알고리즘 2주차
: 우선순위 큐, 힙과힙정렬1

2조 21011948 신아진, 21011954 이은서

00

목차

01

우선순위 큐 개념정리

우선순위 큐란?

- **큐(Queue)**는 먼저 들어오는 데이터가 먼저 나가는 FIFO(First In First Out) 형식의 자료구조.
- **우선순위 큐(Priority Queue)**는 먼저 들어오는 데이터가 아니라, 우선순위가 높은 데이터가 먼저 나가는 형태의 자료구조.

우선순위 큐를 이용한 정렬

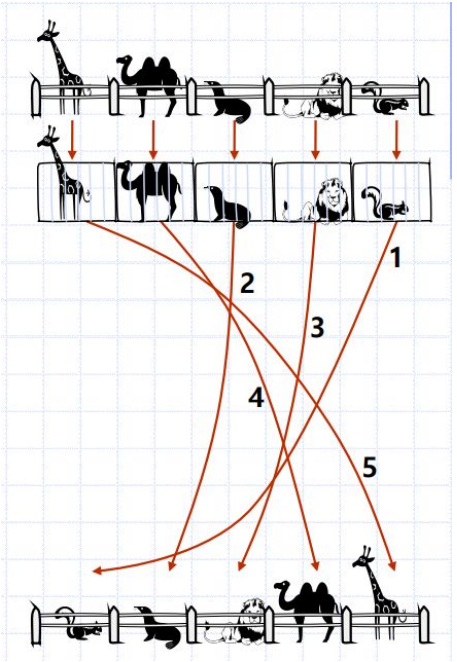
- **선택 정렬**(무순리스트 이용), **삽입 정렬**(순서리스트 이용)
- 우선순위 큐의 구현에 따라 실행시간이 다르다

우선순위 큐 개념정리

선택 정렬(selection-sort)

- 우선순위 큐의 일종
- 무순리스트로 구현
- 우선순위가 가장 높은 것을 선택하여 제일 앞으로 보내기
- 실행시간 : n회의 원소 삽입, n회의 정렬 순서로 삭제

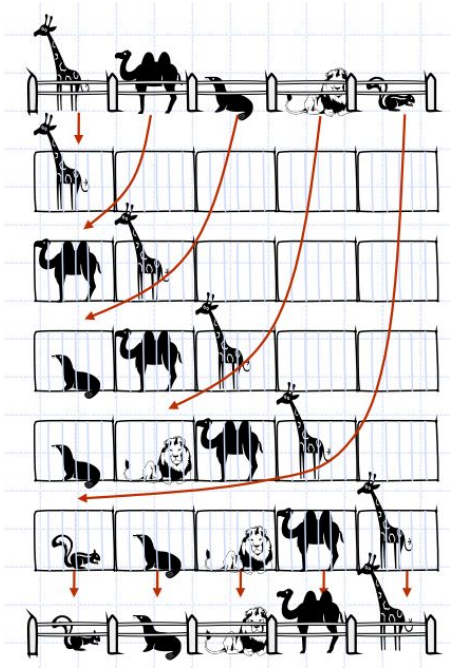
  → n + n-1 + n-2 + ⋯ + 2 + 1 = n*(n+1)/2 , Total = O(n^2)



삽입 정렬(insertion-sort)

- 우선순위 큐의 일종
- 순서리스트로 구현
- 각 원소를 적절한 위치에 삽입하기(들어갈 위치 찾기)
- 실행시간 : n회의 원소 삽입, n회의 정렬 순서로 삭제

  → n + n-1 + n-2 + ⋯ + 2 + 1 = n*(n+1)/2 , Total = O(n^2)

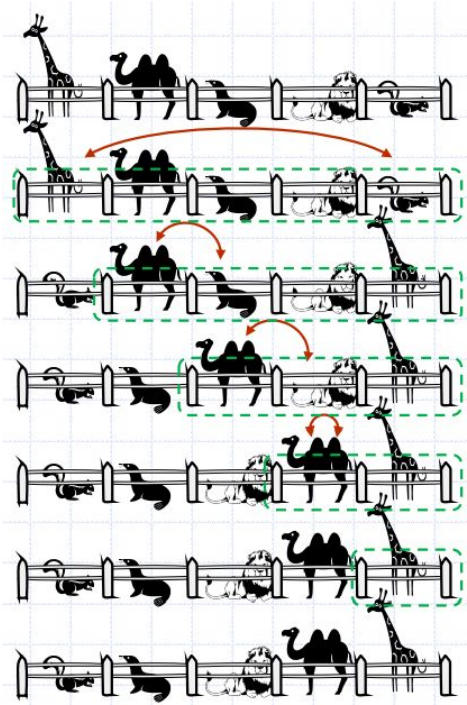


제자리(In-place) 정렬이란?

