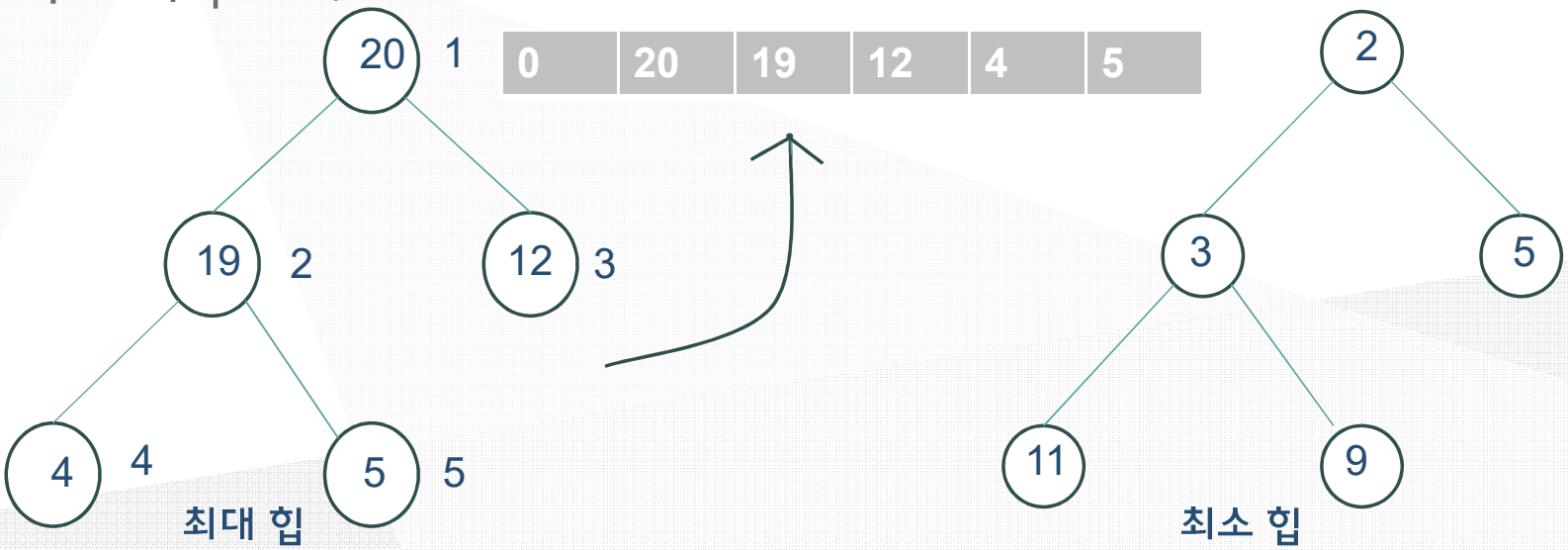
힙소트, 카운팅 소트

SSAFY 9기 구미 6반 김도연

힙 트리?

- 완전 이진트리, 우선순위 큐를 구현하는데 사용된다, 배열 형태로 구현
- 최대 힙: 부모 노드의 키값이 자식 노드의 키 값보다 크거나 같음
- 최소 힙: 부모 노드의 키값이 자식 노드의 키 값보다 작거나 같음
- 노드관계 (0번째 배열을 사용하지 않는 경우, 0번째 배열을 사용하면 각각에 +1)
  - 왼쪽자식: 부모index*2
  - 오른쪽자식: 부모index*2+1
  - 부모: 자식index/2



최대 힙 트리의 구현(insert)

전체 코드는 깃허브에 따로 업로드하겠습니다...

```
public void insert(int x) {
    if(pushPos>=nodes.length) {//배열 공간이 모자란 경우 공간을 늘린다
        resize();
    }
    nodes[pushPos]=x;
    int nowIdx=pushPos;
    while(true) {
        int parentIdx=getParent(nowIdx);
        if(parentIdx==0)
            break;
        if(nodes[nowIdx]>nodes[parentIdx]) {
            int temp=nodes[nowIdx];
            nodes[nowIdx]=nodes[parentIdx];
            nodes[parentIdx]=temp;
            nowIdx=parentIdx;
        }else {
            break;
        }
    }
    pushPos++;
}
```

