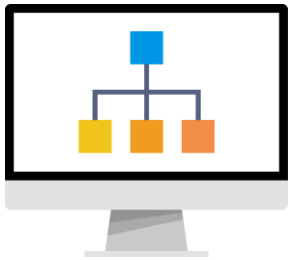


Data Structure & Algorithm
자료구조 및 알고리즘

17. 정렬 (Sorting, Part 1)

15강 보고서 돌아보기

- 포화 k진 트리(포화 3진 트리, 4진 트리..)가 있다고 가정하자. 아래 문제에 대한 답을 n과 k에 대한 식으로 나타내어라.

- 레벨 n에는 몇 개의 노드가 있을까? $\boldsymbol{k^n}$

- 높이 n인 포화 k진 트리에는 몇 개의 노드가 있을까? $\frac{\boldsymbol{k^{n+1}-1}}{\boldsymbol{k-1}}$

16강 보고서 돌아보기

1) 길이 N인 배열에 N개의 정수가 있다. 이 정수를 커지는 순서대로 나열하는 작업을 “오름차순 정렬”이라고 한다.

최소 힙을 하나 사용하여 오름차순 정렬을 구현할 수 있을까? 또, 이 때 시간복잡도는 얼마일까? **힙 정렬, $\boldsymbol{O(N\ log\ N)}$**

16강 보고서 돌아보기

2) 배열에 1이 하나 들어가 있다. 매 번 두 개의 정수가 배열에 추가된다. 이 때마다 배열의 중간 값을 출력하고 싶다.

- 정렬을 사용해서 해결할 수 있을까?
- 최소 힙 하나와 최대 힙 하나를 사용해서 더 효율적으로 수행할 수 있을까?

배열: [1], 중간 값: 1

배열: [1, 5, 2], 중간 값: 2

배열: [1, 5, 2, 4, 3], 중간 값: 3

배열: [1, 5, 2, 4, 3, 100, 2], 중간 값: 3

16강 보고서 돌아보기

- 현재 입력 중 중간 값 m (초기 값은 1)
- m보다 작은 수를 넣는 최대 힙 H1
- m보다 큰 수를 넣는 최소 힙 H2

- 새로 들어온 수 a, b $(a \leq b)$가
  - $a \leq m \leq b$ 일 경우, a를 H1에, b를 H2에 삽입
  - $a \leq b \leq m$ 일 경우, a와 b를 H1에 m을 H2에 삽입하고 H1에서 값을 하나 빼면 이것이 새로운 중간값 m
  - $m \leq a \leq b$ 일 경우, a와 b를 H2에 m을 H1에 삽입하고 H2에서 값을 하나 빼면 이것이 새로운 중간값 m

