

# Data Structure #10

Quick Sort

2020년 1학기

# Intro.

- 실습주제 소개
  - quick sort, merge sort, heap sort
- 실습수업 문제
  - quick sort

CSLAB

# Sorting

- 정렬이란?

- 순서 없이 배열되어 있는 자료들을 특정 기준에 따라 순서대로 재배치하는 행위
  - 순서의 예 : 내림차순, 오름차순, 알파벳 순서 등

CSLAB

# Divide and Conquer

- 분할정복법(divide and conquer)은 문제를 작은 2개의 문제로 분리하고 각각을 해결한 다음, 결과를 모아서 원래의 문제를 해결하는 전략이다.
- 분리된 문제가 아직도 해결하기 어렵다면, 즉 충분히 작지 않다면 분할정복방법을 다시 적용한다. 이는 재귀호출을 이용하여 구현된다.

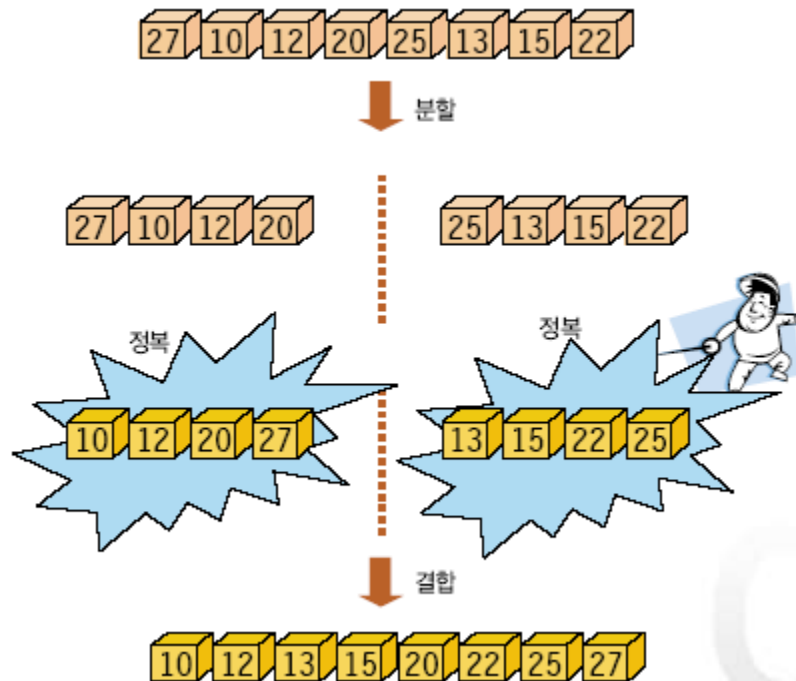
1. 분할(Divide): 배열을 같은 크기의 2개의 부분 배열로 분할한다.
2. 정복(Conquer): 부분배열을 정렬한다. 부분배열의 크기가 충분히 작지 않으면 재귀호출을 이용하여 다시 분할정복기법을 적용한다.
3. 결합(Combine): 정렬된 부분배열을 하나의 배열에 통합한다.

입력파일: 27 10 12 20 25 13 15 22

1. 분할(Divide): 배열을 27 10 12 20 과 25 13 15 22의 2개의 부분배열로 분리
2. 정복(Conquer): 부분배열을 정렬하여 10 12 20 27 과 13 15 22 25를 얻는다.
3. 결합(Combine): 부분배열을 통합하여 10 12 13 15 20 22 25 27을 얻는다.