- 정렬하기 위해 일정한 추가공간(O(1)개 or 상수개)을 사용
- 주어진 리스트(or 배열)에 속한 원소의 순서만 바꿔서 리스트를 정렬
- 선택 정렬과 삽입 정렬은 주어진 리스트 및 루프(반복문) 변수 정도만 사용하므로 제자리 정렬
- 외부 데이터구조 사용 대신, 제자리 선택 정렬, 제자리 삽입 정렬 구현 가능

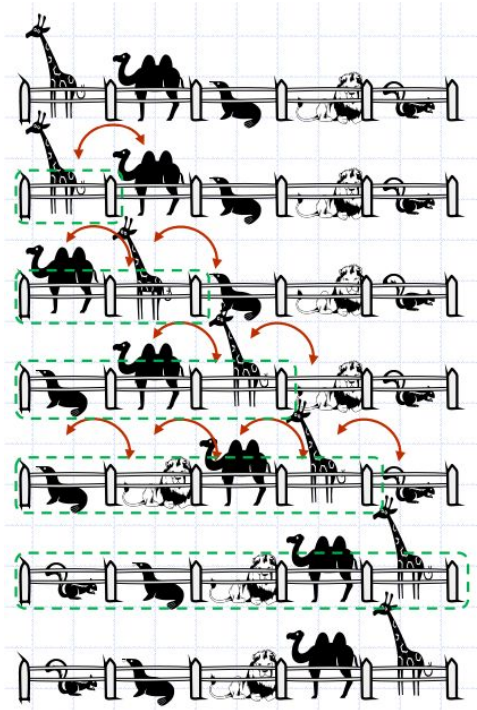
제자리 선택 정렬(In-place selection-sort)

- 외부 데이터구조를 사용하는 대신 주어진 리스트만을 사용
- 리스트를 변경하는 대신 swapElements(원소끼리 바꾸기) 사용
- 우선순위가 가장 높은 것을 선택하여 제일 앞으로 보내기



제자리 삽입 정렬(In-place insertion-sort)

- 외부 데이터구조를 사용하는 대신 주어진 리스트만을 사용
- 리스트를 변경하는 대신 swapElements(원소끼리 바꾸기) 사용
- 각 원소를 적절한 위치에 삽입하기(들어갈 위치 찾기)



02 우선순위 큐 실습문제 - 1) 선택 정렬

[문제 1] (선택 정렬) n개의 양의 정수(중복 가능)를 입력받아, 아래에서 설명하는 선택 정렬을 이용하여 정렬하는 프로그램을 작성하시오.

- 구현해야 할 선택 정렬 알고리즘 (가장 큰 값을 찾는 버전):
  - 크기가 **n**인 배열을 동적 할당하여, 입력된 양의 정수 저장(입력 정수는 중복 가능)
  - 제자리(in place) 정렬 사용.
    즉, 입력 값 저장을 위한 배열 이외에 **O(1)**의 추가 공간만 사용
  - 배열의 뒷 부분을 정렬 상태로 유지하고, 매 반복마다 최대 한 번의 교환 연산만 사용
    (매 반복마다 가장 큰 값을 찾아, 오른쪽부터 채우는 방식으로 정렬)
  - 가능하면 교재의 의사코드를 보지 말고 구현해볼 것을 권장
  - **참고:** 아래 그림에 예시된 버전은 매 반복마다 가장 작은 값을 찾아 왼쪽부터 채워 나가는, 따라서 교재의 알고리즘과는 정반대 방향으로 작동하는 버전이다.

- 알고리즘 동작 과정 예시 (**n = 8**)

초기 상태:	8	31	48	73	3	65	20	29
1번째 반복 후:	8	31	48	29	3	65	20	**73**
2번째 반복 후:	8	31	48	29	3	20	**65**	73
3번째 반복 후:	8	31	20	29	3	**48**	65	73
4번째 반복 후:	8	3	20	29	**31**	48	65	73
5번째 반복 후:	8	3	20	**29**	31	48	65	73
6번째 반복 후:	8	3	**20**	29	31	48	65	73
7번째 반복 후:	3	**8**	20	29	31	48	65	73

(73과 29 교환) (65와 20 교환) (48과 20 교환) (31과 3 교환) (29 제자리 교환) (20 제자리 교환) (8과 3 교환)

입력 예시 1	출력 예시 1
8 ↦ n 8 31 48 73 3 65 20 29	□3 8 20 29 31 48 65 73 ↦ 정렬 결과

입력 예시 2	출력 예시 2
8 ↦ n 73 65 48 31 29 20 8 3	□3 8 20 29 31 48 65 73 ↦ 정렬 결과

우선순위 큐 실습문제 - 1) 선택 정렬

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void) {
	int n, temp;
	scanf("%d", &n);
	getchar();
	int* A = (int*)malloc(n * sizeof(int));
	for (int i = 0; i < n; i++)
		scanf("%d", A + i);

	for (int i = n - 1; i >= 1; i--) {
		int maxLoc = i;
		for (int j = i - 1; j >= 0; j--) {
			if (A[j] > A[maxLoc])
				maxLoc = j;
		}
		temp = A[i];
		A[i] = A[maxLoc];
		A[maxLoc] = temp;
	}

