



부산대학교

PUSAN NATIONAL UNIVERSITY

Heap Operations

Up and Down One Level

Insert, Remove, Heapify, and Heap Sort

DS3600082-140

자료구조

2026학년도 2학기

오용경

데이터사이언스전공

yongkyung.oh@pusan.ac.kr

Insert: Sift Up

값을 넣은 뒤 힙 성질을 되찾는 절차를 정한다

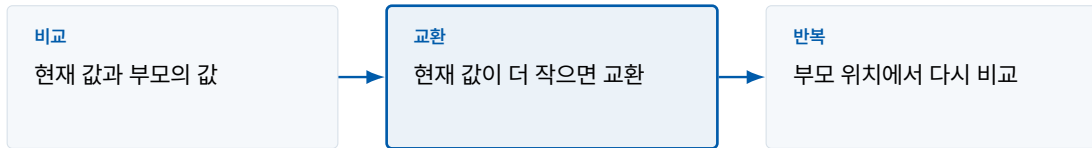
새 원소를 배열 끝에 넣는다

모양 조건을 지키려면 넣을 자리는 하나뿐이다

항목	내용
넣는 자리	마지막 레벨의 다음 빈 자리
배열에서의 위치	끝에 붙인다
깨질 수 있는 조건	힙 성질(새 값이 부모보다 작을 때)

sift up

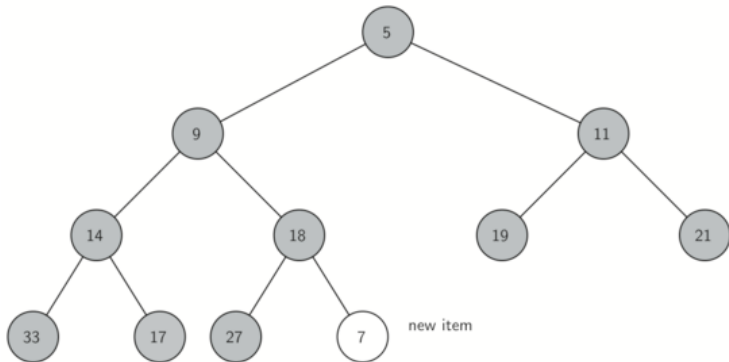
sift up(상향 이동, 새 값을 부모와 바꾸며 위로 올리기)은 삽입 뒤 힙 성질을 복원한다



종료 루트에 도달했거나 현재 값이 부모의 값 이상이면 멈춘다

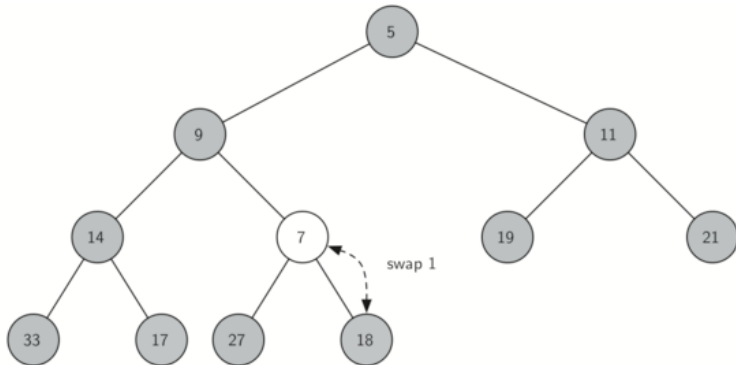
교환할 때마다 한 레벨씩 올라가므로 많아야 높이만큼 반복한다

7을 마지막 자리에 넣는다



새로 넣은 흰 노드 7이 부모 18보다 작으므로 힙 성질이 깨져 있다.