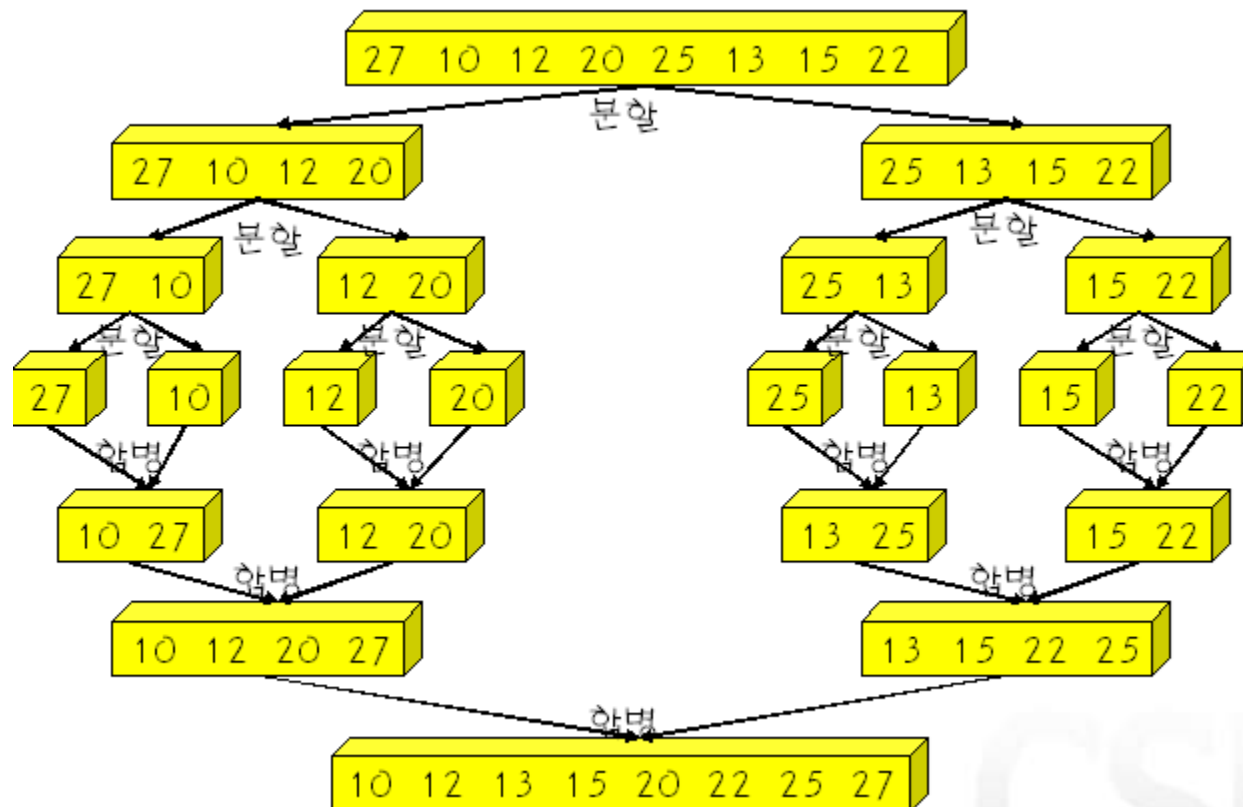
# Merge sort

- 합병정렬은 리스트를 두 개로 나누어, 각각을 정렬한 다음, 다시 하나로 합치는 방법이다.
- 합병정렬은 분할정복법을 적용함.



CSLAB

# Merge sort의 과정



CSLAB

# Merge sort

- mergeSort

```
void mergeSort(int data[], int p, int r)
{
    int q;
    if (p < r)
    {
        q = (p + r) / 2;
        mergeSort(data, p, q);
        mergeSort(data, q + 1, r);
        merge(data, p, q, r);
    }
}
```