	for (int i = 0; i < n; i++)
		printf(" %d", A[i]);

	return 0;
}
```

1) n개의 양의 정수를 입력받기 (동적할당 이용)

2) 제자리 선택 정렬 알고리즘,
외부 반복문 i = n-1 ~ 1, 배열 A의 뒷부분부터 정렬

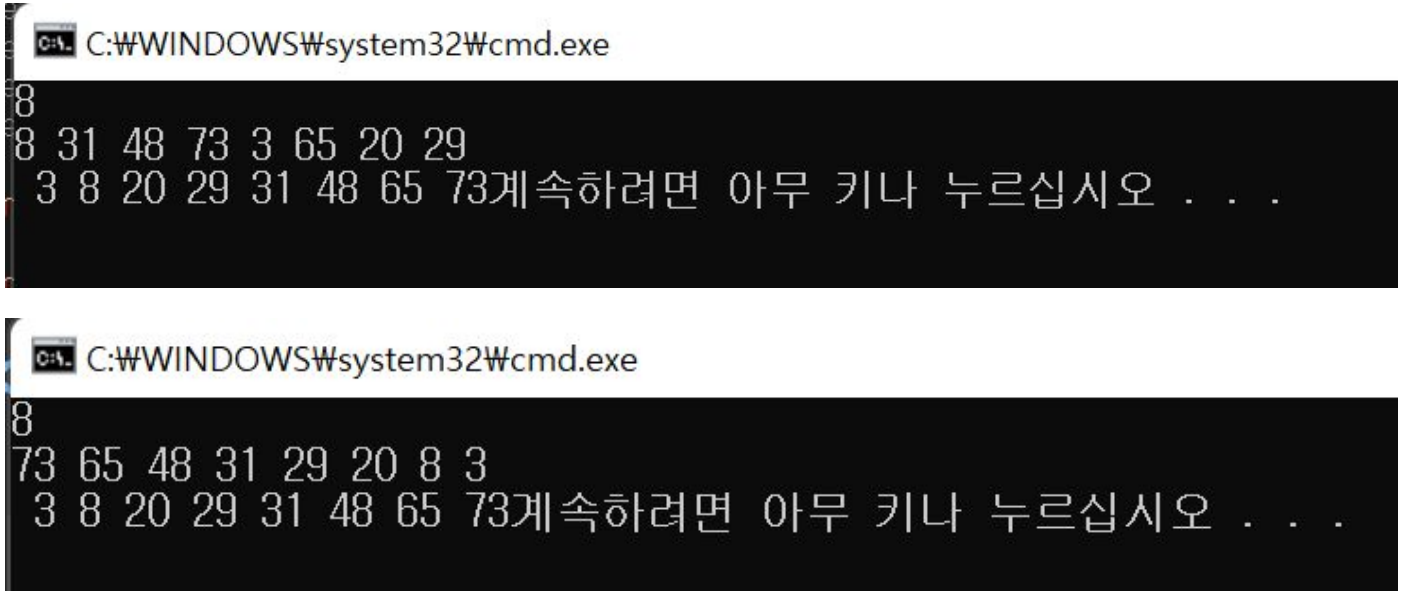
2-1) 인덱스 i번의 바로 전 원소부터 0번 인덱스 원소까지 순회
A[j] > A[maxLoc] → maxLoc = j

2-2) 배열의 인덱스 i 번과 maxLoc 번에 해당하는 값들을 swap

3) 배열의 모든 값 출력

우선순위 큐 실습문제 - 1) 선택 정렬

실행 결과



우선순위 큐 실습문제 - 2) 삽입 정렬

[문제 2] (삽입 정렬) n개의 양의 정수를 입력(중복 가능)받아, 아래에서 설명하는 삽입 정렬을 이용하여 정렬하는 프로그램을 작성하시오.

◦ 구현해야 할 삽입 정렬 알고리즘:

- 크기가 **n**인 배열을 동적 할당하여, 입력된 양의 정수 저장(입력 정수는 중복 가능)
- 제자리(in-place) 정렬 사용.
  즉, 입력 값 저장을 위한 배열 이외에 **O(1)**의 추가 공간만 사용
- 배열의 앞부분을 정렬 상태로 유지
- 가능하면 교재의 의사코드를 보지 말고 구현해볼 것을 권장

◦ 알고리즘 동작 과정 예시 (**n = 7**)

초기 상태 :	3	73	48	31	8	11	20	
1번째 반복 후:	3	**73**	48	31	8	11	20	73 삽입
2번째 반복 후:	3	**48**	73	31	8	11	20	48 삽입
3번째 반복 후:	3	**31**	48	73	8	11	20	31 삽입
4번째 반복 후:	3	**8**	31	48	73	11	20	8 삽입
5번째 반복 후:	3	8	**11**	31	48	73	20	11 삽입
6번째 반복 후:	3	8	11	**20**	31	48	73	20 삽입

입력 예시 1	출력 예시 1
7 ↦ n 3 73 48 31 8 11 20	□3 8 11 20 31 48 73 ↦ 정렬 결과

입력 예시 2	출력 예시 2
8 ↦ n 73 65 48 31 29 20 8 3	□3 8 20 29 31 48 65 73 ↦ 정렬 결과

우선순위 큐 실습문제 - 2) 삽입 정렬

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void) {
        int n, save;
        scanf("%d", &n);
        getchar();
        int* A = (int*)malloc(n * sizeof(int));
        for (int i = 0; i < n; i++)
                scanf("%d", A + i);

        for (int i = 1; i < n; i++) {
                save = A[i];
                int j = i - 1;
                while ((j >= 0) && (A[j] > save)) {
                        A[j + 1] = A[j];
                        j--;
                }
                A[j + 1] = save;
        }

        for (int i = 0; i < n; i++)
                printf(" %d", A[i]);

        return 0;
}
```