가장 끝에 값을 추가
타고 올라가면서
힙구조를 맞춘다

현재 노드의 부모
가 0인덱스이면
현재 노드가 루트

현재 노드가
부모보다 크면
현재 노드를 올린다

부모가 현재 노드보다
크거나 같으면 올리기
종료

최대 힙의 구현(pop)

```
public int pop(){
    if(pushPos==1)
        return -1;
    int root=nodes[1];//힙트리의 삭제는 루트에서 한다
    nodes[1]=nodes[pushPos-1];//가장 마지막 노드를 루트로 올린다.
    pushPos--;

    int nowIdx=1;
    while(true) {
        if(nowIdx*2>=pushPos) {//현재 인덱스의 자식이 없는 경우
            break;
        }else {
            int leftChild=getLeftChild(nowIdx);
            int rightChild=getRightChild(nowIdx);
            int maxChild=nodes[leftChild];
            int maxPos=leftChild;
            if(rightChild < pushPos && nodes[rightChild]>maxChild) {//오른쪽 자식 존재하고 왼쪽 자식보다 오른쪽이 더 클때
                maxChild=nodes[rightChild];
                maxPos=rightChild;
            }
            if(nodes[nowIdx]>maxChild) {//부모가 더 크므로 교환 종료
                break;
            }else {
                int temp=nodes[nowIdx];
                nodes[nowIdx]=maxChild;
                nodes[maxPos]=temp;
                nowIdx=maxPos;
            }
        }
    }
    return root;
}
```

