6조 최고은 / 이재윤

알고리즘
1주차

우선순위 큐의 구현

우선순위 큐

큐: **먼저 삽입**된 데이터를 **먼저 추출**

우선순위 큐: **우선순위가 높은 데이터**를 **먼저 추출**

(1) 우선순위 큐 구현 방법

배열 / 연결리스트 / 힙

모두 사용 가능하나,

배열은 삽입 및 삭제 시 데이터를 전부 한 칸씩 밀거나 당기는 연산 필요

배열, 연결리스트는 삽입의 위치를 찾기 위해 모든 데이터와 우선순위의 비교 필요

이러한 배열과 리스트의 단점 때문에,

힙이라는 자료구조를 이용해서 구현하는 것이 일반적임

리스트로 구현한 우선순위 큐

(2) 리스트에 기초한 우선순위 큐

무순리스트와 순서리스트로 구현

1. 무순리스트 성능

- 삽입: O(1), 무작위 삽입
- 삭제: O(n), 정렬하며 추출

Total: O(n), n개의 데이터: $O(n*n) = O(n^2)$

2. 순서리스트 성능

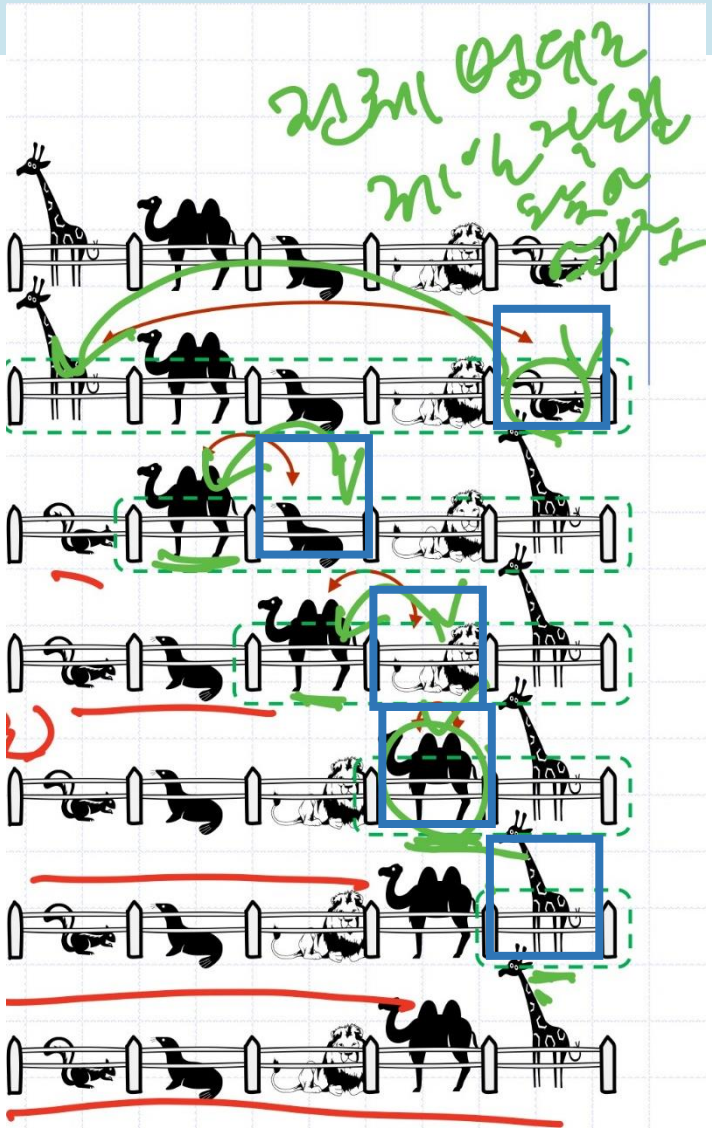
- 삽입: O(n), 정렬하며 삽입
- 삭제: O(1), 가장 앞 순위 값 추출

Total: O(n), n개의 데이터: $O(n*n) = O(n^2)$

제자리 리스트로 구현한 우선순위 큐

제자리: 상수 메모리만 추가로 사용
공간 성능 향상

제자리 선택 정렬



앞부분을 정렬 상태로 유지

무순리스트 (선택 정렬)

전체 범위에서 제일 작은 걸(우선순위가 높은 것) 찾아 앞으로 당김,
이후 2순위로 작은 걸 찾고, 3순위로 작은 걸 찾으면서 앞으로 당김

순서리스트 (삽입 정렬)