1) n개의 양의 정수를 입력받기 (동적할당 이용)

2) 제자리 삽입 정렬 알고리즘,

외부 반복문 i = 1 ~ n-1, 배열 A의 앞부분부터 정렬

2-1) 인덱스 j 값이 0 이상이고 A[j]값이 변수 save(=A[i])보다 클 때

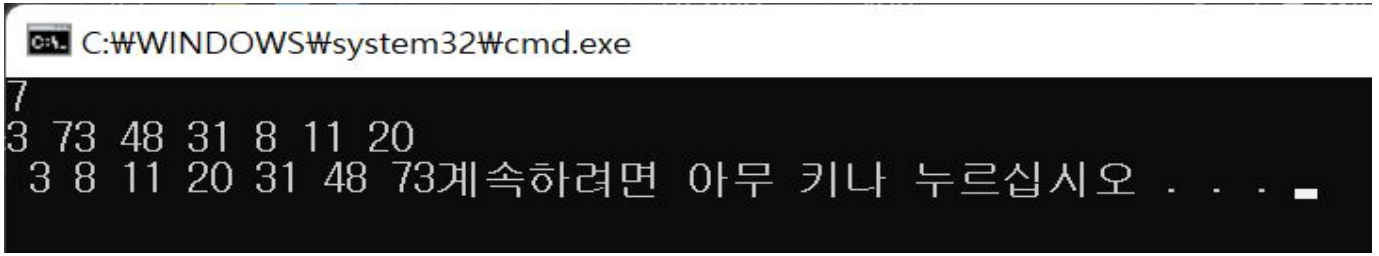
→ A[j]의 다음에 A[j]값 저장, j값 1 감소

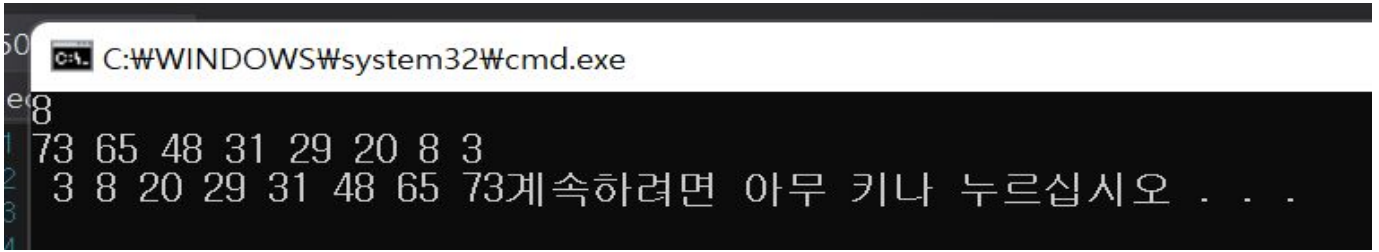
2-2) 위 반복문에서 멈춰버린 j의 바로 다음 원소값에 A[i] 저장

3) 배열의 모든 값 출력

우선순위 큐 실습문제 - 2) 삽입 정렬

실행 결과





힙과 힙 정렬 개념 정리

힙(Heap)이란?

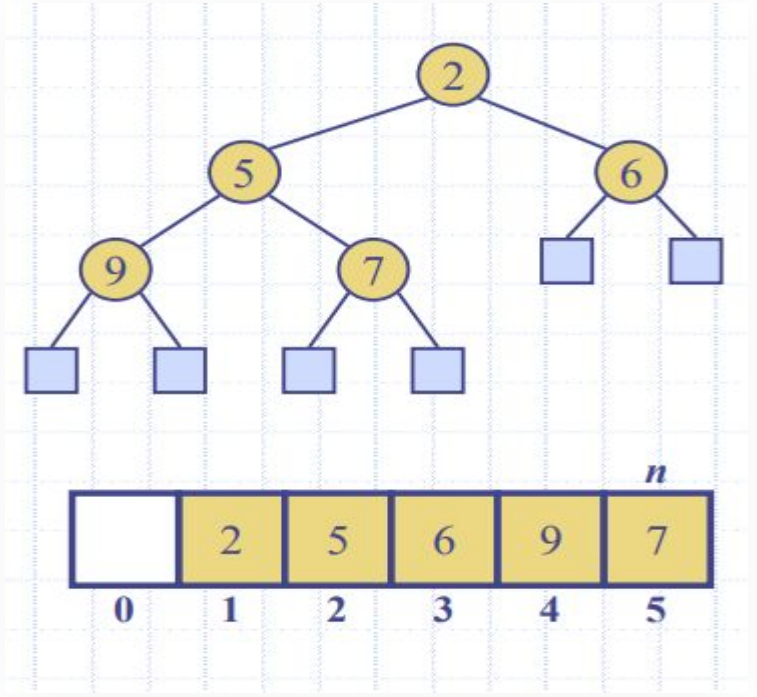
- 내부노드에 키를 저장하며 힙순서와 완전이진트리의 속성을 만족하는 이진트리
- 최솟값이나 최댓값을 빠르게 찾아내기 위해 완전 이진 트리를 기반으로 하는 트리