루트 노드 값 삭제

가장 마지막 노드를 루트로 올리고 루트 노드를 아래로 내리면서 힙구조를 맞춘다

노드 삽입 위치를 -1 한다

현재 인덱스 자식이 없는 경우 더이상 내릴곳이 없으므로 중지

왼쪽, 오른쪽 자식 둘 중 큰것을 선택

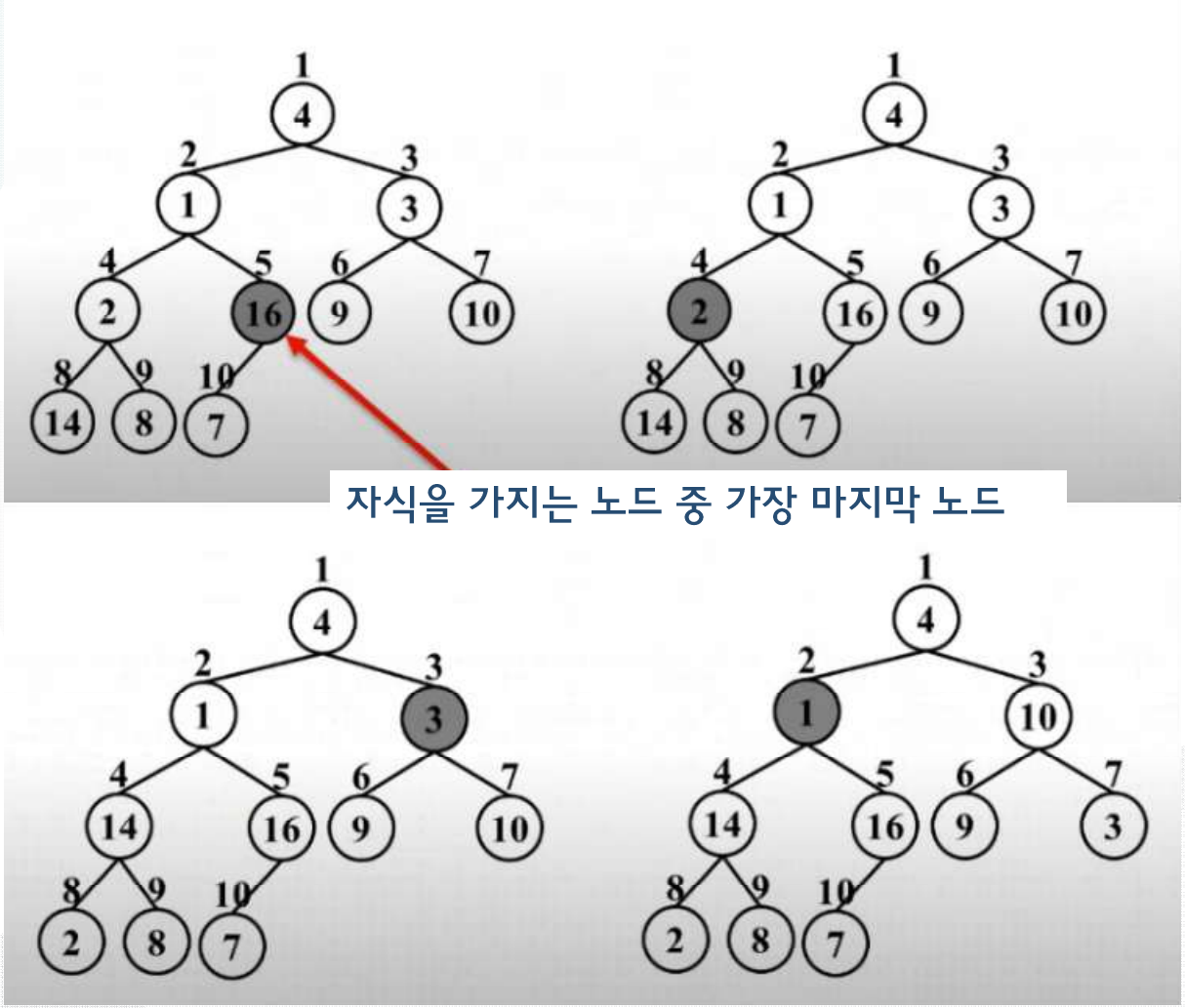
왼쪽, 오른쪽 자식 둘 중 더 큰것과 현재 노드 교환

힙 소트?

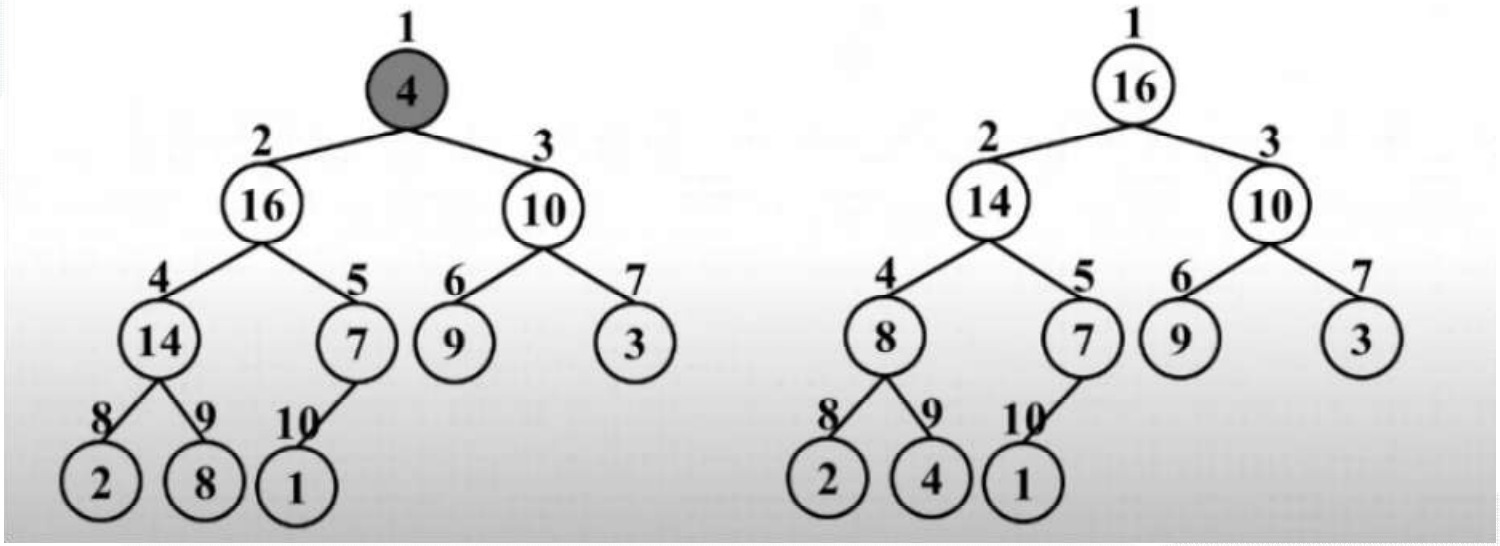
- 최대 힙 혹은 최소 힙 트리 구조를 이용해서 정렬한다
- 수행시간 O(nlogn)
- Heapify: 노드가 입력으로 주어졌을때 heap 특성이 유지되도록 바꾸는 연산
- 최대 힙을 만들면 오름차순 정렬, 최소 힙을 만들면 내림차순 정렬
- 이미 존재하는 배열에 대해 힙 정렬을 하려면 아래와같은 절차를 거친다
  - 1. 입력으로 받은 배열로 최대 혹은 최소 힙 구성 Heapify
  - 2. 루트 값을 하나씩 맨 뒤의 값과 교환, 맨 뒤 제외하고 Heapify -> 가장 우선순위 높은 값이 맨뒤로 -> 오름차순정렬

