

## <알고리즘 실습> - 힙과 힙정렬(1)

### ※ 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  $\mapsto$  이 후는 각 입력과 출력에 대한 설명이다.

힙은 **이진트리(binary tree)**로 구현된 **우선순위 큐(priority queue)**다. 여기서 이진트리는 배열을 이용한 **순차트리** 형태로, 또는 연결리스트를 이용한 **연결트리** 형태로 구현할 수 있다. 순차트리로 구현된 힙을 **순차힙**, 연결트리로 구현된 힙을 **연결힙**이라고 부른다.

우선순위 큐의 각 항목은 (**키, 원소**) 쌍이며 이진트리의 각 노드로 구현된다. 이진트리의 루트에 최소 키를 저장한 경우 **최소힙(min-heap)**, 반대로 최대 키를 저장한 경우 **최대힙(max-heap)**이라 부른다.

**힙 생성**은 **삽입식(insertion)**과 **상향식(bottom-up)**의 두 가지 방식이 있다. **삽입식**은 모든 키들이 미리 주어진 경우, 또는 키들이 차례로 주어지는 경우, 양쪽에 적용 가능하지만 **상향식**은 전자인 경우에만 적용 가능하다. 동일 키들의 동일 순서로 구성된 초기 리스트라도 **삽입식, 상향식** 중 어느 방식을 적용하느냐에 따라 상이한 힙이 생성될 수 있다.

다음 문제 **1**과 **2**는 각각 **삽입식** 및 **상향식** 두 가지 버전의 **힙 생성**에 대한 프로그래밍 요구다. 다음은 두 문제 모두에 공통된 요구 사항이다.

- 1) **순차힙**으로 구현한다. 즉, 배열을 이용한 순차트리 형식으로 힙을 저장한다.
- 2) **최대힙**으로 구현한다. 따라서 힙으로부터 삭제 작업은 우선순위 큐에서 일반적으로 이루어지는 최소값 삭제가 아닌 최대값 삭제가 된다(참고로 최소힙에서는 최소값 삭제를 수행함).
- 3) 연산 효율을 위해 배열 인덱스 0 위치는 사용하지 않고 비워둔다.
- 4) 데이터구조를 단순화하기 위해 힙의 항목으로써 (**키, 원소**) 쌍에서 원소를 생략하고 **키만** 저장하는 것으로 한다.
- 5) 키들은 중복이 없는 1 이상의 정수로 전제한다 – 즉, 중복 키 검사는 불필요하다.
- 6)  **$O(1)$**  공간으로 수행할 것 – 즉, 주어진 키들을 저장한 초기 배열 외 추가 메모리 사용은  **$O(1)$** 을 초과할 수 없다.
- 7) 힙은 어느 시점에서라도 최대 항목 수  $< 100$  으로 전제한다.

### [ 문제 1 ] 삽입식 힙 생성

다음의 대화식 프로그램을 작성해 **삽입식 힙 생성**을 구현하라.

- 1) 키들은 한 개씩 차례로 삽입 명령과 함께 주어진다. 즉, 키가 입력될 때마다 즉시 힙에 삽입해야 한다. 만약 이렇게 하지 않고 문제 **2**에서 하는 것처럼 키들이 모두 입력되기를 기다려 한꺼번에 상향식으로 힙을 생성하면 대화식 프로그램의 인쇄(**p**) 또는 삭제(**d**)

명령의 수행이 어려워진다.

- 2) 대화식 프로그램에 주어지는 명령은 **i**, **d**, **p**, **q** 네 가지며 각 명령에 대해 다음과 같이 수행해야 한다.

**i** <키> : <키>를 힙에 삽입하고 **0**을 인쇄.

**d** : 힙에서 삭제(이때 루트에 저장된 키가 삭제되어야 한다)하여 반환된 키를 인쇄.

**p** : 힙의 내용을 인쇄(이때 이진트리의 **레벨순서**로 항목들을 인쇄해야 한다).

\* **레벨순서**란 이진트리의 레벨 **0**의 노드부터 다음 레벨 **1, 2, 3, ...**의 노드들을 차례로 방문한다. 같은 레벨의 노드들은 왼쪽에서 오른쪽으로 방문한다.

**q** : 프로그램 종료

주의:

1. 힙 인쇄 명령 **p**는 힙의 크기가 **1** 이상인 경우에만 적용한다고 전제한다.

힌트:

1. 현재 총 항목 수를 유지하는 변수 **n**을 유지하고 삽입 및 삭제 직후 갱신한다.  
2. **순차힙**에 새로운 항목의 초기 삽입 위치는 항상 **n + 1**이다.