힙 정렬이란?

- 힙에 기초한 우선순위 큐를 이용함으로써, n개의 원소로 이루어진 리스트를 O(n log n) 시간에 정렬 가능
- 선택 정렬이나 삽입 정렬과 같은 2차 정렬 알고리즘보다 훨씬 빠름
- 제자리 힙 정렬, (비재귀적)상향식 힙생성, 삽입식 힙생성 등이 있음

배열에 기초한 힙 구현

- n개의 키를 가진 힙을 크기 n의 배열을 사용해 표현 가능
- 첨자 i에 존재하는 노드
  →왼쪽 자식 첨자 2i
  →오른쪽 자식 첨자 2i+1
  →부모 첨자 i/2
- 첨자 0 셀은 사용하지 않음
- insertItem 작업은 첨자 n+1 위치에 삽입
- removeMin 작업은 첨자 n 위치에서 삭제

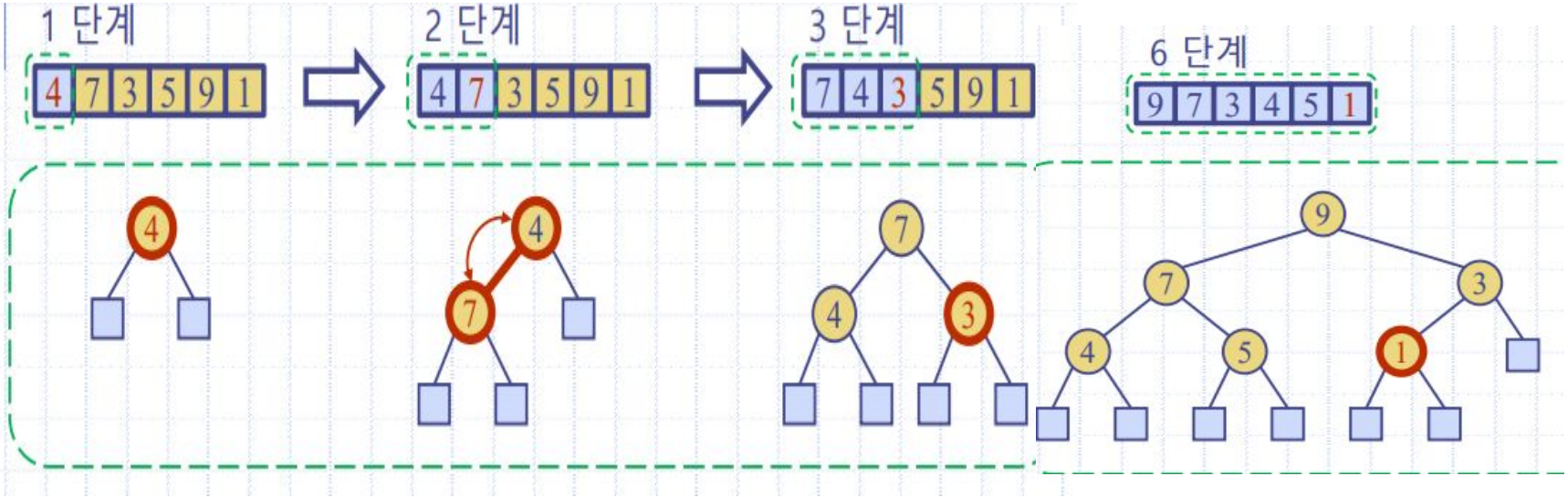


힙과 힙 정렬 개념 정리

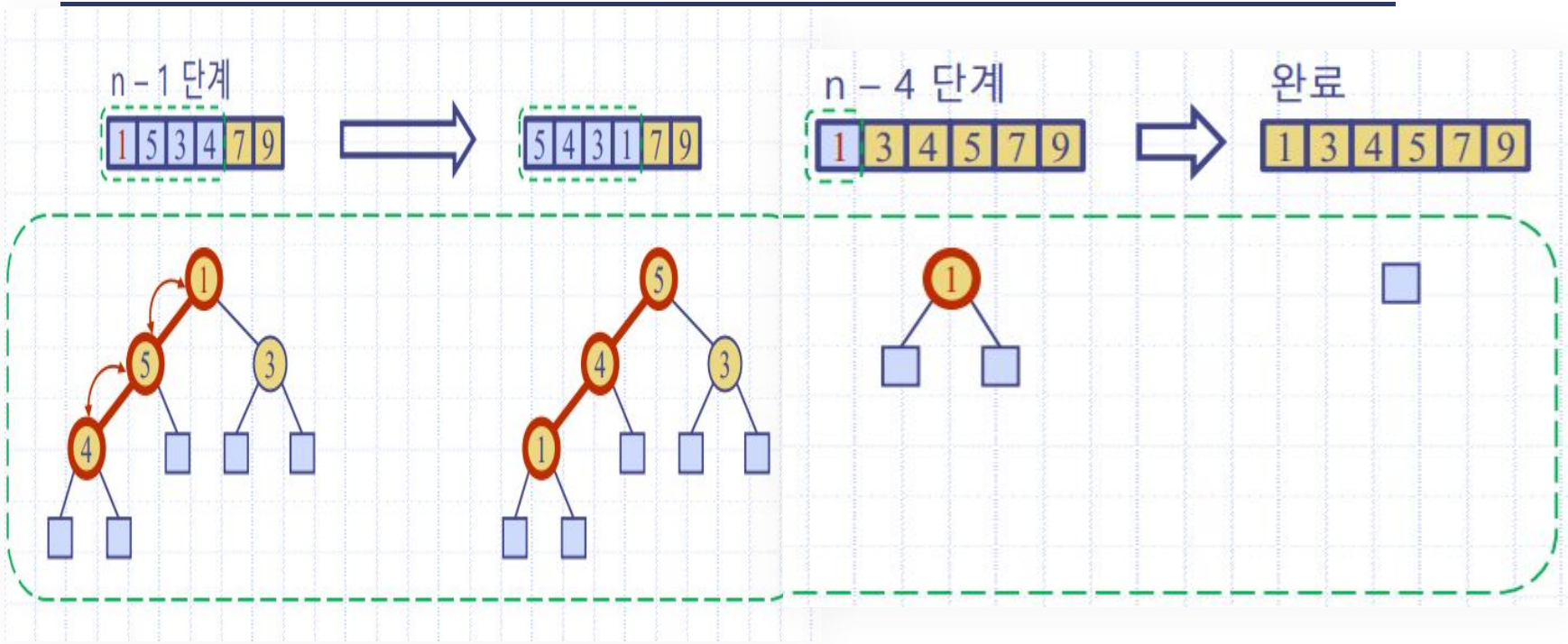
제자리 힙 정렬

- 정렬되어야 할 리스트가 배열로 주어진 경우에만 적용
- 힙 저장 시에 리스트 일부를 사용→ 외부 힙 사용 피함
- 최소힙 대신 최대 원소가 맨위에 오는 최대힙 사용
- 첨자 k의 원소는 첨자 2k 및 2k+1의 자식들보다 크거나 같아야 함
- 1기 작업과 2기 작업으로 나눌 수 있음

제자리 힙 정렬 - 1기



제자리 힙 정렬 - 2기



힙 생성

- **삽입식(insertion)**
  모든 키들이 미리 주어진 경우, 키들이 차례로 주어지는 경우, 모두 적용 가능

- **상향식(bottom-up)**