힙 소트 예시 (입력배열 힙으로 만들기) -1

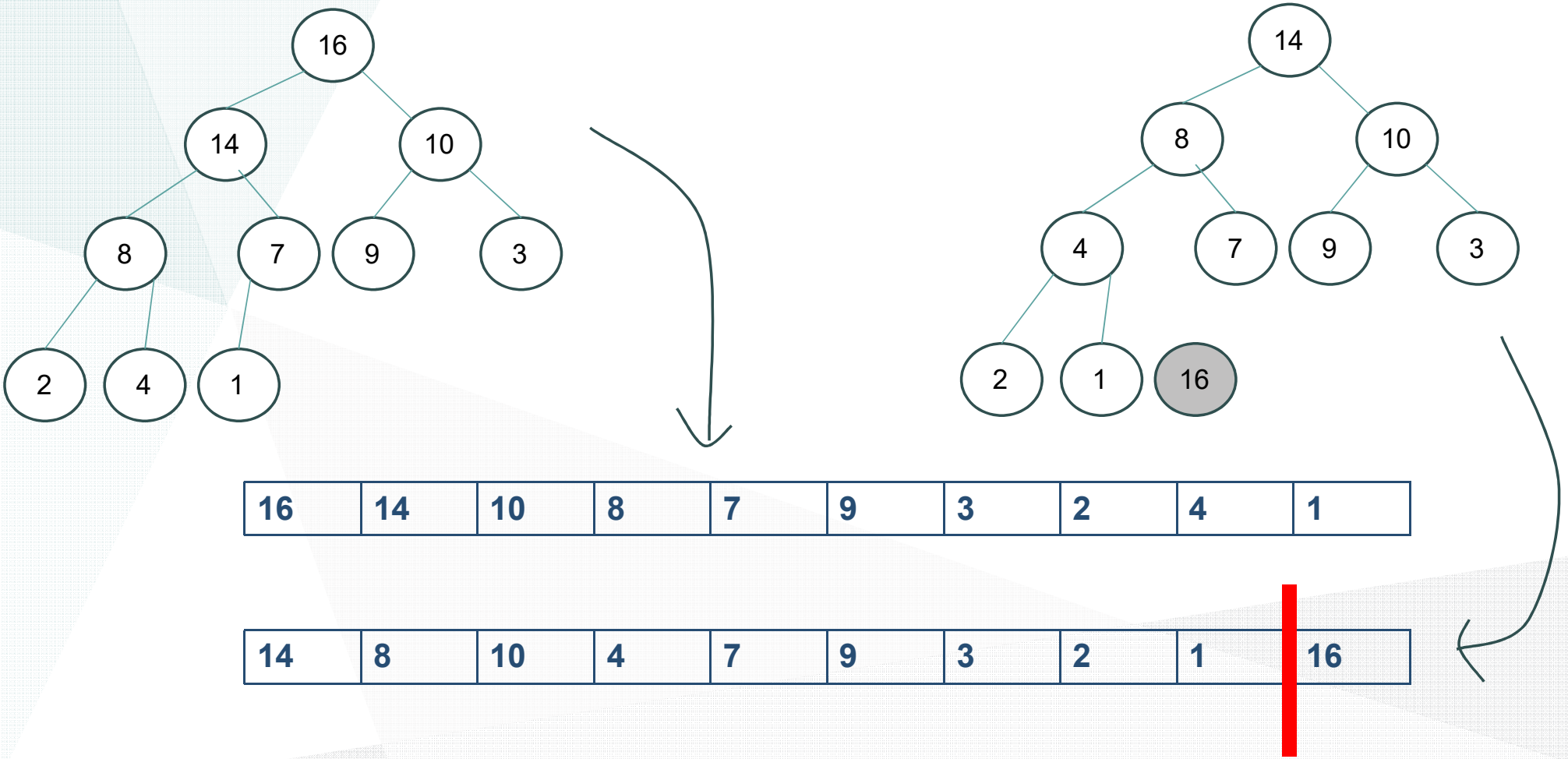
가장 작은 서브트리를 힙 구조에 맞게 변경하고 점차 위로 올라가면서 트리를 힙에 맞게 변경함