앞부분부터 차례로 훑어가면서 작은 것을 찾아 앞으로 당김,
더욱 작은 것이 나올 때마다
계속 앞으로 당기면서 업데이트 필요

제자리 리스트로 구현한 우선순위 큐

제자리: 상수 메모리만 추가로 사용
공간 성능 향상

제자리 삽입 정렬

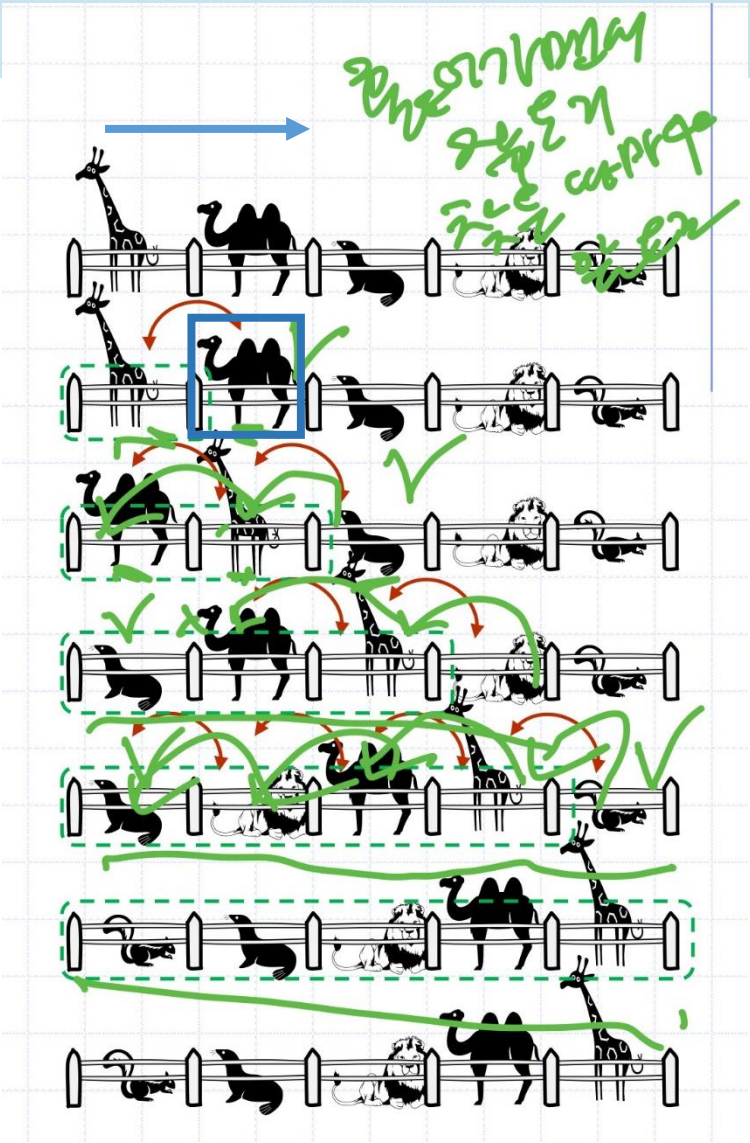
앞부분을 정렬 상태로 유지

무순리스트 (선택 정렬)

전체 범위에서 제일 작은 걸(우선순위가 높은 것) 찾아 앞으로 당김,
이후 2순위로 작은 걸 찾고, 3순위로 작은 걸 찾으면서 앞으로 당김

순서리스트 (삽입 정렬)

앞부분부터 차례로 훑어가면서 작은 것을 찾아 앞으로 당김,
더욱 작은 것이 나올 때마다
계속 앞으로 당기면서 업데이트 필요



힙의 특징

힙

(1) 힙의 조건

1. **완전 이진트리**

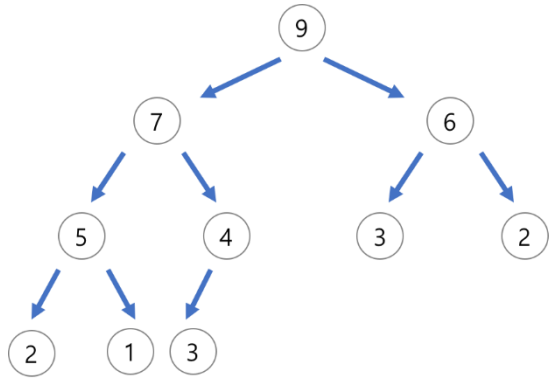
; 단말 노드를 제외한 나머지 노드가 두 개의 자식노드를 가지고 있는 트리