  모든 키들이 미리 주어진 경우에만 적용 가능
  각 재귀호출이 힙인 부트리를 반환하는 방식이므로 상향식이라 불림
  즉, 외부노드에서 시작하여 각 재귀호출이 반환함에 따라 트리 위쪽으로 진행

- **비재귀적 상향식**
  정렬 되어야 할 리스트가 배열로 주어진 경우만 적용
  힙생성 절차는 후진 방향으로 반복 수행
  시작 노드: 첨자 n/2인 노드

힙과 힙 정렬 실습문제

다음 문제 1과 2는 각각 삽입식 및 상향식 두 가지 버전의 **힙 생성**에 대한 프로그래밍 요구다. 먼저, 두 문제 모두에 공통된 요구 사항이다.

1) **순차힙**으로 구현한다. 즉, 배열을 이용한 순차트리 형식으로 합을 저장한다.
2) **최대힙**으로 구현한다. 따라서 힙으로부터 삭제 작업은 우선순위 큐에서 일반적으로 이루어지는 최소값 삭제가 아닌 최대값 삭제가 된다(참고로 최소힙에서는 이것과 반대로 수행함).
3) 연산 효율을 위해 배열 인덱스 0 위치는 사용하지 않고 비워둔다.
4) 데이터구조를 단순화하기 위해 힙의 항목으로써 (**키**, **원소**) 쌍에서 원소를 생략하고 **키**만 저장하는 것으로 한다.
5) 키들은 중복이 없는 1 이상의 정수로 전제한다 – 즉, 중복 키 검사는 불필요하다.
6) O(1) 공간으로 수행할 것 – 즉, 주어진 키들을 저장한 초기 배열 외 추가 메모리 사용은 O(1)을 초과할 수 없다.
7) 힙은 어느 시점에서라도 최대 항목 수 < 100 으로 전제한다.