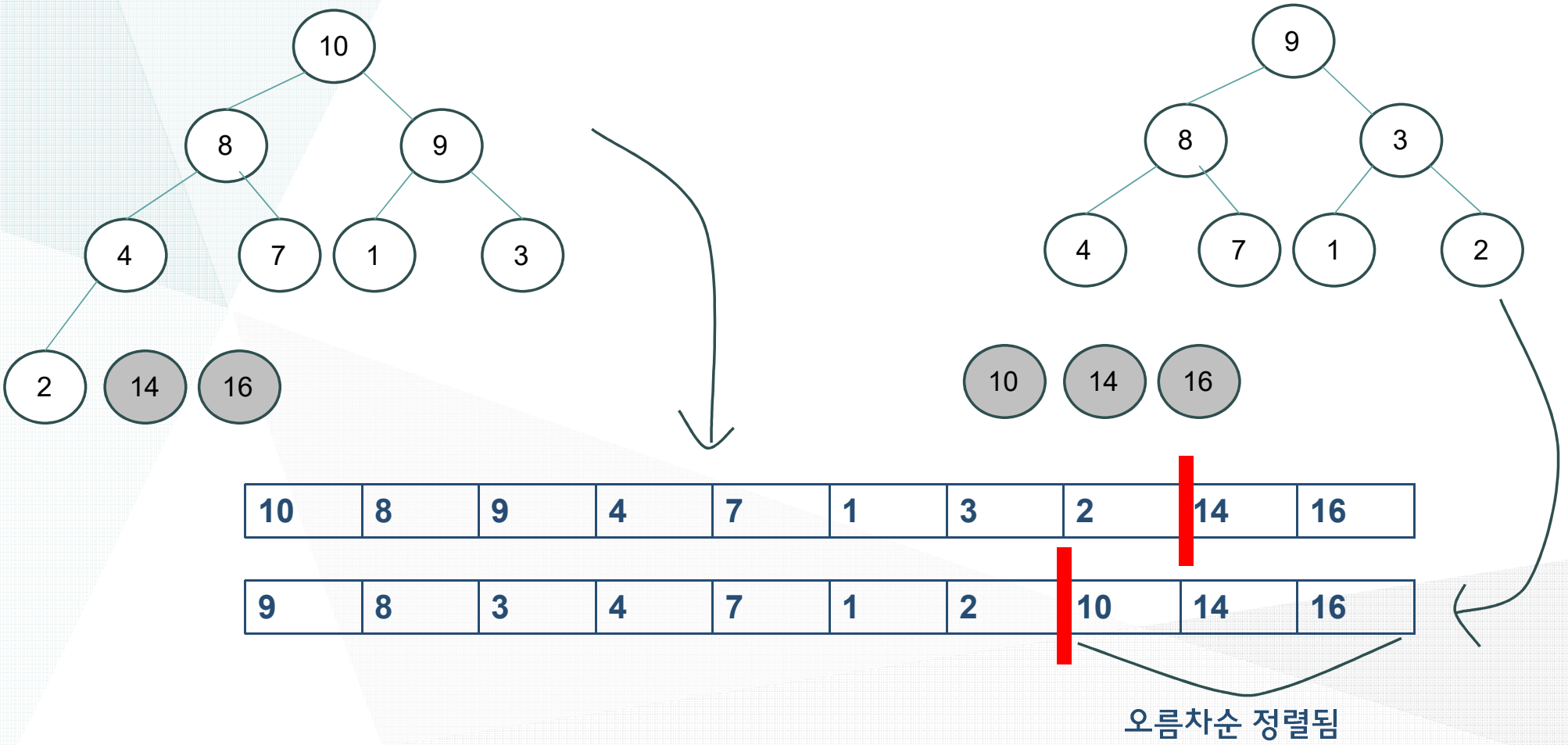
힙 소트 예시 (입력배열 힙으로 만들기) -2



힙 소트 예시 (루트 값 빼오기) -3



힙 소트 예시 (루트 값 빼오기) -4



=>1번 인덱스 까지 반복!

힙 소트 구현(JAVA) -1

```java
public static void heapSort(int[] array, int n) {
    for(int i=(n/2)-1 ; i>=0; i--) {
        //n/2-1 부터 시작하는 이유는 이 위치가 자식을 가지는 마지막 노드이기 때문임
        //가장 작은 서브트리부터 시작해서 값을 비교해 서브트리들을 맥스힙으로 만들어주면서 점차적으로 올라가는 형태이므로
        //자식을 가지는 마지막 노드부터 맥스힙으로 변경을 시작하는것
        heapify(array, n, i);
    }
    // Root에 위치한 최대값을 마지막 노드와 바꿔가며 Heap  재구성
    // Heap의 크기를 줄여가며 값이 큰 원소를 차례로 가져옵니다.
    for (int i = n - 1; i > 0; i--) {
        int temp= array[0];
        array[0]=array[i];
        array[i]=temp;
        heapify(array, i, 0);
    }
}
```