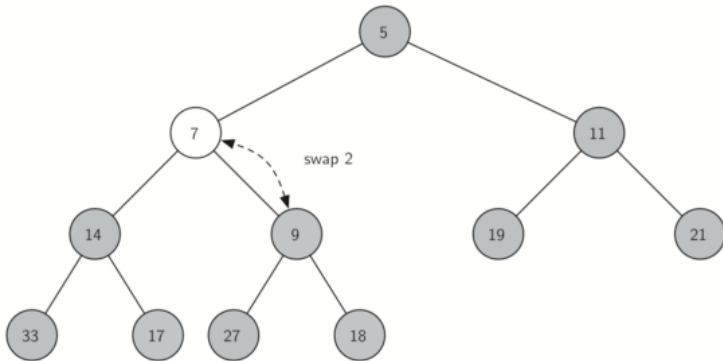
첫 번째 교환



18과 자리를 바꾼 7이 새 부모 9보다 여전히 작으므로 한 번 더 올라간다.

Source: Miller and Ranum (2013), Binary Heap Implementation.

두 번째 교환에서 멈춘다



9와 자리를 바꾼 7이 부모 5보다 크므로 두 번째 교환에서 멈춘다.

배열로 보면

같은 일을 배열의 인덱스 계산으로 적으면 이렇게 된다

단계	비교 (인덱스: 값)	결과
1	10: 7 $\leftrightarrow$ 부모 4: 18	교환
2	4: 7 $\leftrightarrow$ 부모 1: 9	교환
3	1: 7 $\leftrightarrow$ 부모 0: 5	멈춤

부모 인덱스는 $\lfloor (i - 1) / 2 \rfloor$ 다. 결과 배열은 [5, 7, 11, 14, 9, 19, 21, 33, 17, 27, 18]이다

비용

하는 일	횟수	이유
교환	많아야 높이	한 번에 한 레벨
높이	$\lfloor \log_2 n \rfloor$	완전 이진트리
전체	$O(\log n)$	

Checkpoint

Insert: Sift Up

- 새 값을 어디에 넣는가?
- 상향 이동은 언제 멈추는가?
- 삽입 비용이 $O(\log n)$ 인 이유는 무엇인가?

Remove Min: Sift Down

최솟값을 꺼낸 뒤 힙 성질을 되찾는다

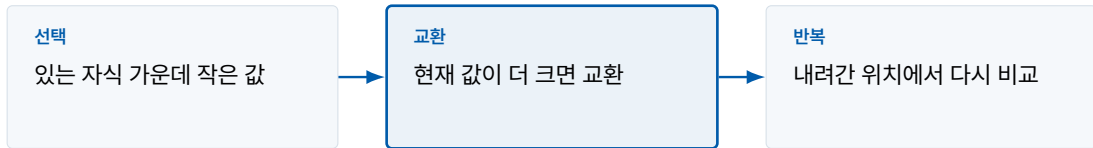
루트를 빼면 자리가 빈다

빈 루트에 아무 값이나 올릴 수는 없다

선택	결과
더 작은 자식 노드를 올림	빈자리가 아래로 내려가 모양 조건이 깨질 수 있다
마지막 자리의 값을 올림	마지막 칸이 비므로 모양 조건이 항상 지켜진다

sift down

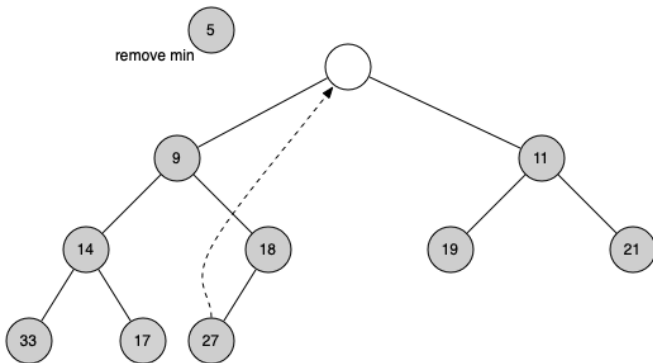
sift down(하향 이동, 루트 값을 더 작은 자식 노드와 바꾸며 내리기)은 삭제 뒤 힙 성질을 복원한다