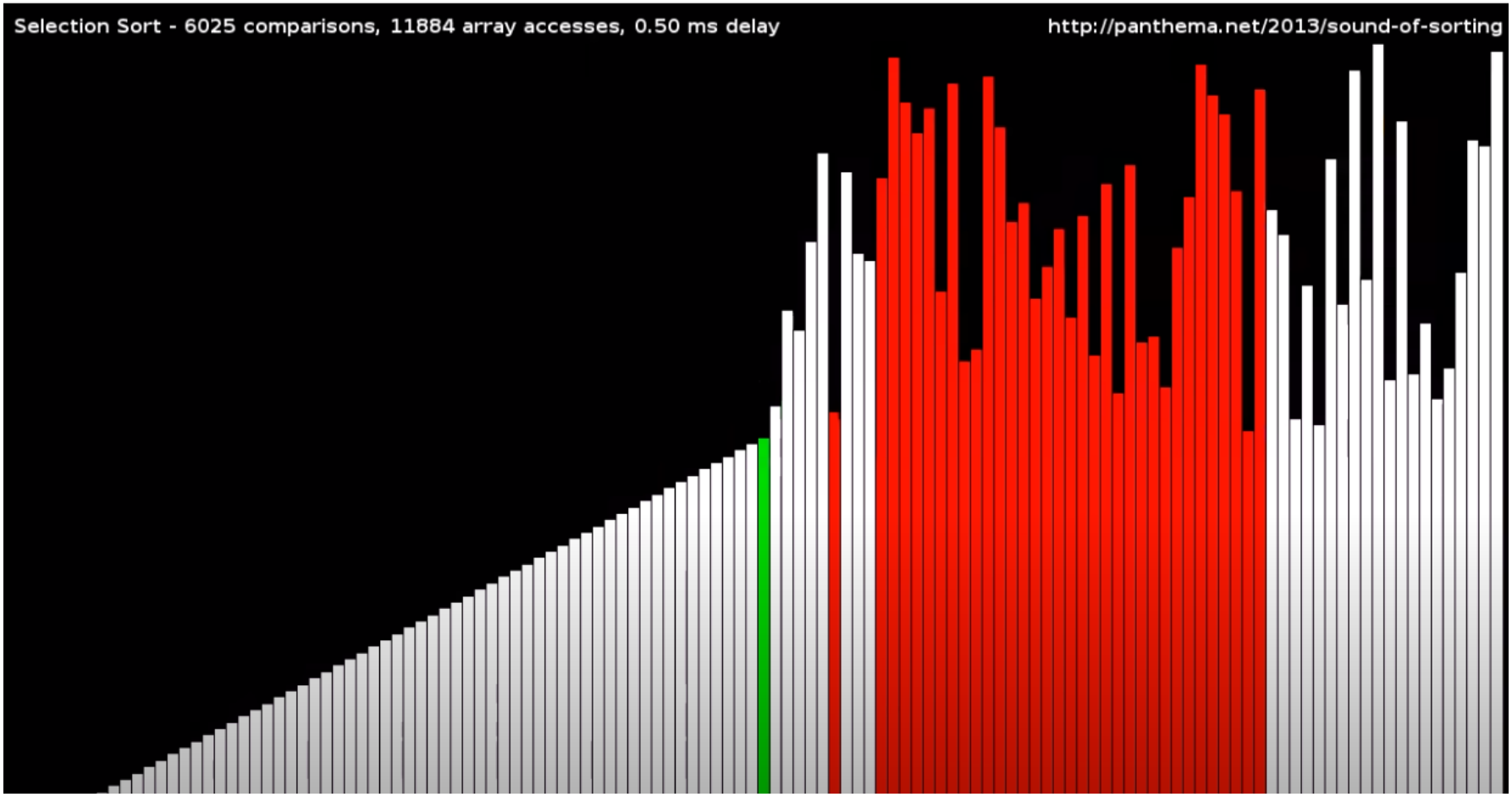
정렬 알고리즘

- 정렬(sorting): n개의 요소가 주어졌을 때 이를 규칙에 맞게 재배열 하는 것
  - 실수의 경우 오름차순, 내림차순

- 버블 정렬: Bubble sort
- 선택 정렬: Selection sort
- 삽입 정렬: Insertion sort
- 힙 정렬: Heap sort
- 병합 정렬: Merge sort
- 퀵 정렬: Quick sort
- 기수 정렬: Radix sort