입력 배열 힙으로 구성

루트값 빼오기

루트값과 맨 뒤 값 교환

루트부터 아래로 내린다

힙 소트 구현(JAVA) -2

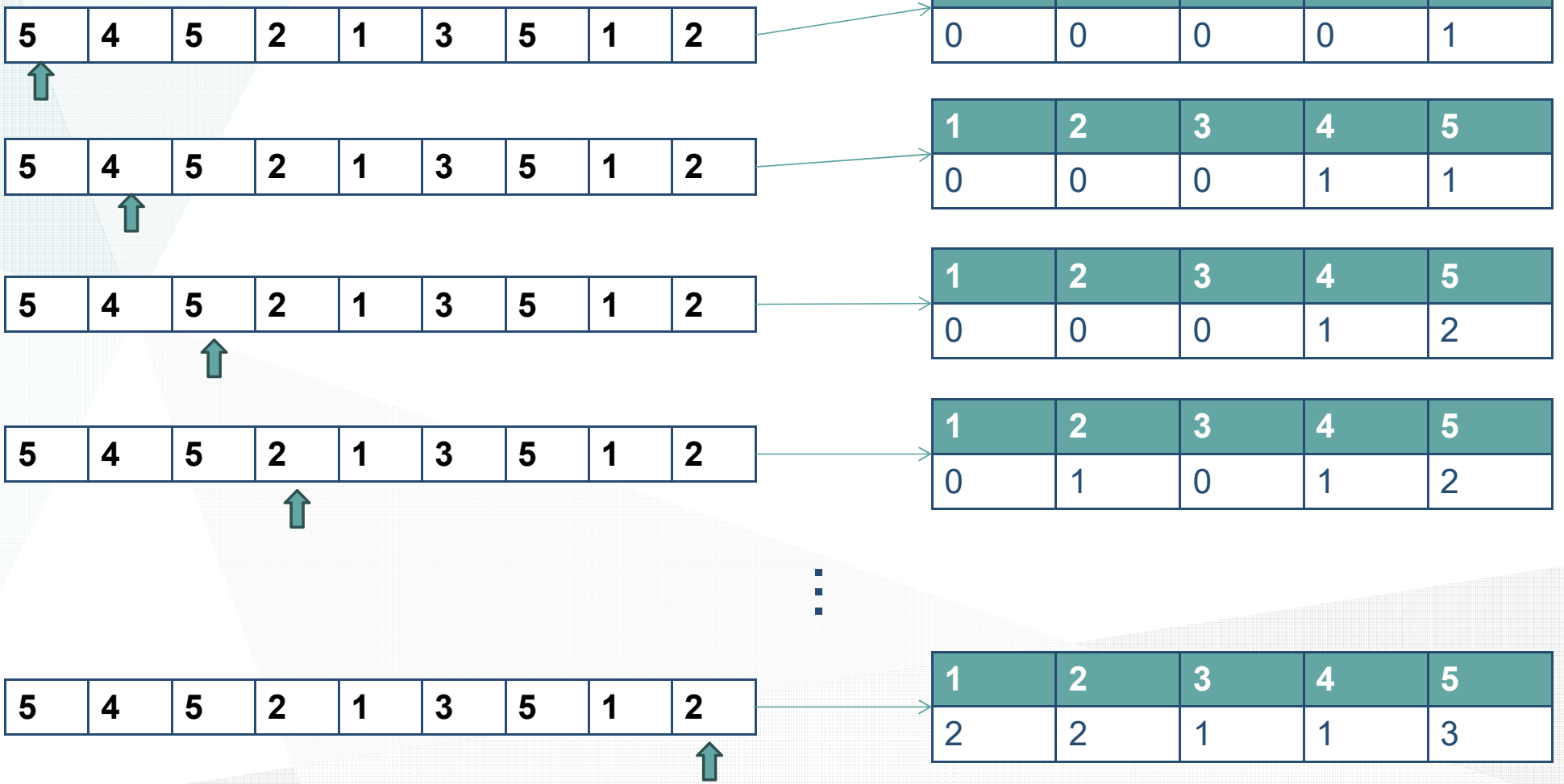
```java
public static void heapify(int array[], int n, int i) {
    //i번 인덱스 밑으로 존재하는 트리를 맥스힙으로 만든다
    int parentIdx=i;
    while(true) {
        if(parentIdx*2+1>=n) {//현재 인덱스 자식 없는 경우
            break;
        }
        int leftChildNode = parentIdx * 2 + 1;
        int rightChildNode = parentIdx * 2 + 2;
        int maxChild=array[leftChildNode];
        int maxPos=leftChildNode;
        if(rightChildNode<n && array[rightChildNode]>maxChild) {
            //오른쪽 자식이 존재하고 오른쪽이 왼쪽보다 클때
            maxChild=array[rightChildNode];//더 큰 자식을 선택한다
            maxPos=rightChildNode;
        }

        if(array[parentIdx]>=maxChild) {//큰 자식보다 부모가 크면 변경 종료
            break;
        }else {//큰 자식과 부모를 바꾸고 아래로 내려간다
            int temp=array[parentIdx];
            array[parentIdx]=maxChild;
            array[maxPos]=temp;
            parentIdx=maxPos;
        }
    }
}
```