CSLAB

# Merge sort

- merge

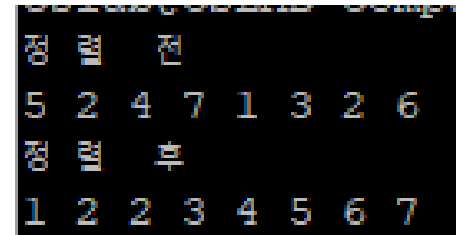
```
void merge(int data[], int p, int q, int r)
{
    int i = p, j = q + 1, k = p;
    int tmp[8];
    while (i <= q && j <= r)
    {
        if (data[i] <= data[j])
            tmp[k++] = data[i++];
        else
            tmp[k++] = data[j++];
    }
    while (i <= q)
        tmp[k++] = data[i++];
    while (j <= r)
        tmp[k++] = data[j++];
    for (int a = p; a <= r; a++)
        data[a] = tmp[a];
}
```



# Merge sort

- main

```
int main()
{
    int data[8] = {5, 2, 4, 7, 1, 3, 2, 6};
    int i;
    printf("정렬 전\n");
    for (i = 0; i < 8; i++)
    {
        printf("%d ", data[i]);
    }
    mergeSort(data, 0, 7);
    printf("\n정렬 후\n");
    for (i = 0; i < 8; i++)
    {
        printf("%d ", data[i]);
    }
    return 0;
}
```



```
정렬 전
5 2 4 7 1 3 2 6
정렬 후
1 2 2 3 4 5 6 7
```

CSLAB

# Quick sort

## • 퀵 정렬(quick sort)이란?

- 정렬할 전체 데이터에 대해서 정렬을 수행하지 않고 기준 값을 중심으로 왼쪽 부분집합과 오른쪽 부분집합으로 분할
  - 왼쪽 부분집합에는 기준 값보다 작은 데이터를 이동시키고 오른쪽 부분 집합에는 기준 값보다 큰 데이터를 이동
  - 이 때 사용하는 기준 값을 pivot이라고 하는데 일반적으로 리스트의 가운데 데이터를 pivot으로 지정
- 퀵 정렬 동작
  - ① :왼쪽 끝에서 오른쪽으로 움직이면서 크기를 비교하여 pivot보다 크거나 같은 데이터를 찾아 L로 표시
  - ② :오른쪽 끝에서 왼쪽으로 움직이면서 크기를 비교하여 pivot보다 작은 데이터를 찾아 R로 표시. 단, L과 R이 만나면 중지
  - ③ : ①과 ②에서 찾은 L과 R을 교환. 단, L과 R이 같은 데이터에 서 만나면 피벗과 R을 교환
  - ④ :확정된 pivot을 중심으로 양쪽의 부분집합을 대상으로 퀵 정렬을 반복

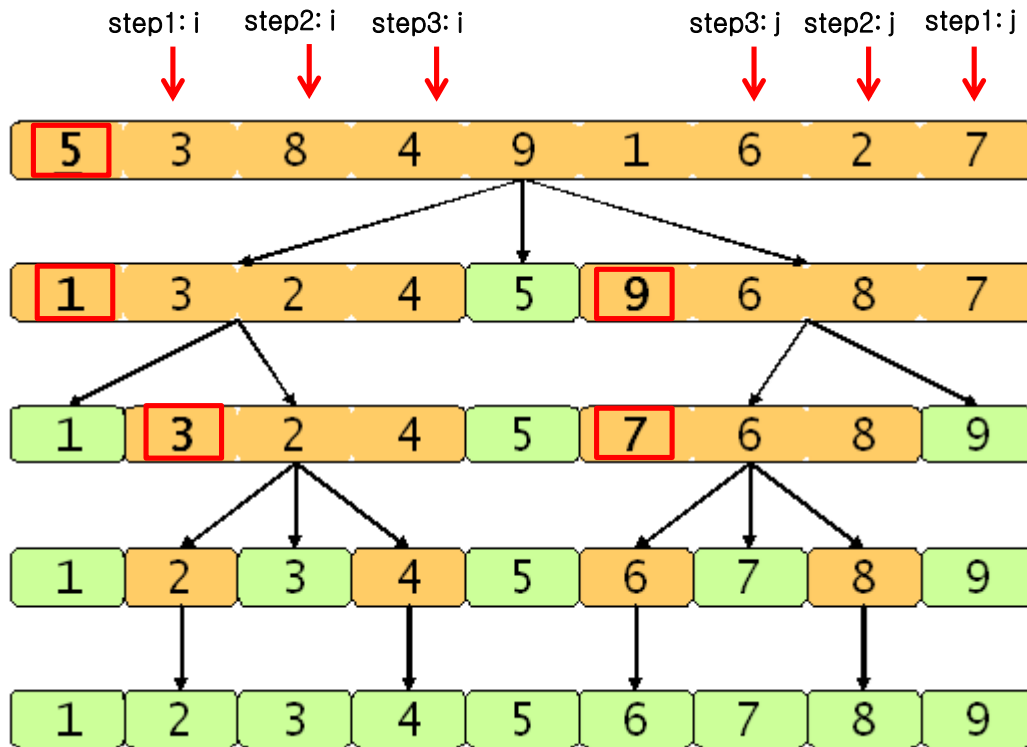
# Quick sort algorithm

```
1. void quick_sort(int list[], int left, int right)
2. {
3.     if(left<right){
4.         int q=partition(list, left, right);
5.         quick_sort(list, left, q-1);
6.         quick_sort(list, q+1, right);
7.     }
8. }
```