2. **모든 자식 노드의 데이터 <= 부모 노드의 데이터**

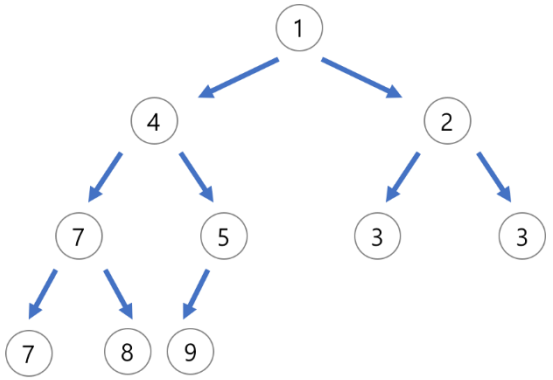
(= 자식 노드 데이터의 우선순위 <= 부모 노드 데이터의 우선순위)

(2) 힙의 종류

1. 최대힙: 루트 노드에 가까울수록 값이 커짐
2. 최소힙: 루트 노드에 가까울수록 값이 작아짐



-최대 힙(max heap)-



-최소 힙(min heap)-

힙의 특징

힙

(3) 힙의 구현 방식

1. 삽입식

모든 키 들이 미리 주어진 경우, 또는 키들이 차례로 주어지는 경우

양쪽에 적용 가능

2. 상향식

상향식은 모든 키 들이 미리 주어진 경우에만 적용 가능

재귀 / 비재귀적 방법으로 구현 가능

삽입식 힙 구현

```
int main() {
    /*
    <command>
    i : insert , p : print
    d : delete , q : quit
    */
    char command;
    int key;
    int root = 0;


    while (1) {
        scanf("%c", &command);

        if (command == 'i') {
            scanf("%d", &key);
            //n위치에 key 삽입
            insertItem(key);
            printf("0\n");
        }

        else if (command == 'd') {
            root = removeMax();
            printf("%d\n", root);
        }

        else if (command == 'p') {
            printHeap();
        }
        else if (command == 'q') {
            break;
        }
    }

}
```

main()

- 인자: 없음
- 반환값: 없음

- 내용: 반복적으로 i, d, p 명령에 따라 insertItem, removeMax, printHeap 함수를 각각 호출 수행, q 명령 입력 시 종료

- 메인에서는 사용자로부터 i,d,p 명령을 입력 받고, 각 명령에 맞는 함수를 호출

삽입식 힙 구현

```
void insertItem(key) {

    //힙의 n+1 위치에 key 삽입
    n = n + 1;
    Heap[n] = key;

    // 상향식 heapify
    upHeap(n);

}
```

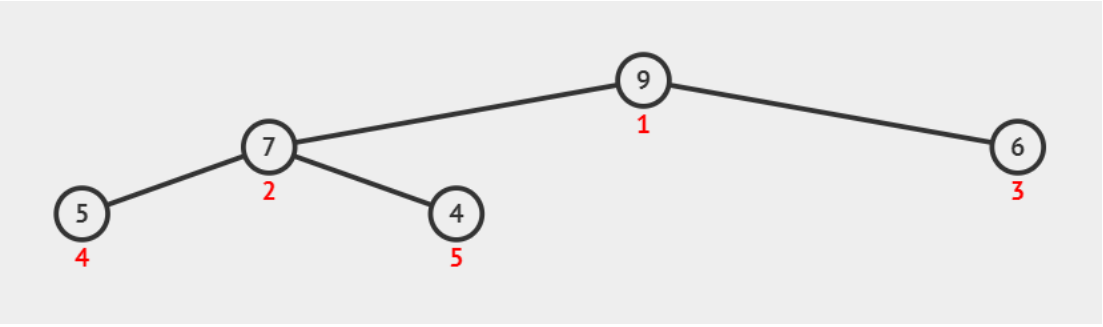
insertItem()

