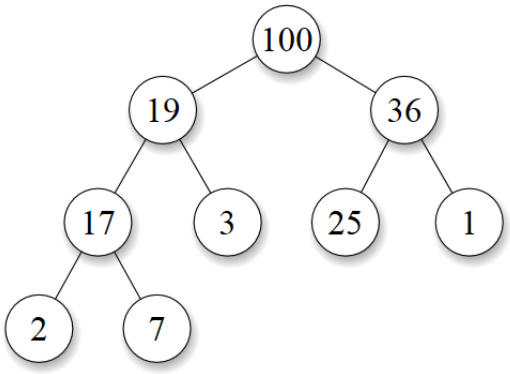
6. 자료구조(5) – 힙(Heap)

• **힙(Heap) 이란?**

힙(Heap)은 우선순위 큐(Priority Queue)의 일종으로 우선순위가 높은 요소를 효율적으로 선택할 수 있는 자료구조이다.
힙은 주로 나무구조로 구현하며 나무구조에서 부모의 키 값을 두 자식의 키 값보다 크게 만들어 주면 된다.
즉, 힙의 뿌리(root)는 전체 데이터 중에서 가장 우선순위가 높은 자료이고 뿌리의 자식은 뿌리보다 작은 두 개의 자료로 구성하면 된다.



[트리로 구성된 우선순위 큐]

** 나무 구조를 배열로 구현하는 법

나무 구조에서 층별로 나무를 타는 방법(Level order Traverse)으로 Root노드를 1번으로 시작해서 번호를 붙이면 다음이 규칙대로 저장되게 된다.

1. 번호 j를 갖는 노드의 부모의 번호는 j/2 이다.
2. 번호 j를 갖는 노드의 왼쪽 자식의 번호는 j*2이고, 오른쪽 자식은 j*2+1 이다.
3. 힙 자료의 개수 n의 절반(n/2)까지가 내부 노드이다.

▶ **Heap(배열로 구현된 heap)에 데이터가 저장되는 과정 이해**

저장 데이터 : 1 2 3 4 5 6 7 8 9 순으로 heap에 저장

1. heap의 가장 마지막 위치에서 데이터를 저장 한 후
2. 부모의 값과 비교하며 upHeap을 실시한다.

index	0	1	2	3	4	5	6	7	8	9
Input : 1	MAX	*1*								
Input : 2	MAX		*2*							
2가 부모 1보다 크므로 upHeap	MAX	**2**	**1**							
Input : 3	MAX	2	1	*3*						
3이 부모 2보다 크므로 upHeap	MAX	**3**	1	**2**						
Input : 4	MAX	3	1	2	*4*					
4가 부모 1보다 크므로 upHeap	MAX	3	**4**	2	**1**					
4가 부보 3보다 크므로 upHeap	MAX	**4**	**3**	2	1					
Input : 5	MAX	4	3	2	1	*5*				
5가 부모 3보다 크므로 upHeap	MAX	4	**5**	2	1	**3**				
5가 부모 4보다 크므로 upHeap	MAX	**5**	**4**	2	1	3				
Input : 6	MAX	5	4	2	1	3	*6*			
6이 부모 2보다 크므로 upHeap	MAX	5	4	**6**	1	3	**2**			
6이 부모 5보다 크므로 upHeap	MAX	**6**	4	**5**	1	3	2			
Input : 7	MAX	6	4	5	1	3	2	*7*		
7이 부모 5보다 크므로 upheap	MAX	6	4	**7**	1	3	2	**5**		
7이 부모 6보다 크므로 upheap	MAX	**7**	4	**6**	1	3	2	5		
Input : 8	MAX	7	4	6	1	3	2	5	*8*	
8이 부모 1보다 크므로 upHeap	MAX	7	4	6	**8**	3	2	5	**1**	
8이 부모 4보다 크므로 upHeap	MAX	7	**8**	6	**4**	3	2	5	1	
8이 부모 7보다 크므로 upHeap	MAX	**8**	**7**	6	4	3	2	5	1	
Input : 9	MAX	8	7	6	4	3	2	5	1	*9*
9가 부모 4보다 크므로 upHeap	MAX	8	7	6	**9**	3	2	5	1	**4**
9가 부모 7보다 크므로 upHeap	MAX	8	**9**	6	**7**	3	2	5	1	4
9가 부모 8보다 크므로 upHeap	MAX	**9**	**8**	6	7	3	2	5	1	4
Heap(배열)에 저장된 최종순서	MAX	9	8	6	7	3	2	5	1	4

