<알고리즘 실습> - 힙과 힙정렬(2)

※ 입출력에 대한 안내

- 특별한 언급이 없으면 문제의 조건에 맞지 않는 입력은 입력되지 않는다고 가정하라.
- 특별한 언급이 없으면, 각 줄의 맨 앞과 맨 뒤에는 공백을 출력하지 않는다.
- 출력 예시에서 □는 각 줄의 맨 앞과 맨 뒤에 출력되는 공백을 의미한다.
- 입출력 예시에서 ↦ 이 후는 각 입력과 출력에 대한 설명이다.

힙 정렬(heap sort)은 **힙**을 이용한 정렬이다. 무순리스트(unordered list)에 대한 힙 정렬은 두 단계로 진행된다. **1기**는 **힙 생성 단계**로서 초기 리스트로부터 **최대힙**을 생성한다. **2기**는 **힙 정렬 단계**로서 전 단계에서 생성된 최대힙으로부터 **오름차순 정렬**을 수행한다. 힙 정렬은 **O(n log n)** 시간에 수행한다.

1기에서 생성되는 힙은 **삽입식** 또는 **상향식** 가운데 한 방식을 사용하여 각각 **순차힙** 또는 **연결힙** 둘 중 한 형태로 생성할 수 있다(**3**주차 실습문제 참고).

만약 **순차힙**으로 구현한다면, **1**기와 마찬가지로 **2**기에서도 제자리 힙 정렬이 가능하다. 즉, 추가 메모리를 사용하지 않고 초기 배열 내에서 정렬을 완성할 수 있다.

반대로 **연결힙**으로 구현한다면, **1**기에서 초기 리스트의 키들에 대해 동적메모리 노드들을 할당하여 힙을 생성한다. 그리고 **2**기를 진행하는 동안 힙 정렬의 결과를 초기 리스트에 다시 저장하여 반환해야 한다.

[문제 1] 힙 정렬 - 유일 키

다음 조건에 맞추어 힙 정렬 프로그램을 작성하라.

1) **순차힙**으로 구현한다. 즉, 배열을 이용한 순차트리 형식으로 힙을 저장한다.
2) 연산 효율을 위해 배열 인덱스 **0** 위치는 사용하지 않고 비워둔다.
3) 데이터구조를 단순화하기 위해 힙의 항목으로써 (**키**, **원소**) 쌍에서 원소를 생략하고 **키**만 저장하는 것으로 한다.
4) 키들은 중복이 없는 **1** 이상의 정수로 전제한다 - 즉, 중복 키에 대한 처리는 불필요하다.
5) 최대 키 개수 < **100** 으로 전제한다.
6) **1**기(힙 생성 단계)에서 **삽입식** 또는 **상향식** 가운데 어떤 방식을 사용해도 좋다.
7) **O(n log n)** 시간, **O(1)** 공간에 수행해야 한다.

입출력 형식:

1) main 함수는 아래 형식의 표준입력으로 키들을 한꺼번에 입력받는다.

   입력 : 첫 번째 라인 : 키 개수
   두 번째 라인 : 키들

2) main 함수는 아래 형식의 표준출력으로 정렬된 리스트를 인쇄한다.

출력 : 첫 번째 라인 : 정렬 리스트

입력 예시 1

3	↦ 키 개수
209 400 77	↦ 키들

출력 예시 1

□77 209 400	↦ 정렬 리스트

입력 예시 2

6	↦ 키 개수
24 17 33 50 60 70	↦ 키들

출력 예시 2

□17 24 33 50 60 70	↦ 정렬 리스트

입력 예시 3

8	↦ 키 개수
5 15 10 20 30 25 31 29	↦ 키들

출력 예시 3

□5 10 15 20 25 29 30 31	↦ 정렬 리스트

필요 데이터구조:

- **H** 배열
  - 크기: **100** 미만
  - 데이터 형: 정수
  - 내용: 힙
  - 범위: 전역
- **n** 변수
  - 데이터 형: 정수
  - 내용: 리스트 또는 힙의 크기(총 키 개수)
  - 범위: 전역

필요 함수:

- main() 함수
  - 인자: 없음
  - 반환값: 없음
  - 내용: 초기 리스트에 대해 힙 정렬의 **1**기와 **2**기 작업을 수행하여 정렬 리스트를 인쇄하고 종료.
- inPlaceHeapSort() 함수
  - 인자: 없음
  - 반환값: 없음
  - 내용: **n**개의 키로 구성된 무순배열을 제자리 힙 정렬
  - 시간 성능: **O**(**n** log **n**)
- printArray() 함수
  - 인자: 없음
  - 반환값: 없음
  - 내용 : 배열에 저장된 키들을 차례로 인쇄

- 시간 성능: O(n)

◦ downHeap(i) 함수 : 3주차 문제 1의 downHeap과 동일

◦ insertItem(key) 함수 : 3주차 문제 1의 insertItem과 동일(삽입식 힙 생성에서 사용)

◦ upHeap(i) 함수 : 3주차 문제 1의 upHeap과 동일(삽입식 힙 생성에서 사용)

◦ rBuildHeap(i) 함수 : 3주차 문제 2의 rBuildHeap과 동일(재귀 상향식 힙 생성에서 사용)

◦ buildHeap() 함수 : 3주차 문제 2의 buildHeap과 동일(비재귀 상향식 힙 생성에서 사용)

[문제 2] 힙 정렬 – 중복 키

중복 키를 처리할 수 있도록 작성된 알고리즘은 **유일 키** 환경에서도 정확히 작동한다. 하지만 **유일 키**를 처리할 수 있도록 작성된 알고리즘은 **중복 키** 환경에서도 정확히 작동한다는 보장이 없다.

이번 문제는 문제 1에서 작성한 힙 정렬 관련 알고리즘들이 **중복 키**를 정확히 처리하는지 검증하고, 만약 부정확하다면 수정하는 것이다.

즉, 다음 조건 한 개를 제외하고는 문제 1과 동일하다.

1) 키들은 중복이 있을 수 있는 1 이상의 정수로 전제한다.

입출력 형식:

1) main 함수는 아래 형식의 표준입력으로 키들을 한꺼번에 입력받는다.

입력 : 첫 번째 라인 : 키 개수
두 번째 라인 : 키들

2) main 함수는 아래 형식의 표준출력으로 정렬된 리스트를 인쇄한다.

출력 : 첫 번째 라인 : 정렬 리스트

입력 예시 1

3	↦ 키 개수
209 400 209	↦ 키들

출력 예시 1

□209 209 400	↦ 정렬 리스트

입력 예시 2

6	↦ 키 개수
24 17 17 50 24 24	↦ 키들

출력 예시 2

□17 17 24 24 24 50	↦ 정렬 리스트

입력 예시 3

8	↦ 키 개수
31 10 10 20 31 31 31 10	↦ 키들

출력 예시 3

□10 10 10 20 31 31 31 31	↦ 정렬 리스트

필요 데이터구조: 문제 1과 동일

필요 함수: 문제 1과 동일