- line3: 정렬할 범위가 2개 이상의 데이터이면
- line4: partition 함수를 호출하여 pivot을 기준으로 2개의 리스트로 분할한다.  
partition 함수의 반환값은 pivot의 위치가 된다.
- line5: left에서 pivot 위치 바로 앞까지를 대상으로 순환호출한다.  
(pivot은 제외된다).
- line6: pivot 위치 바로 다음부터 right까지를 대상으로 순환호출한다.  
(pivot은 제외된다).

CSLAB

# Quick sort 전체과정



- pivot보다 작은 요소들은 모두 pivot의 왼쪽으로 옮겨지고 pivot보다 큰 요소들은 모두 pivot의 오른쪽으로 옮겨지도록 i와 j를 교환한다. i와 j의 인덱스가 서로 교차되면 멈춘다.

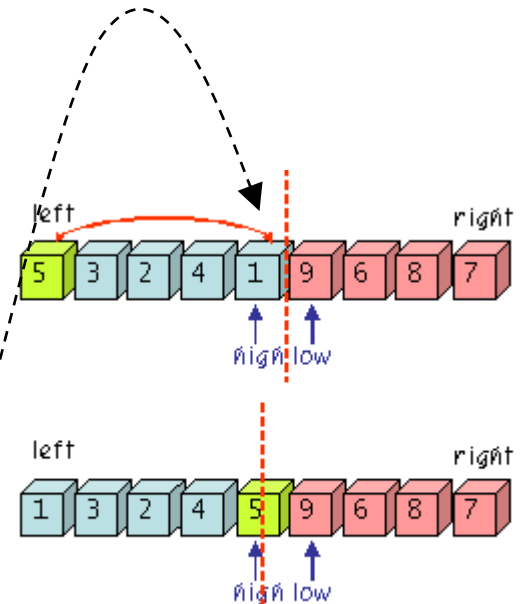
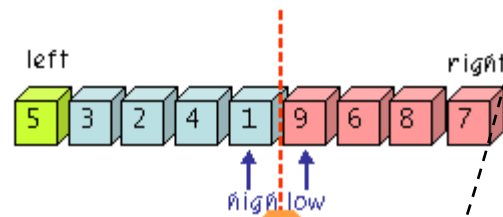
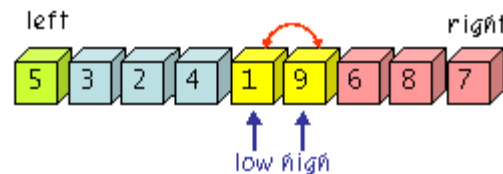
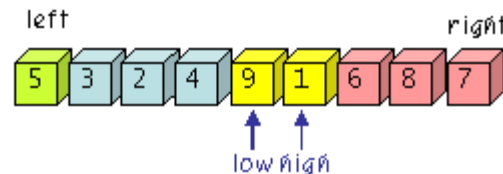
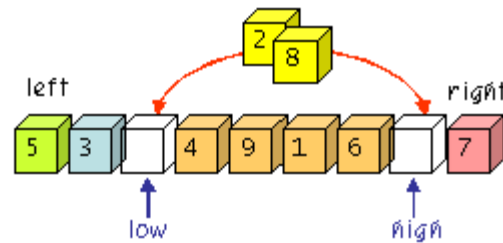
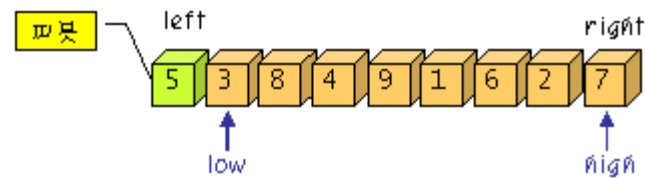
- 왼쪽 리스트와 오른쪽 리스트를 반복함.

