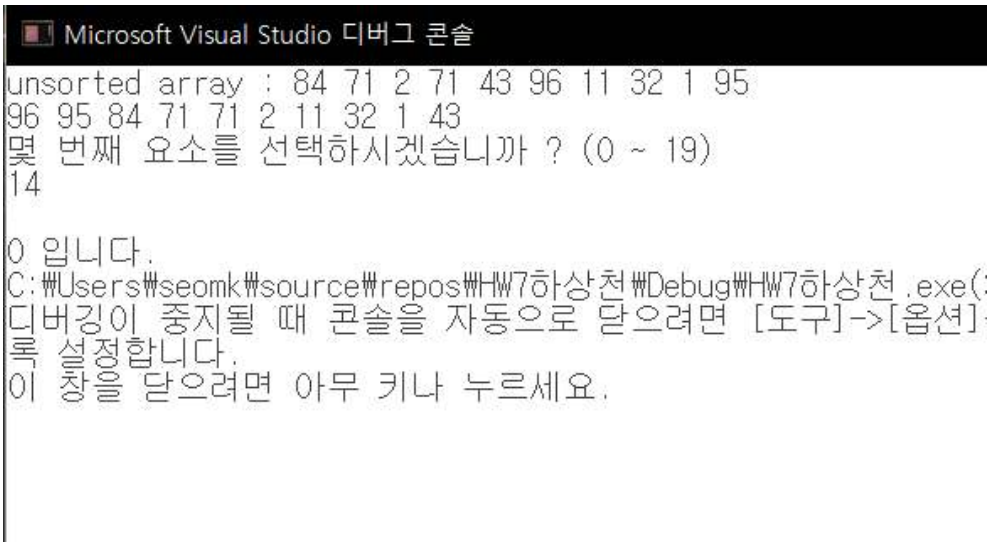
SelectK 함수의 동작을 테스트할 수 있는 Test 함수이다. 몇 번째 요소를 선택할 것인지 사용자에게 입력을 받고, SelectK 함수를 통해 결과값을 출력한다.

5. 실행결과 및 설명 (실행 결과를 캡쳐하여 첨부한 후 설명)


Microsoft Visual Studio 디버그 콘솔
unsorted array : 28 40 91 35 21 94 89 100 99 86
100 99 91 94 86 40 89 28 35 21
몇 번째 요소를 선택하시겠습니까 ? (0 ~ 19)
3
91 입니다.
C:₩Users₩seomk₩source₩repos₩HW7하상천₩Debug₩HW7하상천.exe(3
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구]->[옵션]->
록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요.


Microsoft Visual Studio 디버그 콘솔
unsorted array : 40 4 18 55 78 98 24 69 47 12
98 69 78 55 40 18 24 4 47 12
몇 번째 요소를 선택하시겠습니까 ? (0 ~ 19)
4
47 입니다.
C:₩Users₩seomk₩source₩repos₩HW7하상천₩Debug₩HW7하상천.exe(3
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구]->[옵션]-
록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요.


Microsoft Visual Studio 디버그 콘솔
unsorted array : 11 92 90 65 17 3 66 61 90 28
92 90 90 65 28 3 66 11 61 17
몇 번째 요소를 선택하시겠습니까 ? (0 ~ 19)
5
61 입니다.
C:₩Users₩seomk₩source₩repos₩HW7하상천₩Debug₩HW7하상천.exe(37
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구]->[옵션]->
록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요.



크기가 20인 array를 만들었기 때문에 0부터 19사이의 값을 입력 받았고, 3번째 요소를 선택하면 최대값을 3개 지우고 그 다음 최대값을 출력한다. array에 10개의 값을 넣었기 때문에 14번째 요소를 선택하면 0이 출력 된다.