힙과 힙 정렬 실습문제 – 삽입식 힙 생성

[문제 1] 삽입식 힙 생성

다음의 대화식 프로그램을 작성해 삽입식 힙 생성을 구현하라.

1) 키들은 한 개씩 차례로 삽입 명령과 함께 주어진다. 즉, 키가 입력될 때마다 즉시 힙에 삽입해야 한다. 만약 이렇게 하지 않고 문제 2에서 하는 것처럼 키들이 모두 입력되기를 기다려 한꺼번에 상향식으로 힙을 생성하면 대화식 프로그램의 인쇄(p) 또는 삭제(d) 명령의 수행이 어려워진다.

힌트:

1. 현재 총 항목 수를 유지하는 변수 n을 유지하고 삽입 및 삭제 직후 갱신한다.
2. 순차힙에 새로운 항목의 초기 삽입 위치는 항상 n+1이다.

필요함수: insertItem(key)
removeMax()
upHeap(i)
downHeap(i)
printHeap()

힙과 힙 정렬 실습문제 – 삽입식 힙 생성

```c
int H[100];
int n = 0;
void upHeap(int i) {
    int tmp;
    if (i == 1)
        return;
    if (H[i] <= H[i / 2])
        return;
    tmp = H[i];
    H[i] = H[i / 2];
    H[i / 2] = tmp;
    upHeap(i / 2);
}
void insertItem(int key) {
    H[++n] = key;
    upHeap(n);
}
void downHeap(int i) {
    int tmp;
    if ((n < (i * 2)) && (n < (i * 2 + 1)))//자식 노드가 없는경우
        return;
    int big = i * 2;
    if (n >= i * 2 + 1) {//왼쪽,오른쪽 자식노드 중에서 큰 노드를 big
        if (H[i * 2 + 1] > H[big])
            big = i * 2 + 1;
    }
    if (H[i] >= H[big])//부모노드나 루트노드보다 자식노드가 작은경우
        return;
    tmp = H[i];
    H[i] = H[big];
    H[big] = tmp;
    downHeap(big);
}
```

i(=힙의 크기)가 1이거나 부모의 힙이 현재 힙보다 크거나 같은 경우 return 함

위의 경우가 아니면 부모의 힙과 현재 힙 swap

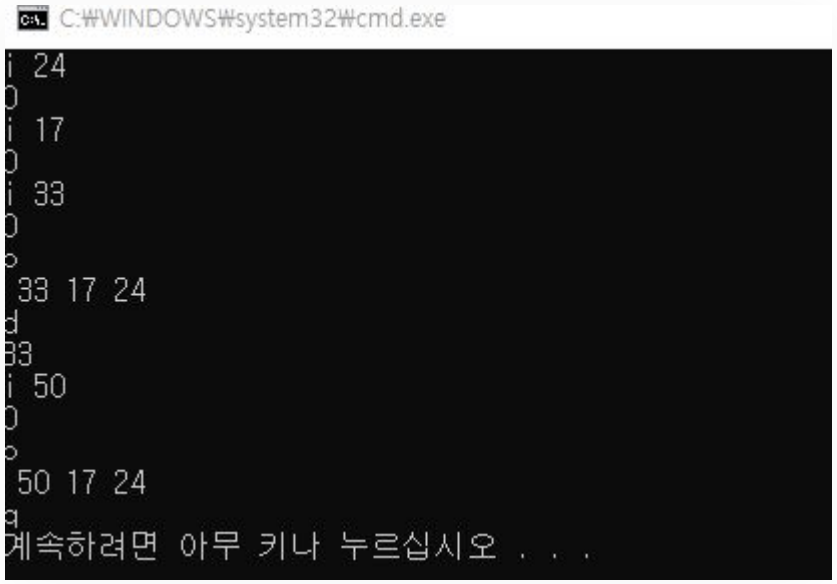
n(키 개수)위치에 key값을 삽입한 후, upHead 호출하고, n 갱신