종료 자식이 없거나 현재 값이 선택한 자식의 값 이하이면 멈춘다

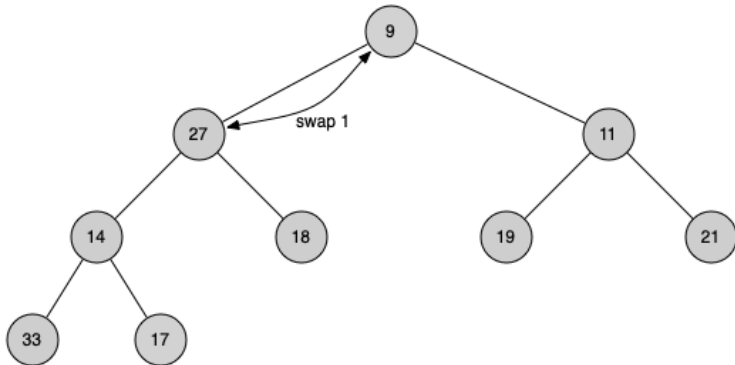
큰 자식과 바꾸면 새 부모가 다른 자식보다 커질 수 있으므로 더 작은 자식과 바꾼다

5를 꺼내고 마지막 값을 올린다



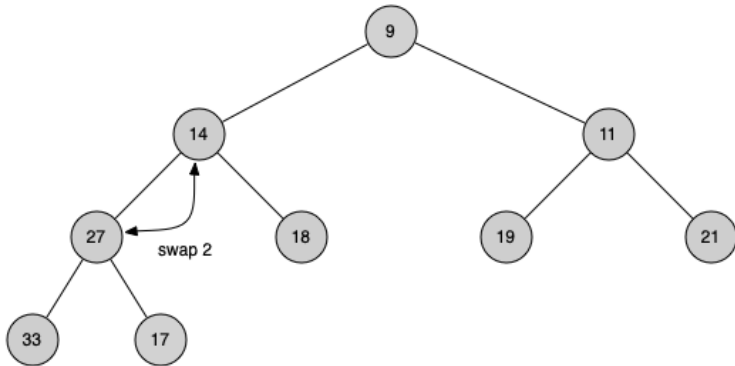
루트의 5를 꺼내고 마지막 자리의 27을 그 자리로 옮긴다.

더 작은 자식 노드와 바꾼다



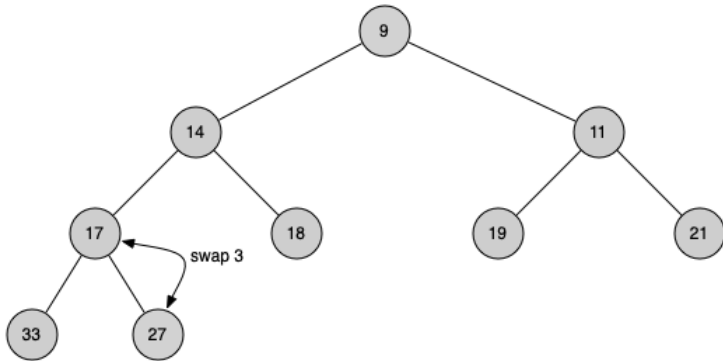
27이 두 자식 노드 9와 11 가운데 더 작은 9와 바뀌어 9가 루트가 된다.

한 레벨씩 내려간다



27이 14와 자리를 바꾸며 한 칸 더 내려간다.

제자리를 찾으면 멈춘다



17과 자리를 바꾼 27은 자식 노드가 없어 세 번째 교환에서 멈춘다.