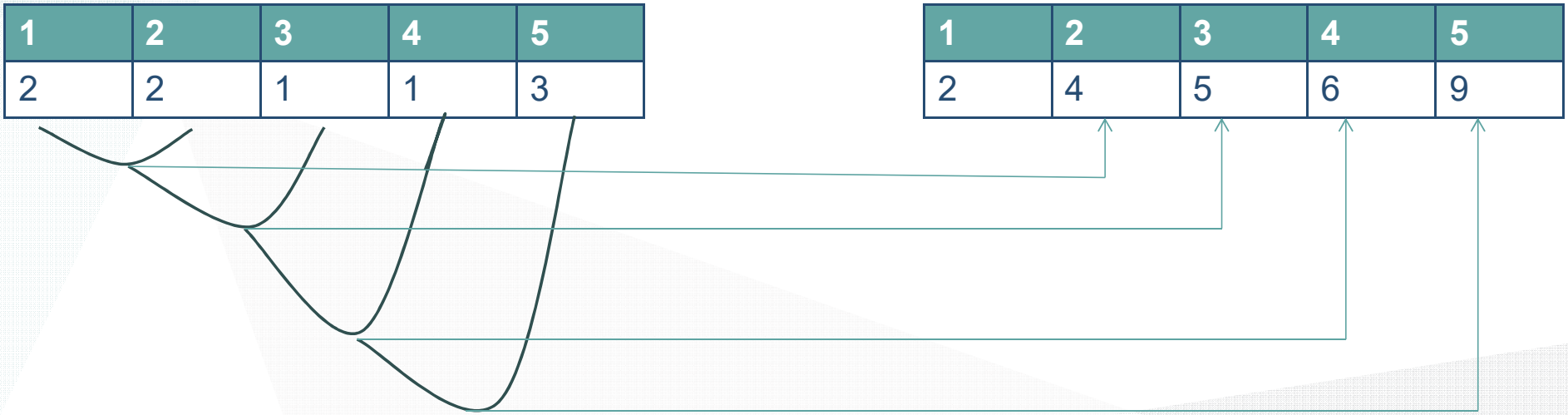
카운팅 소트?

- 계수정렬
- 값을 비교하지않고 값들의 개수를 세어서 정렬한다
- 데이터의 크기 범위가 제한된 경우에 사용 가능하다
- 데이터가 양의 정수일때만 사용 가능하다
- 시간 복잡도: O(n)
- 정렬 할 배열내 최댓값의 +1 크기의 새로운 배열을 생성해야하므로 수의 범위가 클 경우 메모리 낭비가 심하다
- 카운팅 소트는 아래와 같은 절차를 거친다
  - 1. 카운팅
  - 2. 누적합으로 바꾸기
  - 3. 결과 배열 만들기