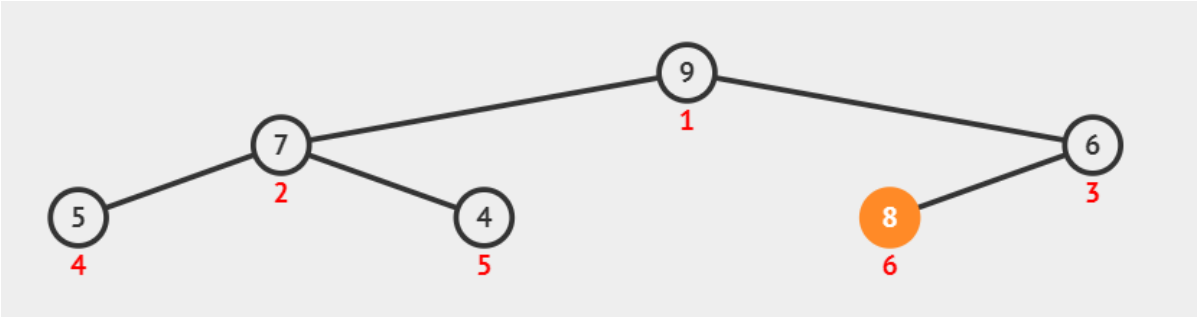
- 인자: 정수 key
- 반환값: 없음

- 내용: n 위치에 key 삽입, upHeap(n) 호출 수행 후 n을 갱신

- 시간 성능: O(log n)

메인에서 입력받은 key 값을 현재 힙에 삽입한다.
insertItem 호출전에 힙의 크기가 n 이라면, 배열 인덱스 n+1 에 키 값을 삽입한다.





삽입식 힙 구현

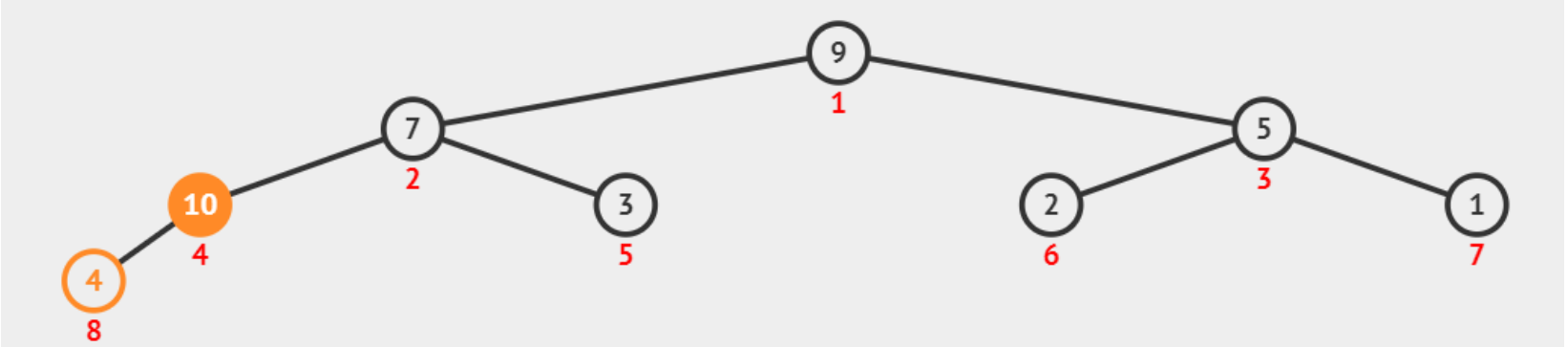
```
void upHeap(int v) {

    if (v == 1) { // isRoot ?
        return;
    }
    int parent = v / 2;

    if (Heap[v] <= Heap[parent]) {
        return;
    }

    //swap
    int temp = Heap[v];
    Heap[v] = Heap[parent];
    Heap[parent] = temp;
    upHeap(parent);

}
```

upHeap()

- 인자: 배열 인덱스 i
- 반환값: 없음

- 내용:
힙 내 위치 i에 저장된 키를 크기에 맞는 위치로 상향 이동

- 시간 성능: O(log n)



삽입식 힙 구현

```
int removeMax() {

    int key = Heap[1];
    Heap[1] = Heap[n];
    n--;
    downHeap(1);
    return key;
}
```

removeMax()

- 인자: 없음
- 반환값: 삭제된 키(정수) - 내용: downHeap 호출 수행 후 n(총 키 개수)을 갱신

- 시간 성능: O(log n)