하향 이동을 배열로 보면

27을 인덱스 0에 올린 [27, 9, 11, 14, 18, 19, 21, 33, 17]에서 시작한다

단계	비교 (인덱스: 값)	결과
1	0: 27 $\leftrightarrow$ 작은 자식 1: 9	교환
2	1: 27 $\leftrightarrow$ 작은 자식 3: 14	교환
3	3: 27 $\leftrightarrow$ 작은 자식 8: 17	교환
4	8: 27, 자식 인덱스 17은 배열 밖	멈춤

자식 인덱스는 $2i + 1$ 과 $2i + 2$ 다. 결과 배열은 [9, 14, 11, 17, 18, 19, 21, 33, 27]이다

자식 노드 수에 따른 교환 대상

레벨마다 비교는 두 번, 교환은 많아야 높이만큼이므로 최솟값 꺼내기도 $O(\log n)$ 이다

경우	바꿀 대상	참고
자식 노드가 둘	더 작은 쪽	비교 두 번
자식 노드가 하나	그 하나	마지막 레벨
자식 노드가 없음	없음, 멈춘다	단말 노드

Caution 큰 쪽과 바꾸면 그 자리에서 조건이 다시 깨진다

Checkpoint

Remove Min: Sift Down

- 루트를 뺀 자리에 무엇을 올리는가?
- 하향 이동에서 어느 자식 노드와 바꾸어야 하는가?
- 더 큰 자식 노드와 바꾸면 무엇이 잘못되는가?

Section 3 • Heap Operations

Building a Heap

배열 전체를 한 번에 힙으로 만든다

이미 있는 배열을 힙으로

값이 모두 주어져 있으면 하나씩 넣기보다 한꺼번에 만드는 편이 싸다

방법	비용	하는 일
n 번 삽입	$O(n \log n)$	매번 상향 이동
heapify	$O(n)$	아래에서 하향 이동

아래에서 위로

단계	하는 일	범위·근거
1	단말 노드는 그냥 둔다	이미 힙이다
2	마지막 부모부터 하향 이동	뒤에서 앞으로
3	루트까지 반복	인덱스 0까지

아래쪽 부분 트리가 이미 힙이므로 하향 이동 한 번이면 된다

heapify는 전체 배열을 $O(n)$ 에 힙으로 만든다

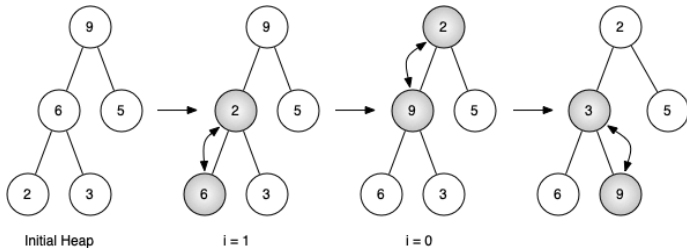
깊은 곳에 노드가 많고 그곳에서는 내려갈 거리가 짧다

레벨	노드 수	내려갈 거리
맨 아래	약 $n/2$	0
그 위	약 $n/4$	1
루트	1	$\log_2 n$

내려갈 거리의 합은 $1 \cdot n/4 + 2 \cdot n/8 + 3 \cdot n/16 + \dots$ 이고 이 합은 n 보다 작다

노드가 많은 아래쪽 레벨은 이동 거리가 짧아 총이동 횟수의 상한이 n 에 비례한다

만들어지는 과정



첫 트리 [9, 6, 5, 2, 3]은 아직 힙이 아닌 입력이다. 아래쪽 부모부터 차례로 하향 이동하며 힙이 된다.

heapify의 교환을 배열로 추적한다

인덱스는 0부터 세며 마지막 부모 $i = \lfloor 5/2 \rfloor - 1 = 1$ 에서 시작한다

시작 i	교환 (인덱스: 값)	배열 상태
입력	없음	[9, 6, 5, 2, 3]
1 (완료)	$1 : 6 \leftrightarrow 3 : 2$	[9, 2, 5, 6, 3]
0 (중간)	$0 : 9 \leftrightarrow 1 : 2$	[2, 9, 5, 6, 3]
0 (완료)	$1 : 9 \leftrightarrow 4 : 3$	[2, 3, 5, 6, 9]

$i = 0$ 에서 시작한 하향 이동은 9를 따라 인덱스 1에서도 계속된다. 두 번 교환한 뒤에야 루트의 하향 이동이 끝난다.