(그림을 문서에 포함, 글자처럼 취급 옵션, 잉크 절약과 잘 보이게 하기위해 그림 반전)

6. 고찰 (과제를 진행하면서 배운점 이나, 시행 착오 내용, 기타 느낀점)

이번 과제는 Heap ADT를 구현하고 selectK 함수와 이 함수의 동작을 테스트할 수 있는 테스트 함수를 구현하는 것이었다. 처음에는 Heap 구조체 안에 last변수를 따로 선언하지 않고, size-1로 접근하여 메모리의 낭비를 막으려고 하였으나 가독성이 떨어지는 것 같아서 따로 선언을 하였다. 처음에 코드를 구현할 때는 알아보기 쉬웠는데, 며칠이 지나서 못 끝냈던 부분을 다시 구현하려고 할 때 이해하는데 시간이 오래 걸렸다. 본인이 작성한 코드도 이해하는데 시간이 오래 걸리는데, 남이 읽으면 더욱더 이해하기가 어렵겠다는 생각이 들었다. 그래서 앞으로 코드를 작성할 때는 가독성을 좋게 하고, 주석을 최대한 많이 달아야겠다는 생각을 하였다. 함수이름이나, 변수에 이름을 x,y 등등 아무렇게나 작성하는 것이 아니라 누가 읽어도 이 함수는 이러한 동작을 하겠구나, 이 변수는 이런 값을 가지고 있겠구나 추측할 수 있도록 이름을 작성할 때에도 신중을 기해야겠다. 난수가 계속 똑같이 나와서 인터넷에 검색을 해보니 시드(seed)를 바꾸어 주지 않았기 때문이었다. srand(time(NULL))을 이용해서 시드를 계속 지정해주면, 실행할 때 마다 시드 위치가 바뀌고 다른 난수가 발생하였다.