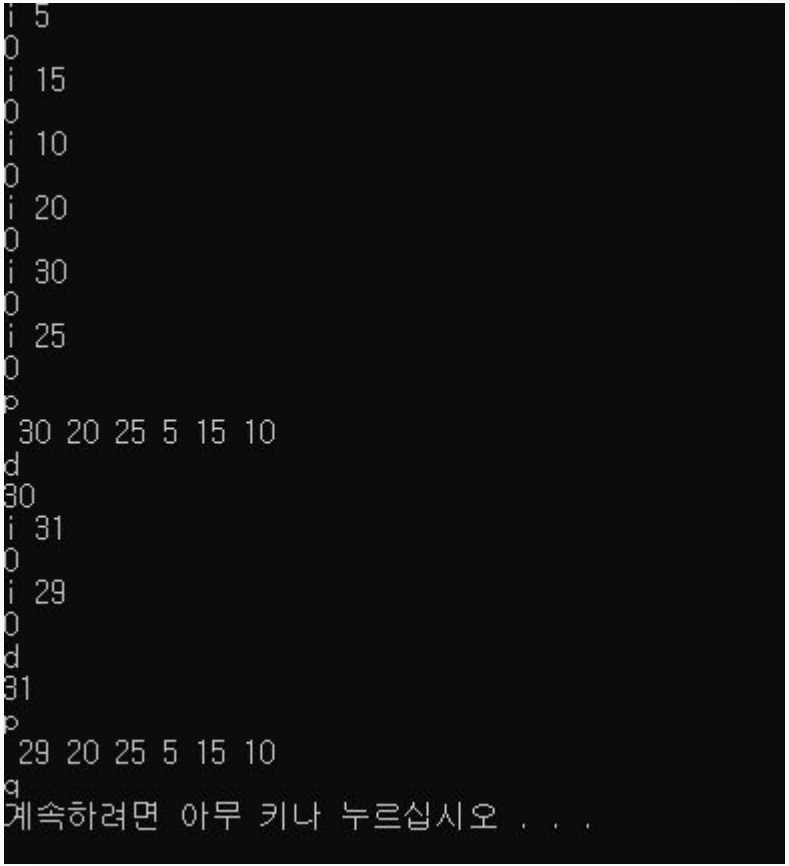
부모 또는 루트의 key와 big의 key를 swap

```c
int removeMax() {
    int key;
    key = H[1];
    H[1] = H[n--];
    downHeap(1);
    return key;
}
void printHeap() {
    for (int i = 1; i <= n; i++)
        printf(" %d", H[i]);
    printf("₩n");
}
int main() {
    char com;
    int key;
    while (1) {
        scanf("%c", &com);
        if (com == 'i') {
            scanf("%d", &key);
            insertItem(key);
            printf("O₩n");
        }
        else if (com == 'd') {
            printf("%d₩n", removeMax());
        }
        else if (com == 'p') {
            printHeap();
        }
        else {
            break;
        }
        getchar();
    }
```

루트 키를 보관하고 힙의 마지막 키를 루트로 함
n의 개수 조정 후 downHeap 호출
원래 루트 리턴

힙과 힙 정렬 실습문제 – 삽입식 힙 생성





힙과 힙 정렬 실습문제 – 상향식 힙 생성

[문제 2] 상향식 힙 생성

다음 프로그램을 작성해 상향식 힙 생성을 구현하라.

1) 이번엔 키들이 미리 한꺼번에 주어진다. 이들을 차례로 초기 배열에 저장한다.
2) 초기 배열에 저장된 키들을 상향식으로 재배치하여 힙을 생성한다. 상향식 힙 생성을 위한 **재귀 또는 비재귀** 방식의 알고리즘 가운데 어느 전략을 사용해도 좋다(나머지 전략도 나중에 작성해보기 바람).
3) 참고로 재귀, 비재귀 두 가지 방식 모두 O(n) 시간에 수행 가능하다(왜 그런지 복습하기 바람). 그렇게 되도록 작성해야 한다.

힌트: 문제 1 삽입식 힙 생성에서 이미 작성한 downHeap과 printHeap 함수를 그대로 사용하면 된다.

입출력 형식:

1) main 함수는 아래 형식의 표준입력으로 키들을 한꺼번에 입력받는다.

   입력 : 첫 번째 라인 : 키 개수
   두 번째 라인 : 키들

2) main 함수는 아래 형식의 표준출력으로 생성된 힙을 인쇄한다.

   출력 : 첫 번째 라인 : 힙 내용 (레벨 순서)

힙과 힙 정렬 실습문제 – 상향식 힙 생성

```
int H[100];
int n = 0;
void downHeap(int i) {
    int tmp;
    if ((n < (i * 2)) && (n < (i * 2 + 1)))
        return;
    int big = i * 2;
    if (n >= i * 2 + 1) {
        if (H[i * 2 + 1] > H[big])
            big = i * 2 + 1;
    }
    if (H[i] >= H[big])
        return;
    tmp = H[i];
    H[i] = H[big];
    H[big] = tmp;
    downHeap(big);
}
```

```
void rBuildHeap(int i) {
    if (i > n)
        return;
    rBuildHeap(2 * i);
    rBuildHeap(2 * i + 1);
    downHeap(i);
}
```

<재귀 버전>
루트 인덱스가 n(힙 크기)
보다 크면 return

현재 부트리의 왼쪽, 오른쪽
부트리의 힙 생성
현재 부트리 루트와 좌우
부트리 합친 힙 생성

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

<비재귀 버전>
마지막 내부 노드부터 루트까지
역방향으로 올라가면서 힙 생성

```
void printHeap() {
    for (int i = 1; i <= n; i++)
        printf(" %d", H[i]);
    printf("₩n");
}
int main() {
    scanf("%d", &n);
    for (int i = 1; i <= n; i++)
        scanf("%d", &H[i]);
    //rBuildHeap(1);
    buildHeap();
    printHeap();
    return 0;
}
```

힙과 힙 정렬 실습문제 – 상향식 힙 생성