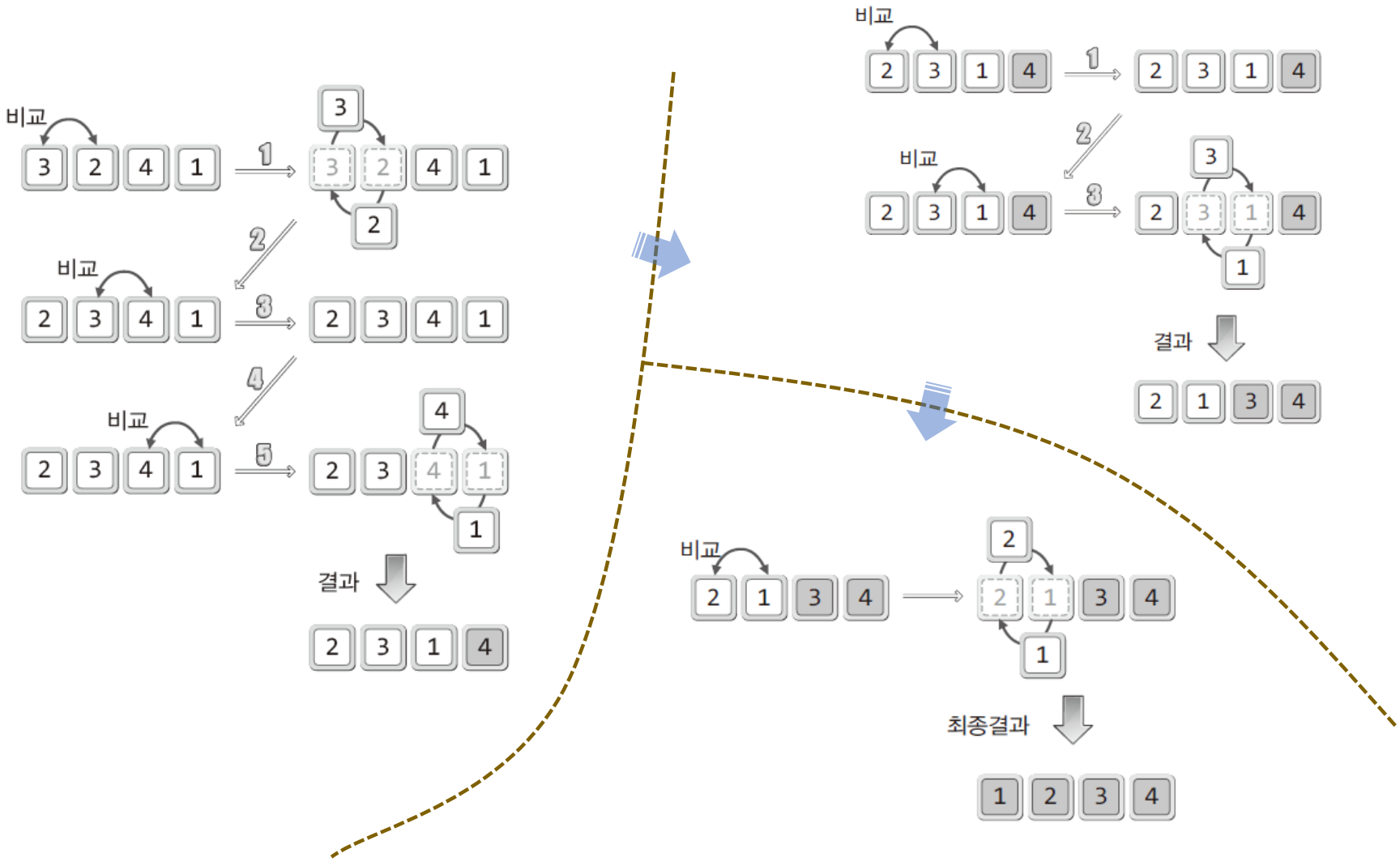
정렬 알고리즘 “듣기”

- https://www.youtube.com/watch?v=kPRA0W1kECg



$O(N^2)$ 정렬 알고리즘

버블 정렬: 이해


비교
3 2 4 1
1
3 2 4 1
2
2 3 4 1
비교
2 3 4 1
2
2 3 4 1
3
2 3 4 1
비교
2 3 4 1
4
2 3 4 1
5
1
결과
2 3 1 4
비교
2 3 1 4
1
2 3 1 4
2
2 3 1 4
비교
2 3 1 4
3
2 3 1 4
1
결과
2 1 3 4
비교
2 1 3 4
2
2 1 3 4
1
최종결과
1 2 3 4

버블 정렬: 구현

```
void BubbleSort(int arr[], int n)
{
    int i, j;
    int temp;

    for(i=0; i<n-1; i++)
    {
        for(j=0; j<(n-i)-1; j++)
        {
            if(arr[j] > arr[j+1])
            {
                // 데이터의 교환 /////
                temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }
}
```

버블 정렬의 실질적 구현 코드

```
int main(void)
{
    int arr[4] = {3, 2, 4, 1};
    int i;

    BubbleSort(arr, sizeof(arr)/sizeof(int));

    for(i=0; i<4; i++)
        printf("%d ", arr[i]);

    printf("\n");
    return 0;
}
```

실행결과

```
1 2 3 4
```

버블 정렬: 성능평가

- 비교의 횟수 : 두 데이터간의 비교연산의 횟수
- 이동의 횟수 : 위치의 변경을 위한 데이터의 이동횟수

```
for(i=0; i<n-1; i++)
{
    for(j=0; j<(n-i)-1; j++)
    {
        if(arr[j] > arr[j+1]) { . . . . }
    }          비교 연산
}
```

최악의 경우

비교의 횟수와 이동의 횟수는 일치한다.