- 빨간색 박스: pivot

CSLAB

# partition 함수

- pivot 가장 왼쪽 숫자라고 가정
- 두 개의 변수 low와 high를 사용한다.
- low는 pivot보다 작으면 통과, 크면 정지
- high는 pivot보다 크면 통과, 작으면 정지
- 정지된 위치의 숫자를 교환
- low와 high가 교차하면 종료



## Quick sort 구현 (개인 실습 #1)

- Print 함수

```
void print(int list[], int n)
{
    int i;
    for(i=0;i<n;i++)
        printf("%d ", list[i]);
    printf("\n");
}
```

CSLAB

# Quick sort 구현

- Partition 함수

```
int partition(int v[], int left, int right)
{
    int pivot, temp;
    int low, high;
    /* fill in the blank */
}
```

CSLAB

# Quick sort 구현

- Quick sort 함수

```
void quick_sort(int list[], int left, int right)
{
    /* 리스트에 2개 이상의 레코드가 있을 경우 */
    /* 왼쪽 부분리스트를 퀵정렬 */
    /* 오른쪽 부분리스트를 퀵정렬 */
}
```

CSLAB



# Quick sort 구현

- Main 함수

```
#include <stdio.h>

#define MAX_SIZE 100
int n=9;
int list[MAX_SIZE]={9, 8, 7, 6, 5, 4, 3, 2, 1};
#define SWAP(x, y, t) ( (t)=(x), (x)=(y), (y)=(t) )

int main()
{
    quick_sort(list, 0, 8);
}
```

Result:

1	8	7	6	5	4	3	2	9
1	8	7	6	5	4	3	2	9
1	2	7	6	5	4	3	8	9
1	2	7	6	5	4	3	8	9
1	2	3	6	5	4	7	8	9
1	2	3	6	5	4	7	8	9
1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9