같은 값인데 배열이 다르다

9, 5, 6, 2, 3, 8, 1, 7, 4를 두 방법으로 힙으로 만들어 본다

방법	결과 배열
n 번 삽입	1, 3, 2, 4, 5, 8, 6, 9, 7
heapify	1, 2, 6, 4, 3, 8, 9, 7, 5

Caution 두 배열은 다르지만 둘 다 힙 조건을 지키므로 같은 값 집합의 힙이 하나로 정해지지 않는다

Checkpoint

Building a Heap

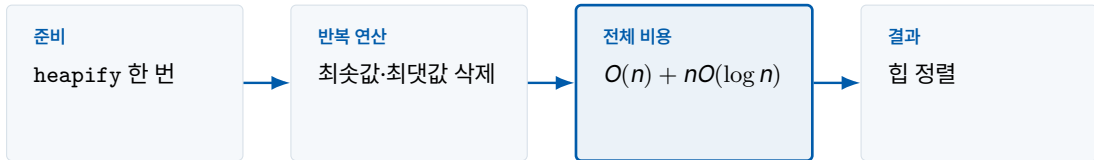
- n 번 삽입과 `heapify`의 비용은 각각 얼마인가?
- `heapify`가 아래에서 위로 가는 이유는 무엇인가?
- 같은 값으로 만든 두 힙의 배열이 다를 수 있는가?

Section 4 • Heap Operations

Heap Sort

힙 연산을 되풀이해 배열을 정렬한다

힙 정렬은 힙 연산을 반복한 응용이다



힙 정렬(heap sort)은 힙 연산의 비용을 합한 $O(n \log n)$ 정렬이다

계속 꺼내면 정렬이 된다

최솟값을 꺼내는 연산을 n 번 되풀이하면 오름차순이 나온다

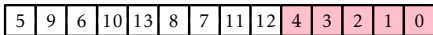
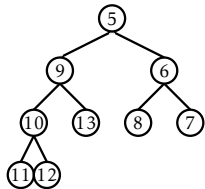
단계	비용	몇 번
heapify	$O(n)$	한 번
최솟값 꺼내기	$O(\log n)$	n 번
전체	$O(n \log n)$	

정의

제자리(in-place)

입력 배열 밖에 큰 추가 공간을 쓰지 않고 정렬하는 성질이다. 꺼낸 값을 배열의 뒤쪽 빈 자리에 넣으면 새 배열이 필요 없고, 힙이 앞에서 줄어드는 만큼 정렬된 부분이 뒤에서 자란다.

제자리 정렬의 결과 순서는 힙의 종류가 정한다



최소 힙으로 정렬하는 중간 상태다. 앞 9칸이 힙이고, 음영 칸에는 꺼낸 최솟값이 끝에서부터 쌓여 4, 3, 2, 1, 0의 내림차순이 되었다. 다 쌓은 뒤 배열을 뒤집는다.

사용하는 힙

제자리 정렬 결과

최소 힙

내림차순, 뒤집으면 오름차순

최대 힙

큰 값이 뒤부터 쌓여 오름차순

꺼낸 값을 뒤에 쌓으므로 오름차순에는 최대 힙을 쓰거나, 최소 힙으로 정렬한 뒤 뒤집는다

정의

안정성(stability)

키가 같은 값들의 원래 순서가 정렬 뒤에도 유지되는 성질이다. 같은 점수의 두 학생이 입력 순서대로 남는지를 말한다. 힙 정렬은 멀리 떨어진 자리를 교환하므로 안정적이지 않다. 예를 들어 키가 같은 a, b를 최대 힙으로 정렬하면 루트와 끝을 맞바꾸는 첫 교환에서 b, a 순서가 된다.