비교의 횟수

$$(n-1) + (n-2) + \cdots + 2 + 1$$

$$\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} = \frac{n^2-n}{2}$$

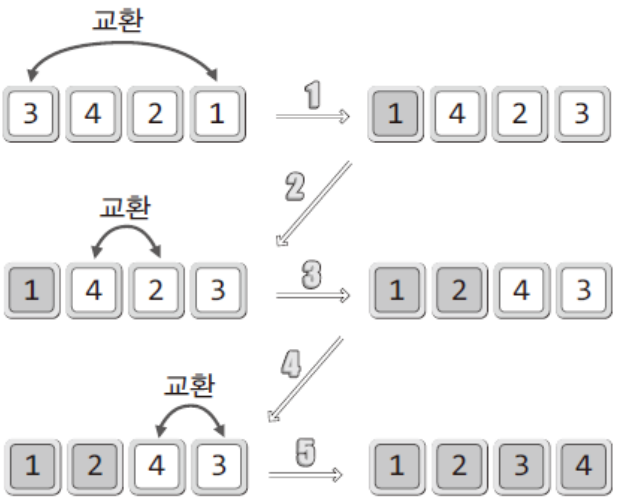
$$O(n^2)$$

선택 정렬: 이해

하나씩 선택해서 정렬 결과를 완성해 나간다!

별도의 메모리 공간이 요구된다는 단점이 있다!

개선!



하나씩 비워 가면서 이동을 시킨다.

선택 정렬: 구현

```
void SelSort(int arr[], int n)
{
    int i, j;
    int maxIdx;
    int temp;

    for(i=0; i<n-1; i++)
    {
        maxIdx = i;

        for(j=i+1; j<n; j++)     // 최솟값 탐색
        {
            if(arr[j] < arr[maxIdx])
                maxIdx = j;
        }

        // 교 환 ///////
        temp = arr[i];
        arr[i] = arr[maxIdx];
        arr[maxIdx] = temp;
    }
}
```

```
int main(void)
{
    int arr[4] = {3, 4, 2, 1};
    int i;

    SelSort(arr, sizeof(arr)/sizeof(int));

    for(i=0; i<4; i++)
        printf("%d ", arr[i]);

    printf("\n");
    return 0;
}
```

실행결과

```
1 2 3 4
```

선택 정렬: 성능평가

비교의 횟수

```
for(i=0; i<n-1; i++)
{
    maxIdx = i;

    for(j=i+1; j<n; j++)
    {
        if(arr[j] < arr[maxIdx])   비교 연산
            maxIdx = j;
    }

    // 교 환 ///////   이동 연산
    temp = arr[i];
    arr[i] = arr[maxIdx];
    arr[maxIdx] = temp;
}
```

$$(n-1) + (n-2) + \cdots + 2 + 1$$

$$\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} = \frac{n^2-n}{2}$$

$$O(n^2)$$

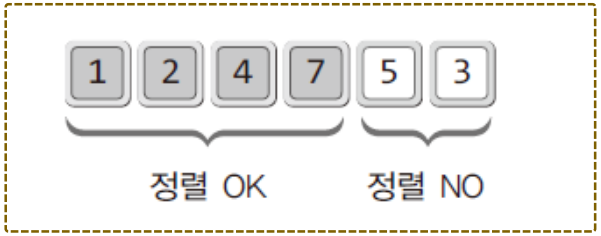
최악의 경우와 최상의 경우 구분 없이 데이터 이동의 횟수는 동일하다!