CSLAB

# Heap sort

- **힙 정렬(heap sort)**

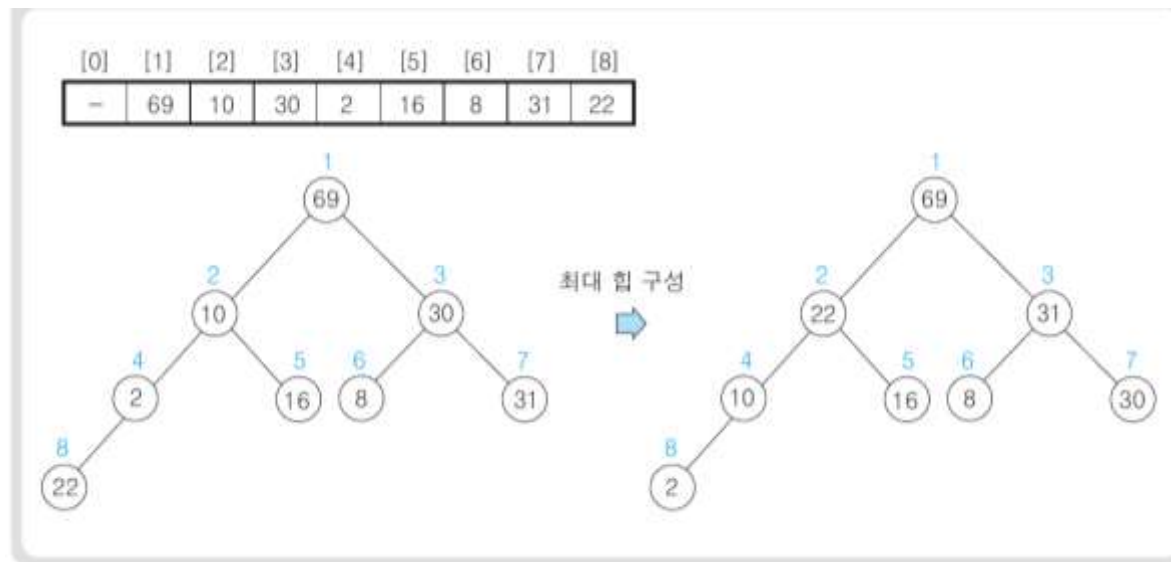
- 힙 자료구조를 이용한 정렬 방법
  - 최대 힙에서는 항상 최대 원소가 루트 노드임
  - 삭제 연산을 수행하면 항상 최대 원소를 삭제하여 리턴
  - 따라서 최대 힙에 대해 원소의 개수만큼 삭제 연산을 수행하면 정렬이 이루어진다.
- 힙 정렬 수행 방법
  - 정렬할 원소들을 최대 힙으로 구성한다.
  - 힙에 대해 삭제 연산을 수행하여 얻은 원소를 마지막 자리에 배치한다.
  - 나머지 원소에 대해 다시 최대 힙으로 재구성한다.
  - 위의 작업을 힙 원소가 모두 삭제될 때까지 반복하여 수행한다.