입출력 형식:

- 1) **main** 함수는 아래 형식의 표준입력으로 대화식 명령들을 입력받는다.

입력 : 첫 번째 : 명령 **i**

두 번째 ~ 마지막-1 라인 : 명령 **i**, **d**, **p** 가운데 하나

마지막 : 명령 **q**

- 2) **main** 함수는 아래 형식의 표준출력으로 각 명령에 응답한다.

출력 : 0 (**i** 명령인 경우)

키 (**d** 명령인 경우)

힙 내용 (**p** 명령인 경우)

입력 예시 1

출력 예시 1

i 209	↳ 키 209 삽입	0	↳ 삽입 완료
i 400	↳ 키 400 삽입	0	↳ 삽입 완료
d	↳ 삭제	400	↳ 삭제 키 반환
i 77	↳ 키 77 삽입	0	↳ 삽입 완료
d	↳ 삭제	209	↳ 삭제 키 반환
p	↳ 인쇄	□77	↳ 힙 인쇄
q	↳ 종료		

입력 예시 2

i 24	↳ 키 24 삽입
i 17	↳ 키 17 삽입
i 33	↳ 키 33 삽입
p	↳ 인쇄
d	↳ 삭제
i 50	↳ 키 50 삽입
p	↳ 인쇄
q	↳ 종료

출력 예시 2

0	↳ 삽입 완료
0	↳ 삽입 완료
0	↳ 삽입 완료
□33 17 24	↳ 힙 인쇄
33	↳ 삭제 키 반환
0	↳ 삽입 완료
□50 17 24	↳ 힙 인쇄

입력 예시 3

i 5	↳ 키 5 삽입
i 15	↳ 키 15 삽입
i 10	↳ 키 10 삽입
i 20	↳ 키 20 삽입
i 30	↳ 키 30 삽입
i 25	↳ 키 25 삽입
p	↳ 인쇄
d	↳ 삭제
i 31	↳ 키 31 삽입
i 29	↳ 키 29 삽입
d	↳ 삭제
p	↳ 인쇄
q	↳ 종료

출력 예시 3

0	↳ 삽입 완료
0	↳ 삽입 완료
0	↳ 삽입 완료
0	↳ 삽입 완료
0	↳ 삽입 완료
0	↳ 삽입 완료
□30 20 25 5 15 10	↳ 힙 인쇄
30	↳ 삭제 키 반환
0	↳ 삽입 완료
0	↳ 삽입 완료
31	↳ 삭제 키 반환
□29 20 25 5 15 10	↳ 힙 인쇄

#### 필요 데이터구조:

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

#### 필요 함수:

- **main()** 함수
  - 인자: 없음
  - 반환값: 없음
  - 내용: 반복적으로 **i**, **d**, **p** 명령에 따라 **insertItem**, **removeMax**, **printHeap** 함수를 각각 호출 수행, **q** 명령 입력 시 종료.
- **insertItem(key)** 함수

- 인자: 정수 **key**
- 반환값: 없음
- 내용: **n** 위치에 **key** 삽입, **upHeap(n)** 호출 수행 후 **n**(총 키 개수)을 갱신
- 시간 성능:  $O(\log n)$
- **removeMax()** 함수
  - 인자: 없음
  - 반환값: 삭제된 키(정수)
  - 내용: **downHeap** 호출 수행 후 **n**(총 키 개수)을 갱신
  - 시간 성능:  $O(\log n)$
- **upHeap(i)** 함수
  - 인자: 배열 인덱스 **i**
  - 반환값: 없음
  - 내용: 힙 내 위치 **i**에 저장된 키를 크기에 맞는 위치로 상향 이동
  - 시간 성능:  $O(\log n)$
- **downHeap(i)** 함수
  - 인자: 배열 인덱스 **i**
  - 반환값: 없음
  - 내용: 힙 내 위치 **i**에 저장된 키를 크기에 맞는 위치로 하향 이동
  - 시간 성능:  $O(\log n)$
- **printHeap()** 함수
  - 인자: 없음
  - 반환값: 없음
  - 내용 : 힙에 저장된 키들을 레벨순서로 인쇄
  - 시간 성능:  $O(n)$

**알고리즘 설계 팁:**

<b>Alg insertItem(key)</b>	{힙 삽입}
input integer <b>key</b>	{ <b>key</b> : 삽입 키}
array <b>H</b>	{ <b>H</b> : 전역 배열, heap}
output array <b>H</b>	{ <b>H</b> : 전역 배열, heap}
1. <b>n</b> ← <b>n</b> + 1	{ <b>n</b> 갱신}
2. <b>H[n]</b> ← <b>key</b>	{힙에 초기 삽입 위치는 <b>n</b> }
3. <b>upHeap(n)</b>	{힙 조정}
4. return	