선택 정렬: 간단하게 짤 때

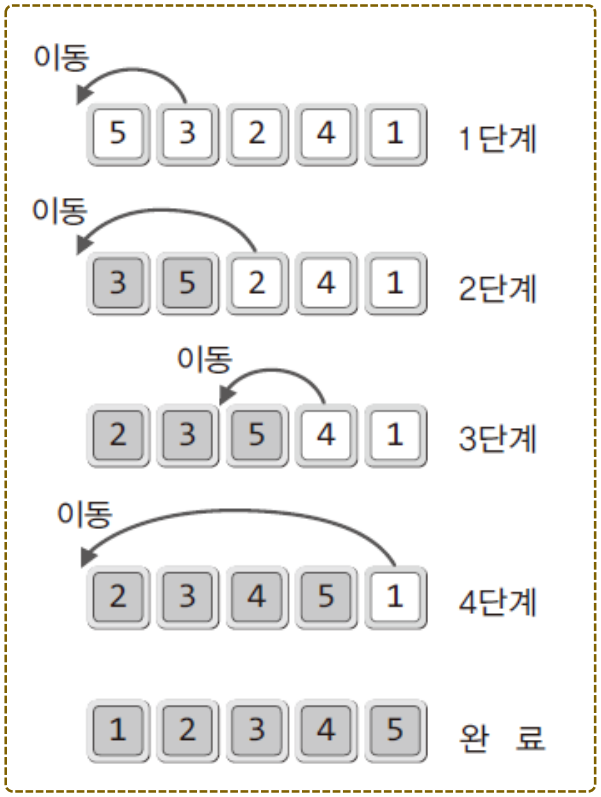
- 비교 횟수는 동일
- 최솟값의 인덱스를 저장하는 변수를 두는 것이 아니라 arr[i] 자체에 최솟값을 저장하므로 교환 횟수가 많다.
  - 내림차순이면 최댓값
- 코드가 간결하므로 통째로 외워서 쓰면 좋음

```c
#include <stdio.h>

void SelSort(int arr[], int n) {
    int i, j, temp;

    for (i = 0; i < n-1; i++) {
        for (j = i + 1; j < n; j++) {
            if (arr[i] > arr[j]) {
                temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }
}
int main() {
    int arr[4] = { 3, 4, 2, 1 };
    int i;

    SelSort(arr, sizeof(arr) / sizeof(int));

    for (i = 0; i < 4; i++)
        printf("%d ", arr[i]);

    printf("\n");
    return 0;
}
```

삽입 정렬: 이해

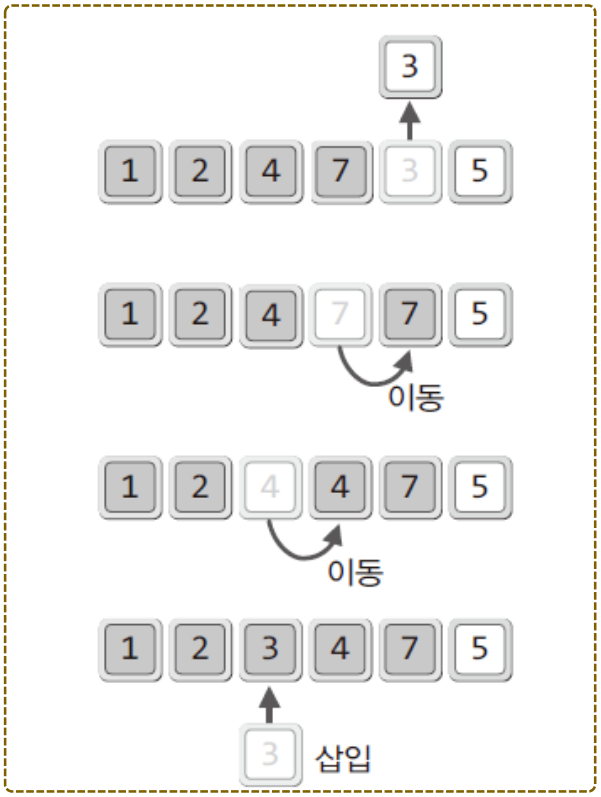


정렬이 완료된 영역과 그렇지 않은 영역을 구분하는 방법



구현을 고려하면!