CSLAB

# Heap sort

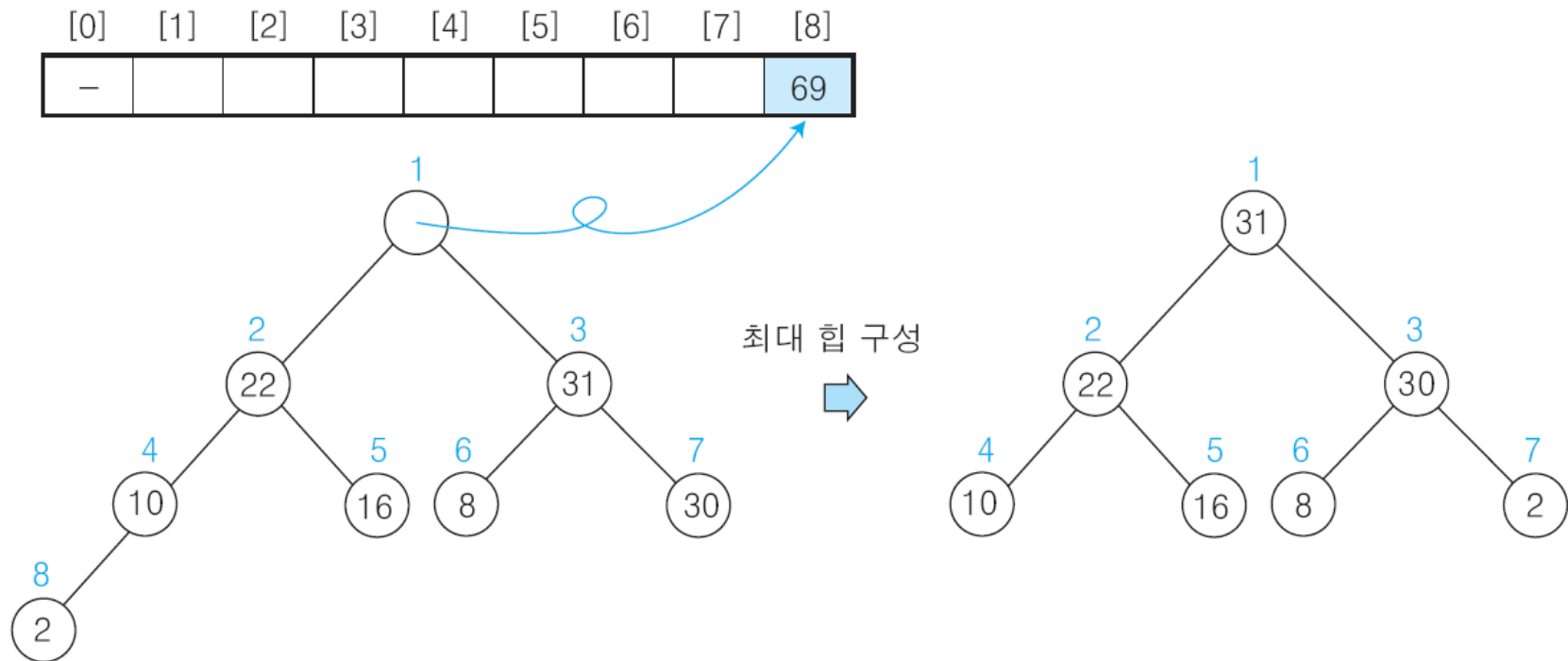
## • 힙 정렬 수행 과정

- 69, 10, 30, 2, 16, 8, 31, 22
- 초기 상태 : 정렬할 원소가 8개이므로 노드가 8개인 완전 이진 트리를 배열로 구현한 것으로 간주하면 된다.
- 완전 이진 트리를 최대 힙으로 구성한다.



# Heap sort

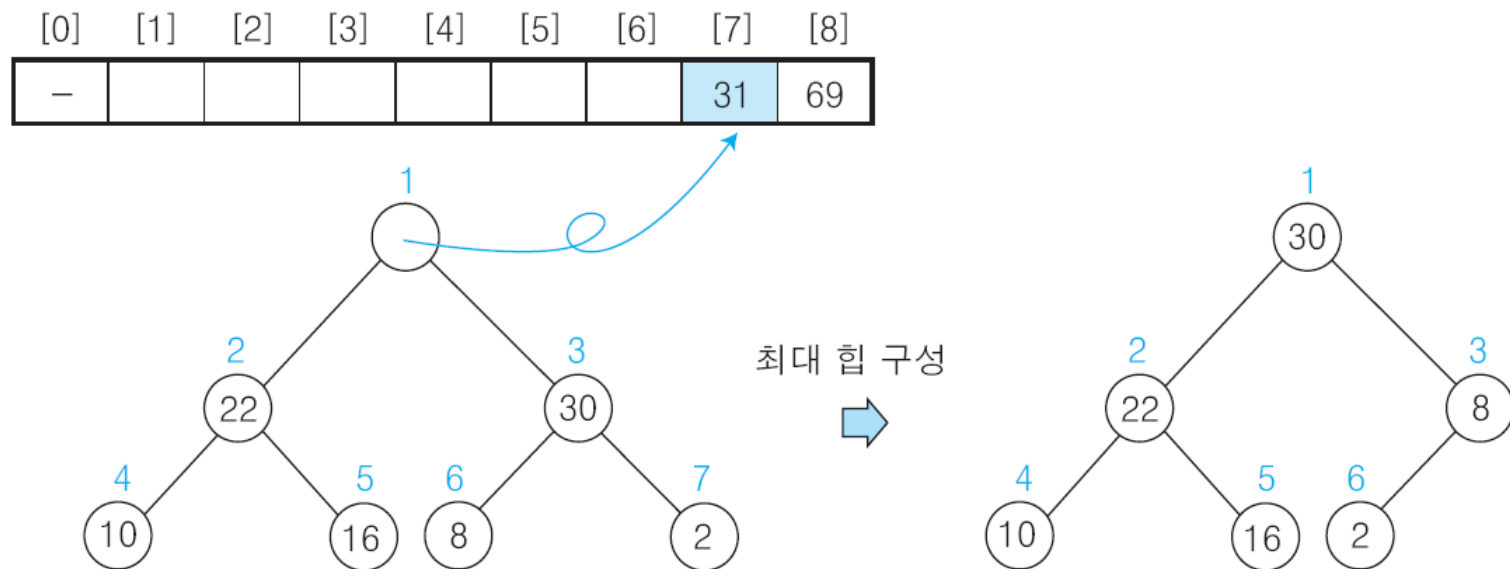
- ① 힙에 삭제 연산을 수행하여 루트 노드의 원소 69를 구해서 배열의 마지막 자리에 저장, 나머지 원소들에 대해서 최대 힙으로 재구성