힙 정렬의 시간·공간·안정성

정렬 방법	최악 시간	제자리	안정성
정렬된 앞부분에 하나씩 끼워 넣기	$O(n^2)$	예	같은 키 뒤에 넣으면 안정적
BST에 넣고 중위 순회	$O(n^2)$	아니요	중복 키 처리 규칙에 따름
힙 정렬	$O(n \log n)$	예	불안정

힙 정렬은 최악 시간 복잡도가 $O(n \log n)$ 인 제자리 정렬이지만 안정적이지 않다

Checkpoint

Heap Sort

- 힙 정렬 전체의 비용은 어떻게 나오는가?
- 제자리 정렬이란 무엇인가?
- 힙 정렬이 안정적이지 않은 이유는 무엇인가?

Section 5 • Heap Operations

Python heapq

표준 모듈 heapq로 힙 연산을 수행한다

heappush로 차례로 넣으면

4, 9, 3, 7, 5, 1, 6, 8, 2를 차례로 heappush한다

확인할 것	실행 결과	읽는 법
내부 list	[1, 2, 3, 5, 7, 4, 6, 9, 8]	정렬 결과가 아님
루트	heap[0] == 1	최솟값
부모 인덱스	(i-1)//2	배열에서 위로 이동

리스트 전체는 정렬되지 않아도 부모가 자식보다 작다는 조건은 지킨다

heappop은 루트를 꺼내고 힙을 복원한다

마지막 원소를 루트에 놓은 뒤 더 작은 자식과 바꾸며 아래로 내린다

```
1 import heapq
2
3 heap = [1, 2, 3, 5, 7, 4, 6, 9, 8]
4 smallest = heapq.heappop(heap)
5 print(smallest) # 1
6 print(heap)     # [2, 5, 3, 8, 7, 4, 6, 9]
```

heappush는 끝에 넣고 위로, heappop은 루트를 꺼낸 뒤 아래로 이동하며 힙 성질을 복원한다

최대 힙 사용법

Python 버전	최대 힙 방법
Python 3.14 이상	<code>heappush_max</code> , <code>heappop_max</code>
Python 3.13 이하	값의 부호를 뒤집어 최소 힙 함수를 쓴다

Source: Python 3.14 Library Reference (n.d.), `heapq`.

힙인지 확인하기

인덱스 1부터 끝까지 각 칸을 부모 $(i-1)//2$ 와 비교한다

```
1 def is_heap(heap):
2     for i in range(1, len(heap)):
3         parent = (i-1) // 2
4         if heap[parent] > heap[i]:
5             return False
6     return True
7
8 is_heap([1, 2, 6, 4, 3, 8, 9])      # True
9 is_heap([4, 9, 3, 7, 5, 1, 6, 8, 2]) # False
```

부모와 자식 노드의 관계만 확인하며, 배열이 정렬되어 있는지는 힙 판정 기준이 아니다

우선순위 큐로 쓰기

튜플을 넣으면 첫 원소가 우선순위가 된다

넣는 것	뜻
(3, task)	우선순위 3인 작업
(1, urgent)	우선순위 1이므로 먼저 나온다

우선순위 큐가 필요할 때 직접 만들지 않고 이 모듈을 쓴다

같은 우선순위가 여럿이면

항목	내용
넣은 순서 보존	보장되지 않는다
튜플의 두 번째 원소	비교에 쓰이므로 비교할 수 없으면 예외가 난다
피하는 방법	(우선순위, 순번, 작업)처럼 순번을 둘째 자리에 둔다

Caution 비교할 수 없는 객체를 튜플 뒤에 두면 같은 우선순위에서 예외가 난다

Checkpoint

Python heapq

- heappush와 heappop은 각각 어떤 절차를 수행하는가?
- Python 3.14 이상과 이전 버전에서 최댓값을 먼저 꺼내는 방법은 어떻게 다른가?
- 같은 우선순위가 여럿일 때 무엇을 조심해야 하는가?

How Do You Keep a Tree From Leaning?

Next Lecture: AVL Balance

- 한쪽으로 치우친 트리의 문제
- 높이 균형과 균형 인수
- 불균형을 찾아내는 법