▶ **Heap(배열로 구현된 heap)에서 데이터가 삭제되는 과정 이해**

1. heap의 최상단(root) 위치에서 데이터를 꺼낸 후
2. heap의 가장 마지막 위치의 데이터를 root위치로 올리고
3. 마지막 위치에는 MIN값을 저장한 후
4. root위치의 값에 대해 downHeap을 실시한다.

index	0	1	2	3	4	5	6	7	8	9
Output : 9	MAX	*9*	8	6	7	3	2	5	1	4
마지막데이터 4가 root자리로 이동	MAX	**4**	8	6	7	3	2	5	1	**MIN**
4가 자식 8보다 작으므로 downHeap	MAX	**8**	**4**	6	7	3	2	5	1	MIN
4가 자식 7보다 작으므로 downHeap	MAX	8	**7**	6	**4**	3	2	5	1	MIN
Output : 8	MAX	*8*	7	6	4	3	2	5	1	MIN
마지막데이터 1이 root자리로 이동	MAX	**1**	7	6	4	3	2	5	MIN	MIN
1이 자식 7보다 작으므로 downHeap	MAX	**7**	**1**	6	4	3	2	5	MIN	MIN
1이 자식 4보다 작으므로 downHeap	MAX	7	**4**	6	**1**	3	2	5	MIN	MIN
Output : 7	MAX	*7*	4	6	1	3	2	5	MIN	MIN
마지막데이터 5가 root자리로 이동	MAX	**5**	4	6	1	3	2	MIN	MIN	MIN
5가 자식 6보다 작으므로 downHeap	MAX	**6**	4	**5**	1	3	2	MIN	MIN	MIN
Output : 6	MAX	*6*	4	**5**	1	3	2	MIN	MIN	MIN
마지막데이터 2가 root자리로 이동	MAX	**2**	4	5	1	3	MIN	MIN	MIN	MIN
2가 자식 5보다 작으므로 downHeap	MAX	**5**	4	**2**	1	3	MIN	MIN	MIN	MIN
Output : 5	MAX	*5*	4	2	1	3	MIN	MIN	MIN	MIN
마지막데이터 3이 root자리로 이동	MAX	**3**	4	2	1	MIN	MIN	MIN	MIN	MIN
3이 자식 4보다 작으므로 downHeap	MAX	**4**	**3**	2	1	MIN	MIN	MIN	MIN	MIN
Output : 4	MAX	*4*	**3**	2	1	MIN	MIN	MIN	MIN	MIN
마지막데이터 1이 root자리로 이동	MAX	**1**	3	2	MIN	MIN	MIN	MIN	MIN	MIN
1이 자식 3보다 작으므로 downHeap	MAX	**3**	**1**	2	MIN	MIN	MIN	MIN	MIN	MIN
Output : 3	MAX	*3*	1	2	MIN	MIN	MIN	MIN	MIN	MIN
마지막데이터 2가 root자리로 이동	MAX	**2**	1	MIN	MIN	MIN	MIN	MIN	MIN	MIN
Output : 2	MAX	*2*	1	MIN	MIN	MIN	MIN	MIN	MIN	MIN
마지막데이터 1이 root자리로 이동	MAX	**1**	MIN	MIN	MIN	MIN	MIN	MIN	MIN	MIN
Output : 1	MAX	*1*	MIN	MIN	MIN	MIN	MIN	MIN	MIN	MIN
heap에서 데이터 꺼내기 종료										