힙 구조에서 removeMax()는 루트노드를 삭제한 뒤 key값을 반환하는 것

```
void downHeap(int v) {
    int left_child = 2 * v;
    int right_child = 2 * v + 1;

    // isleaf ?
    // 현재노드의 자식노드가 없다면 return
    if (left_child > n) {
        return;
    }


    int larger = left_child;

    //더 큰 자식을 찾고
    if (Heap[right_child] > Heap[larger]) {
        larger = right_child;
    }

    //자식과 부모노드를 비교해서, 자식이 크다면 교환
    if (Heap[v] < Heap[larger]) {
        int temp = Heap[v];
        Heap[v] = Heap[larger];
        Heap[larger] = temp;
    }
    //하향식 구조
    downHeap(larger);

}
```

삽입식 힙 구현

downHeap()

인자: 배열 인덱스 i

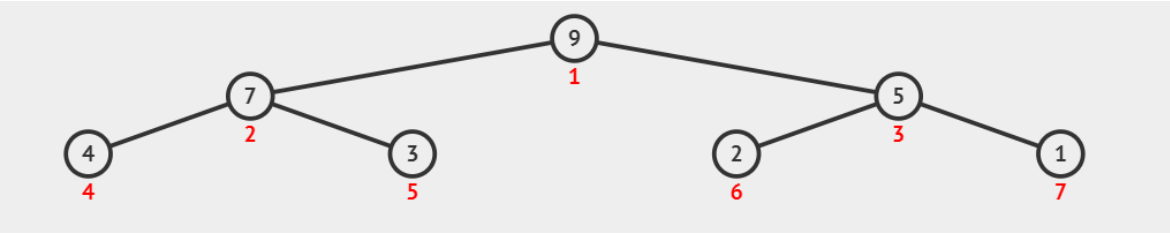
- 반환값: 없음

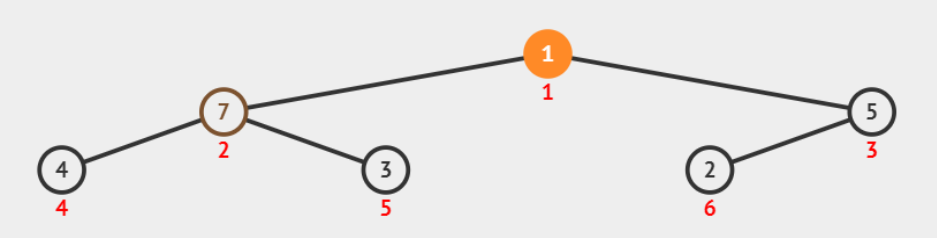
- 내용: 힙 내 위치 i에 저장된 키를 크기에 맞는 위치로 하향 이동

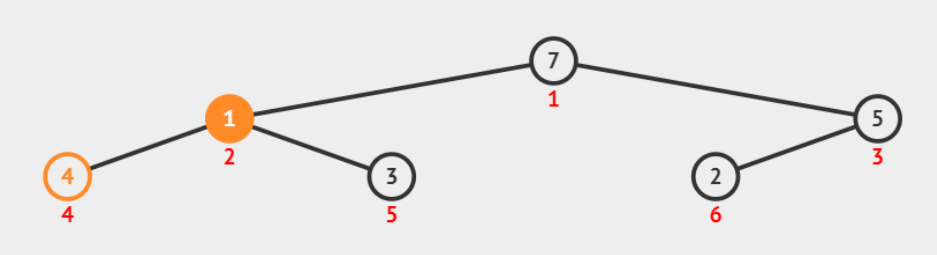
- 시간 성능: O(log n)

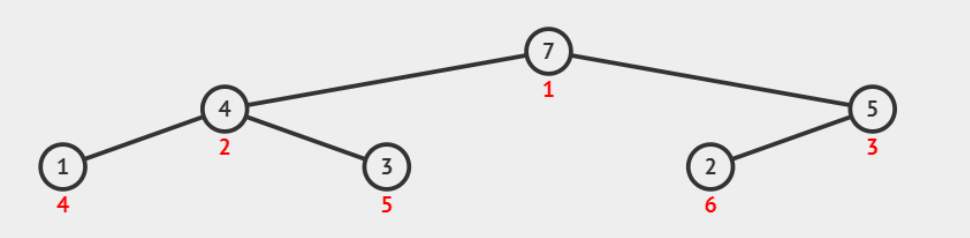
- 재귀적인 알고리즘

삽입식 힙 구현









downHeap()

인자: 배열 인덱스 i

- 반환값: 없음
- 내용: 힙 내 위치 i에 저장된 키를 크기에 맞는 위치로 하향 이동

- 시간 성능: O(log n)

수도코드를 살펴보면 마찬가지로 재귀적인 알고리즘이다.