CSLAB

# Heap sort

- ② 힙에 삭제 연산을 수행하여 루트 노드의 원소 31을 구해서 배열의 비어 있는 마지막 자리에 저장, 나머지 힙을 최대 힙으로 재구성



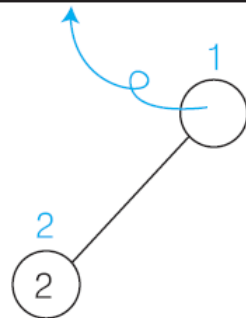
이 과정을 반복...

CSLAB

# Heap sort

- ⑦ 힙에 삭제 연산을 수행하여 루트 노드의 원소 8을 구해서 배열의 비어 있는 마지막 자리에 저장, 나머지 원소들에 대해서 최대 힙으로 재구성

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
-		8	10	16	22	30	31	69



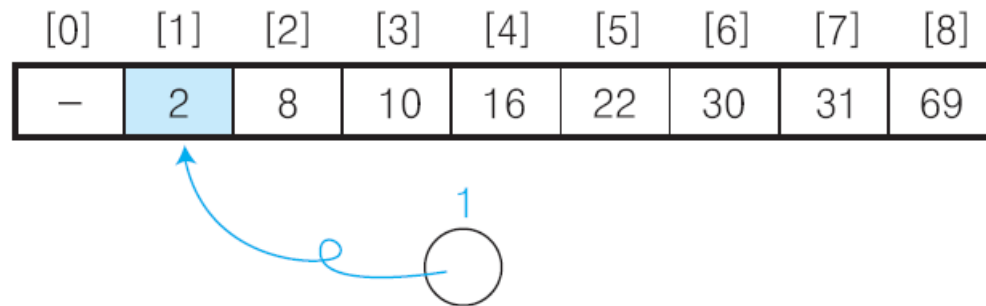
최대 힙 구성



CSLAB

# Heap sort

- ⑧ 힙에 삭제 연산을 수행하여 루트 노드의 원소 2를 구해서 배열의 비어 있는 마지막 자리에 저장, 나머지 힙을 최대 힙으로 재구성하는데 공백 힙이 되었으므로 힙 정렬 종료



CSLAB

# 제출

## • 제출

- 개인실습 (#1)
  - 오늘 자정까지 제출 ( ~ 2020/5/22 23:59 )
  - Quick sort
- 과제
  - Lab10.docx
  - 다음주 목요일 자정까지 제출 ( ~2020/5/28 23:59 )
  - Heap sort

CSLAB