7. 전체 소스코드 (글자크기 9에 줄간격을 120%로 유지하고 한 줄이 너무 길지 않게 작성)

```
#include<stdio.h>
#include<stdlib.h>
#include <time.h>

typedef int Element;
#define true 1
#define false 0

typedef struct Heap {
        Element *heapArray; // array to store items
```

```
        int maxSize; // maximum size of heapArray
        int size; // 현재 heapArray에 들어가있는 갯수
        int last;
} Heap;

 Heap* CreateHeap(int size) {
         Heap* pNewHeap = (Heap*)malloc(sizeof(Heap)); //동적 메모리 할당
         if (pNewHeap == NULL) {
                 return NULL;
         }
         pNewHeap->heapArray = (Element*)malloc(size * sizeof(Element)); //동적 메모리 할
당
         if (pNewHeap->heapArray == NULL) {
                 free(pNewHeap);
                 return NULL;
         }
         pNewHeap->maxSize = size;  // 초기화
         pNewHeap->size = 0; // 초기화
         pNewHeap->last = -1; // 초기화
         return pNewHeap;
 }
 void ReheapUp(Heap *pHeap, int index)
 {
         int parentIndex = 0;
         if (index <= 0 || index >= pHeap->size) //base case
                 return;
         parentIndex = (index - 1) / 2;
         if (pHeap->heapArray[index] > pHeap->heapArray[parentIndex]) { // true : general
case, false : base case
                 // parent와 child의 data를 바꾸는 과정
                 Element temp = pHeap->heapArray[index];
                 pHeap->heapArray[index] = pHeap->heapArray[parentIndex];
                 pHeap->heapArray[parentIndex] = temp;
                 ReheapUp(pHeap, parentIndex); // recursion
         }
 }

void InsertHeap(Heap *pHeap, Element data)
{
        if (pHeap->size >= pHeap->maxSize) {
                printf("Heap is full!₩n");
                return;
        }
        pHeap->heapArray[++pHeap->last] = data;
        // ++last 를 한 이유는 배열이 0부터 시작하기 때문에 last를 1 증가시키고 인덱스로 사용
```

```
한다.
        // shape property
        pHeap->size++; //size 1 증가
        ReheapUp(pHeap, pHeap->last); // order property
}



void ReheapDown(Heap *pHeap, int index)
{
        if (index < 0 || index >= pHeap->size)
                return;
        if (index * 2 + 1 < pHeap->size) {      // left subtree가 존재하는지 확인하고 없으면
바로 함수 종료
                //right subtree를 확인하지 않는 이유는 Heap이 complete binary tree 이기 때문
에 왼쪽에 subtree가 없으면 오른쪽에도 subtree가 없기 때문이다.
                int maxChild = index * 2 + 1;       // maxchild를 left로 대입
                if (maxChild + 1 < pHeap->size && pHeap->heapArray[maxChild + 1] >
pHeap->heapArray[maxChild])
                        // right subtree가 존재하는지 확인하고, right subtree의 값이 left
subtree의 값보다 크다면 maxChild++을 통해 right로 바꿔준다.
                        maxChild++;
                if (pHeap->heapArray[maxChild] > pHeap->heapArray[index]) { //true:
general case, false: base case
                        // parent와 child의 data를 바꾸는 과정
                        Element temp = pHeap->heapArray[index];
                        pHeap->heapArray[index] = pHeap->heapArray[maxChild];
                        pHeap->heapArray[maxChild] = temp;
                        ReheapDown(pHeap, maxChild); //recursion
                }
        }
}

int DeleteHeap(Heap *pHeap, Element *pDataOut)
{
        if (pHeap->size <= 0) {
                printf("Heap is empty!₩n");
                return false;
        }
        *pDataOut = pHeap->heapArray[0];
        pHeap->heapArray[0] = pHeap->heapArray[pHeap->last--]; // heapArray에 마지막
값을 root에 대입하고 last를 1 감소시킨다.
        pHeap->size--; //size 1 감소
        ReheapDown(pHeap, 0);
        return true;
}
```

```
int SelectK(Heap *pHeap, int k) {
	int dataOut;
	int holdOut;
	int originalSize;

	if (k > pHeap->size) {
		return false;
	}
	originalSize = pHeap->size;
	for (int i = 0; i < k; i++) { //k번 반복하여 root data를 삭제
		DeleteHeap(pHeap, &dataOut);
		pHeap->heapArray[pHeap->size] = dataOut; //reconstruct heap을 위해서 root
data를 heapArray의 대입
	}
	holdOut = pHeap->heapArray[0]; //우리가 원하는 data를 holdOut에 대입

	while (pHeap->size < originalSize) {
		pHeap->size++;
		pHeap->last++;
		ReheapUp(pHeap, pHeap->last);
	}
	return holdOut;
}

void DestroyHeap(Heap *pHeap) { //동적 메모리 반납
	free(pHeap->heapArray);
	free(pHeap);
}

void MakeHeap(Heap *pHeap) {
	int num;
	srand(time(NULL)); // 계속 난수가 바뀌도록 한다.
	printf("unsorted array : ");

	for (int i = 0; i < 10; i++) {
		num = (rand() % 100) + 1;
		printf("%d ", num);
		InsertHeap(pHeap, num); // 1 ~ 100 범위의 난수 10개를 heap에 삽입
	}
	printf("₩n");
	for (int i = 0; i < 10; i++) {
		printf("%d ", pHeap->heapArray[i]);
```

```
        }
}

void Test(Heap *pHeap) {
        int key;

        printf("₩n몇 번째 요소를 선택하시겠습니까 ? (0 ~ 19)₩n"); //크기가 20인 array를 만들었기 때문이다.
        scanf("%d", &key);

        printf("₩n%d 입니다.", SelectK(pHeap, key));

}

int main() {
        Heap *pHeap = CreateHeap(20);
        MakeHeap(pHeap);
        Test(pHeap);

        DestroyHeap(pHeap);
}
```

(글자크기는 10으로 유지하고 줄간격도 160%를 유지할 것)