삽입식 힙 구현

printHeap()

```
void printHeap() {
    for (int i = 1; i < n+1; i++) {
        printf(" %d", Heap[i]);
    }
    printf("₩n");
}
```

인자 i에 1부터 n+1까지 대입시키며
전체 순회해
값을 프린트함

일반 힙 정렬 (삽입식 힙 정렬)

힙 정렬

(4) 구현 방법

```
Alg heapSort(L)
   input list L
   output sorted list L

1. H <- empty heap # 비어있는 힙 생성
2. while (!L.isEmpty())     #리스트가 비워질때까지 반복
      k <- L.removeFirst() # 리스트의 첫번째 원소를 힙에 삽입( 즉 while 안에서 n번 반복하게 됨)
      H.insertItem(k) # insertItem은 O(log n) * n번 반복 ==> O ( n * log n)
3. while (!H.isEmpty())
      k <- H.removeMin() #최소힙에서 루트제거
      L.addLast(k) # 최솟값이 리스트에 삽입

4. return

# 결과 리스트가 오름차순으로 정렬된다
```

- 성능: O(n * log n)
- 힙 정렬은 힙을 이용한 정렬 방식으로,
  힙에 넣었다가 꺼내는 것이 전부임

페이즈1:
삽입식 힙 생성

페이즈2:
힙 정렬

k <- H.removeMin()
# 최소 힙에서 루트 제거
L.addLast(k)
 # 최솟값 리스트에 삽입

힙 정렬 개선

힙 정렬

(5) 개선 방향

힙 정렬 알고리즘은 두 가지 관점에서 개선 방향이 존재함

1. 처음 입력으로 주어진 리스트 L 외에 새로운 공간인 H를 사용했음

H를 사용하지 않고 **제자리 힙 정렬**을 수행할 수 있다면 **메모리(공간) 사용을 줄일 수 있을 것**

(**원래 주어진 제 1공간인 L 만 가지고**도 정렬이 가능)

2. **상향식 힙 생성**은 O(n)의 시간에서 동작하기 때문에 힙 정렬의 속도를 높혀 줄 수 있음

(1번 페이즈의 힙 생성 과정에서 **삽입식 힙 생성이 아닌 상향식 힙 생성으로 진행하면**, **속도가 향상**)

재귀 / 비재귀적 방법으로 구현 가능

힙 정렬 개선

힙 정렬

(6) 상향식 힙 정렬

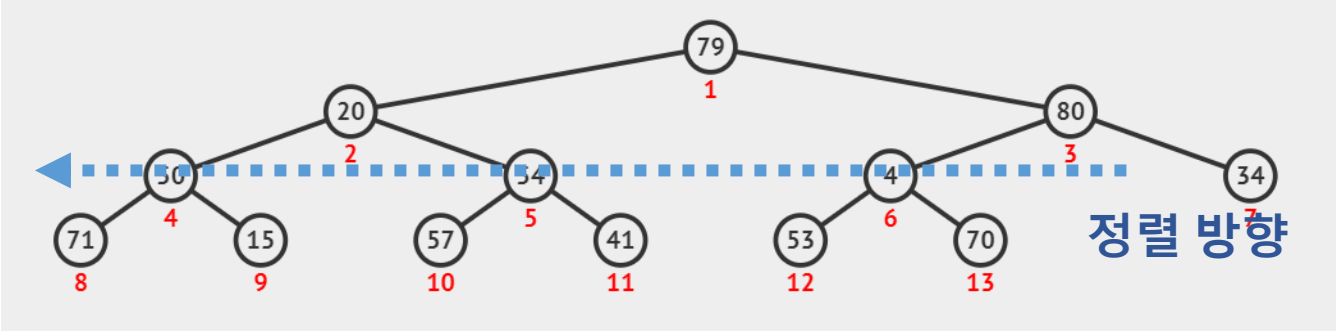
```
void buildHeap() {

    for (int i = n / 2; i >= 1; i--) {
        downHeap(i);
    }

}
```