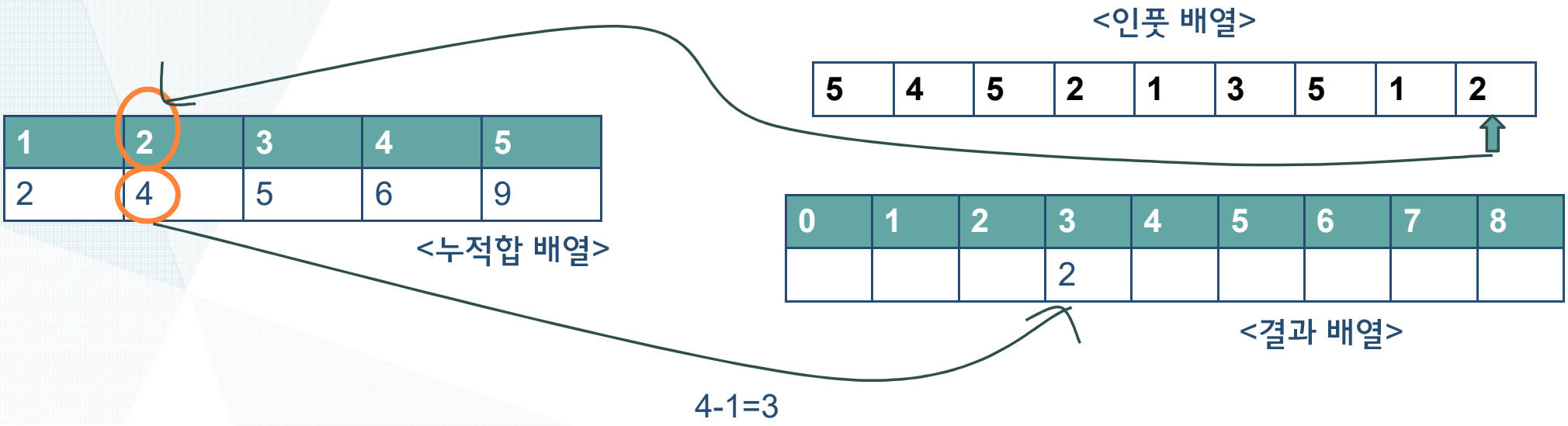
카운팅 소트 예시 (카운팅) -1

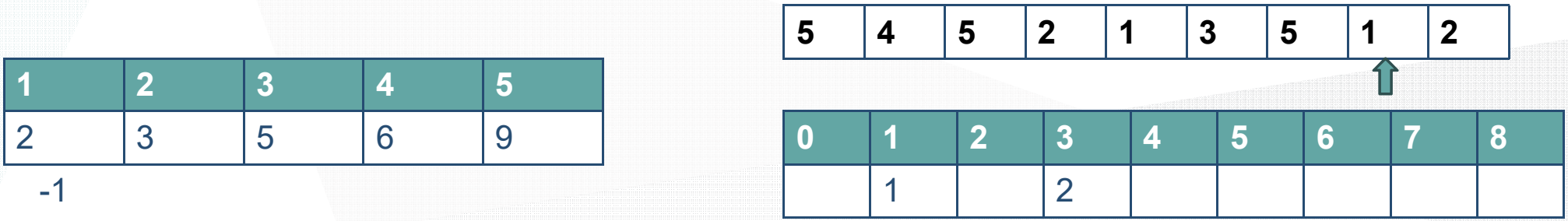


카운팅 소트 예시 (누적합으로 변환) -2



카운팅 소트 예시 (결과 배열 만들기) -3





카운팅 소트 예시 (결과 배열 만들기) -4

1	2	3	4	5
1	3	5	6	9

-1

1	2	3	4	5
0	2	4	5	7

1	2	3	4	5
0	2	4	5	6

5	4	5	2	1	3	5	1	2

0	1	2	3	4	5	6	7	8
	1		2					5

5	4	5	2	1	3	5	1	2

0	1	2	3	4	5	6	7	8
1	1	2	2	3	4		5	5

5	4	5	2	1	3	5	1	2

0	1	2	3	4	5	6	7	8
1	1	2	2	3	4	5	5	5

카운팅 소트 구현(JAVA)

```java
public static void main(String[] args) {
    int[] inputData= {5, 4, 5, 2, 1, 3, 5, 1, 2};
    for(int i=0;i<inputData.length;i++) {//인풋에서 가장 큰 값 구하기
        if(max<inputData[i])
            max=inputData[i];
    }
    countArray=new int[max+1];
    sortedData=new int[inputData.length];

    for(int i=0;i<inputData.length;i++) {//카운팅
        countArray[inputData[i]]++;
    }
    for(int i=1;i<=max;i++) {//누적합 만들기
        countArray[i]+=countArray[i-1];
    }
    for(int i=inputData.length-1; i>=0; i--) {//결과 배열 만들기
        int value=inputData[i];
        countArray[value]--;
        sortedData[countArray[value]]=value;
    }

    for(int i=0;i<sortedData.length;i++)
        System.out.print(sortedData[i]+" ");
```

참고자료

◆ 힙 소트

- go-coding.tistory.com/25 (코드참고)
- st-lab.tistory.com/205 (그림참고)
- velog.io/@redgem92/자료구조-힙-트리Heap-Tree (코드 참고)
- https://mjmjmj98.tistory.com/154
- https://youtu.be/ehNVf2Bcm2Q(예시참고)

◆ 카운팅 소트

- st-lab.tistory.com/104 (코드 참고)
- velog.io/@luvlik207/알고리즘-정리-계수-정렬Counting-Sort (예시 참고)