<b>Alg removeMax()</b>	{힙 삭제}
input array H	{H: 전역 배열, heap}
output array H	{H: 전역 배열, heap}
1. <b>key</b> ← H[1]	{루트 키 보관}
2. H[1] ← H[n]	{힙의 마지막 키를 루트로 이동}
3. <b>n</b> ← n - 1	{n 갱신}
4. <b>downHeap</b> (1)	{힙 조정}
5. return <b>key</b>	{삭제 키 반환}

## [ 문제 2 ] 상향식 힙 생성

다음 프로그램을 작성해 **상향식 힙 생성**을 구현하라.

- 1) 이번엔 키들이 미리 한꺼번에 주어진다. 이들을 주어진 순서대로 초기 배열에 저장한다.
- 2) 초기 배열에 저장된 키들을 상향식으로 재배치하여 힙을 생성한다. 상향식 힙 생성을 위한 **재귀** 또는 **비재귀** 방식의 알고리즘 가운데 어느 전략을 사용해도 좋다(사용하지 않은 나머지 전략도 나중에 작성해보자).
- 3) 참고로 **재귀**, **비재귀** 두 가지 방식 모두 **O(n)** 시간에 수행 가능하므로(왜 그런지 복습하자) 그렇게 되도록 작성해야 한다.

**힌트:** 문제 1 삽입식 힙 생성에서 이미 작성한 **downHeap**과 **printHeap** 함수를 그대로 사용하면 된다.

### 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 생성된 힙을 인쇄한다.

**출력 :** 첫 번째 라인 : 힙 내용 (레벨 순서)

입력 예시 1

3	↪ 키 개수
209 400 77	↪ 키들

출력 예시 1

□400 209 77	↪ 힙 인쇄
-------------	--------

입력 예시 2

6	↪ 키 개수
24 17 33 50 60 70	↪ 키들

출력 예시 2

□70 60 33 50 17 24	↪ 힙 인쇄
--------------------	--------

입력 예시 3

8	↪ 키 개수
5 15 10 20 30 25 31 29	↪ 키들

출력 예시 3

□31 30 25 29 15 5 10 20	↪ 힙 인쇄
-------------------------	--------

**필요 데이터구조:**

- **H** 배열
  - 크기: **100** 미만
  - 내용: 힙
  - 범위: 전역
- **n** 변수
  - 내용: 힙의 크기(현재 총 키 개수)
  - 범위: 전역

**필요 함수:**

- **main()** 함수
  - 인자: 없음
  - 반환값: 없음
  - 내용: 재귀적 **rBuildHeap(1)** 또는 비재귀적 **buildHeap()** 호출하여 힙 생성 후, 힙을 인쇄하고 종료.
- **rBuildHeap(i)** 함수
  - 인자: 정수 **i** (부분 힙의 루트 인덱스)
  - 반환값: 없음
  - 내용: 재귀 방식으로 상향식 힙 생성
  - 시간 성능: **O(n)**
- **buildHeap()** 함수
  - 인자: 없음
  - 반환값: 없음
  - 내용: 비재귀 방식으로 상향식 힙 생성
  - 시간 성능: **O(n)**
- **downHeap(i)** 함수 : 문제 1의 **downHeap**과 동일
- **printHeap()** 함수 : 문제 1의 **printHeap**과 동일

**알고리즘 설계 팁:**

<b>Alg rBuildHeap(i)</b>	{힙 생성 - 재귀 버전}
input integer i	{i: 현재 부트리의 루트 인덱스}
array H	{H: 전역 배열, non-heap}
output array H	{H: 전역 배열, heap}
1. if (i > n)	{n: 전역 변수}
return	
2. rBuildHeap(2i)	{현재 부트리의 좌 부트리를 힙 생성}
3. rBuildHeap(2i + 1)	{현재 부트리의 우 부트리를 힙 생성}
4. downHeap(i)	{현재 부트리의 루트와 좌우 부트리를 합친 힙 생성}
5. return	

<b>Alg buildHeap()</b>	{힙 생성 - 비재귀 버전}
input array H	{H: 전역 배열, non-heap}
output array H	{H: 전역 배열, heap}
1. for $i \leftarrow n/2$ downto 1	{n: 전역 변수, 마지막 내부노드부터 역방향으로 루트까지}
downHeap(i)	
2. return	