1. BuildHeap(i)

- 인자: 정수 i (부분 힙의 루트 인덱스)
- 반환값: 없음

- 내용: **비재귀 방식으로** 상향식 힙 생성

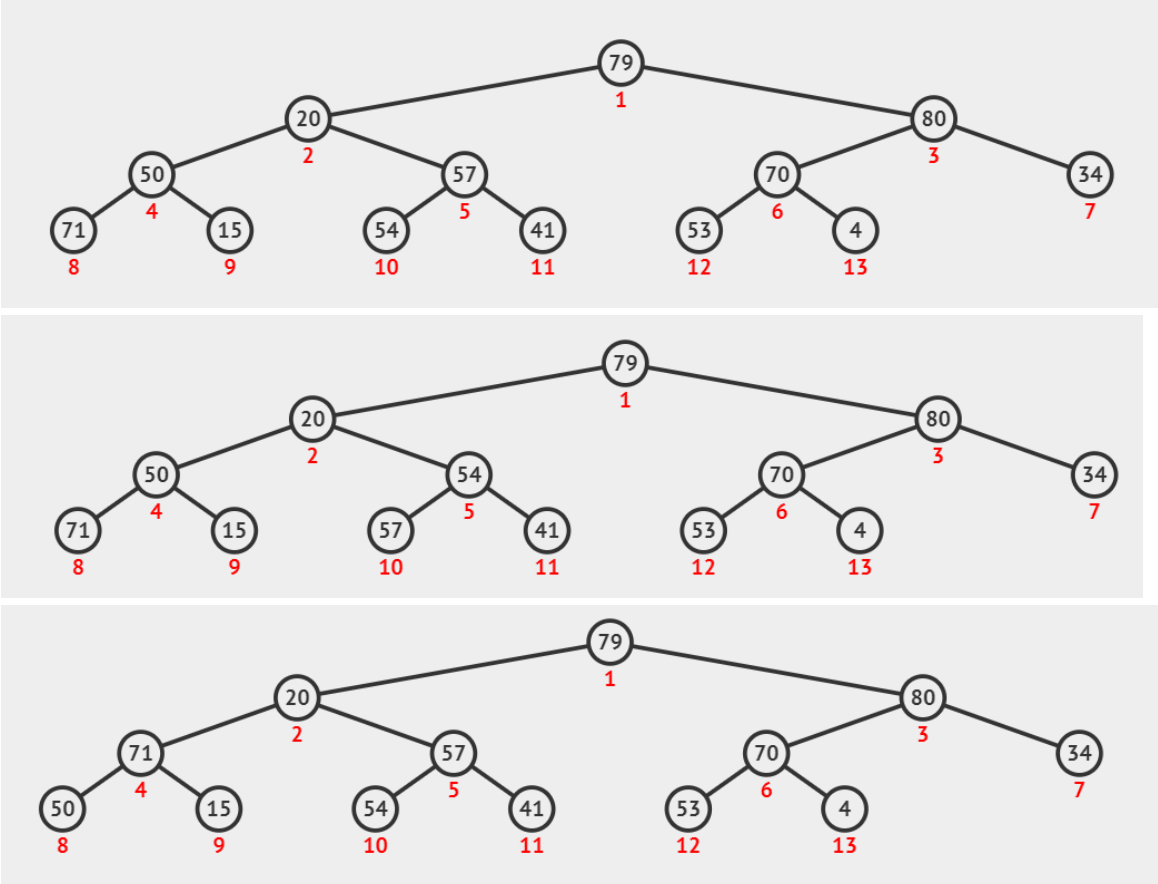
- 시간 성능: O(n)



힙 정렬 개선

힙 정렬

(6) 상향식 힙 정렬



1. BuildHeap(i)

- 인자: 정수 i (부분 힙의 루트 인덱스)
- 반환값: 없음

- 내용: **비재귀 방식으로** 상향식 힙 생성

- 시간 성능: O(n)

힙 정렬 개선

힙 정렬

(6) 상향식 힙 정렬

알고리즘 설계 팁:

```
Alg rBuildHeap(i)          {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
```

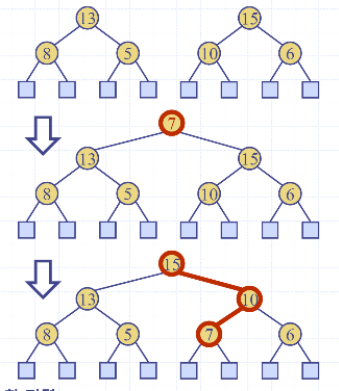
재귀 호출이 반환함에 따라 트리 위쪽으로 진행하기 때문에
상향식이라 명명함

2. rBuildHeap(i)