뒤로 밀어 내는 방법을 포함한 그림



삽입 정렬: 구현

```
void InserSort(int arr[], int n)
{
    int i, j;
    int insData;

    for(i=1; i<n; i++)
    {
        insData = arr[i];

        for(j=i-1; j>=0 ; j--)
        {
            if(arr[j] > insData)
                arr[j+1] = arr[j];
            else
                break;
        }

        arr[j+1] = insData;
    }
}
```

정렬 대상 저장

비교 대상 뒤로 한 칸 밀기

찾은 위치에 정렬대상 삽입

```
int main(void)
{
    int arr[5] = {5, 3, 2, 4, 1};
    int i;

    InserSort(arr, sizeof(arr)/sizeof(int));

    for(i=0; i<5; i++)
        printf("%d ", arr[i]);

    printf("\n");
    return 0;
}
```

실행결과

```
1 2 3 4 5
```

삽입 정렬: 성능평가

```
for(i=1; i<n; i++)
{
    . . . .
    for(j=i-1; j>=0 ; j--)
    {
        if(arr[j] > insData)
            arr[j+1] = arr[j];
        else
            break;
    }
    . . . .
}
```

데이터간 비교연산

데이터간 이동연산

최악의 경우 이 break문은 한번도 실행이 되지 않는다.

데이터의 비교 및 이동 연산이 안쪽 for문 안에 존재한다.

따라서 최악의 경우 빅-오는 버블 및 삽입 정렬과 마찬가지로 $O(n^2)$

효율적인 정렬 알고리즘

힙 정렬: 이해와 구현

힙의 특성을 활용하여, 힙에 정렬할 대상을 모두 넣었다가 다시 꺼내어 정렬을 진행한다!

```
int PriComp(int n1, int n2)
{
    return n2-n1;       // 오름차순 정렬을 위한 문장
//  return n1-n2;
}
```

힙에서 요구하는 정렬 기준 함수와 동일한 기준을 적용한다!

```
void HeapSort(int arr[], int n, PriorityComp pc)
{
    Heap heap;
    int i;

    HeapInit(&heap, pc);

    // 정렬대상을 가지고 힙을 구성한다.
    for(i=0; i<n; i++)
        HInsert(&heap, arr[i]);

    // 순서대로 하나씩 꺼내서 정렬을 완성한다.
    for(i=0; i<n; i++)
        arr[i] = HDelete(&heap);
}
```

```
int main(void)
{
    int arr[4] = {3, 4, 2, 1};
    int i;

    HeapSort(arr, sizeof(arr)/sizeof(int), PriComp);

    for(i=0; i<4; i++)
        printf("%d ", arr[i]);

    printf("\n");
    return 0;
}
```

UsefulHeap.h
UsefulHeap.c

HeapSort.c

```
1 2 3 4
```

실행결과

힙 정렬: 성능평가

하나의 데이터를 힙에 넣고 빼는 경우에 대한 시간 복잡도

- 힙의 데이터 저장 시간 복잡도 $O(log_2 n)$
- 힙의 데이터 삭제 시간 복잡도 $O(log_2 n)$

하나로 묶으면

$O(2log_2 n)$, 그런데

앞의 2는 빅-오에서 무시 할 수 있으므로! $O(log_2 n)$

N개의 데이터

$O(nlog_2 n)$

두 빅-오의 비교를 위한 테이블

n	10	100	1,000	3,000	5,000
n^2	100	10,000	1,000,000	9,000,000	25,000,000
$n\log_2 n$	66	664	19,931	34,652	61,438

요약 (오름차순 정렬 기준)

- 버블 정렬: 인접한 데이터를 비교하여 가장 큰 데이터를 뒤에서부터 쌓아 나가는 정렬
- 선택 정렬: 데이터에서 최솟값을 반복적으로 선택하여 배열의 앞에서부터 저장하는 정렬
- 삽입 정렬: 현재 데이터를 앞서 정렬된 배열의 어디에 오는지 찾고 사이에 있는 데이터를 뒤로 밀고 중간에 삽입하는 정렬

- 힙 정렬: 최소 힙을 이용하여 데이터를 힙에 삽입하고 삭제한 결과를 나열하는 정렬

출석 인정을 위한 보고서 작성

- 아래 질문에 대해 답한 후 포털에 제출 (단답형이므로 분량 제한 없음)

- 오름차순으로 정렬된 배열이 있다. 이 배열에서 무작위 인덱스의 값 하나를 “아주 큰 수”로 덮어썼다.

- 이 배열은 “아주 큰 수” 하나를 제외하고는 거의 정렬되어 있는 상태인데 완벽히 정렬하고 싶다. 버블 정렬, 선택 정렬, 삽입 정렬 중 어떤 정렬 기법이 가장 효과적일까?