- 인자: 정수 i (부분 힙의 루트 인덱스)
- 반환값: 없음

- 내용: **재귀 방식으로** 상향식 힙 생성,
부모 노드와
두 개의 하위 힙을 부트리로 하는
새 힙을 생성해, downheap 수행

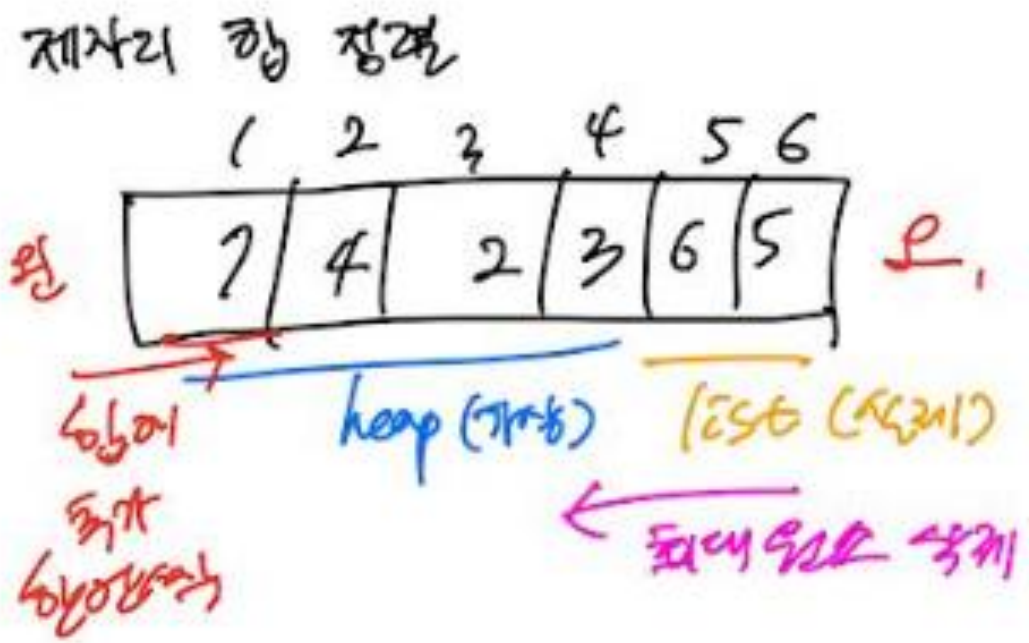
- 시간 성능: O(n)

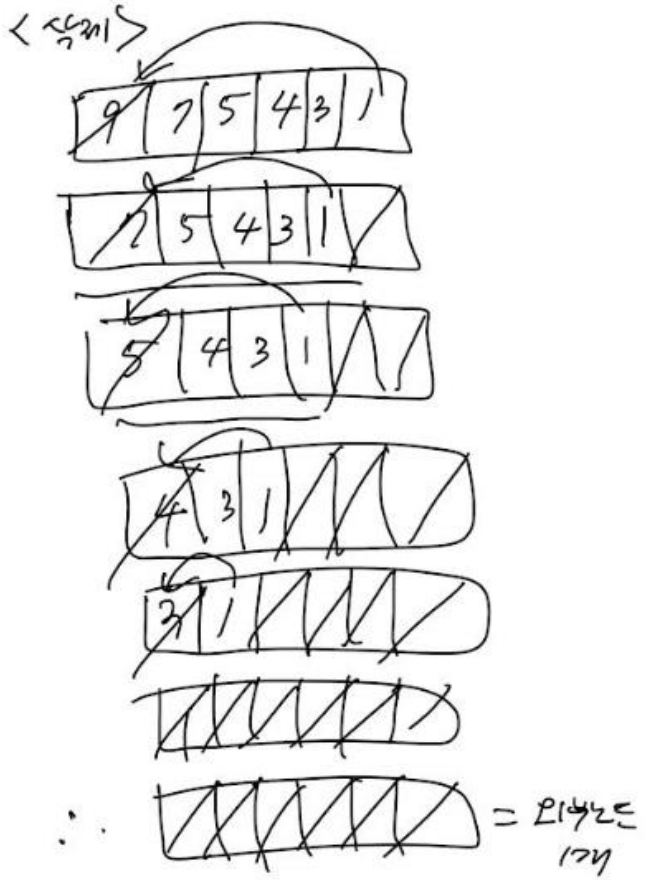


힙 정렬 개선

힙 정렬

(7) 제자리 힙 정렬





6조 최고은 / 이재윤

1주차
Q&A

참고 블로그: https://man-25-1.tistory.com/