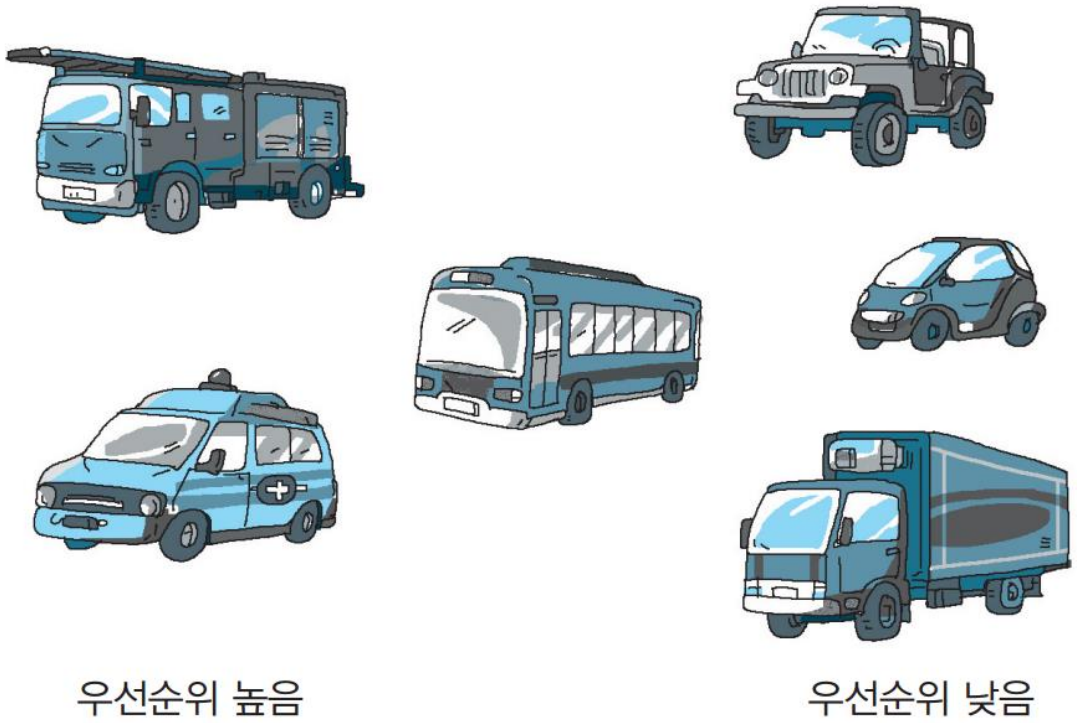
DATA
STRUCTURES
USING C++

CHAPTER 10

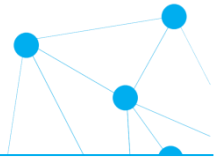
우선순위큐

실생활에서의 우선순위

- 도로에서의 자동차 우선순위



우선순위 큐

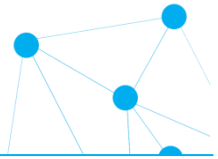


- Priority queue
  - 우선순위를 가진 항목들을 저장하는 큐
  - 우선 순위가 높은 데이터가 먼저 나가게 됨
  - 가장 일반적인 큐로 생각할 수 있음
    - 스택이나 큐를 우선순위 큐로 구현할 수 있다.

자료구조	삭제되는 요소
스택	가장 최근에 들어온 데이터
큐	가장 먼저 들어온 데이터
우선순위큐	가장 우선순위가 높은 데이터

- 응용분야
  - 시뮬레이션, 네트워크 트래픽 제어, OS의 작업 스케줄링 등

우선순위큐 ADT

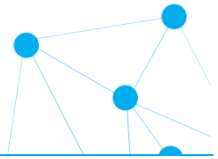


데이터: 우선순위를 가진 요소들의 모음
연산:

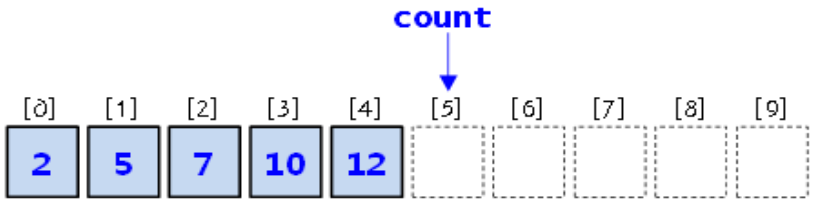
- insert(item): 우선순위 큐에 항목 item을 추가한다.
- remove(): 우선순위 큐로부터 가장 우선순위가 높은 요소를 삭제하고 이 요소를 반환한다.
- find(): 우선순위가 가장 높은 요소를 삭제하지 않고 반환한다.
- isEmpty(): 우선순위 큐가 공백 상태인지를 검사한다.
- isFull(): 우선순위 큐가 포화 상태인지를 검사한다.
- display(): 우선순위 큐의 모든 요소들의 출력한다.

- 가장 중요한 연산
  - insert 연산(요소 삽입), remove 연산(요소 삭제)
- 우선 순위 큐는 2가지로 구분
  - 최소 우선 순위큐, 최대 우선 순위큐

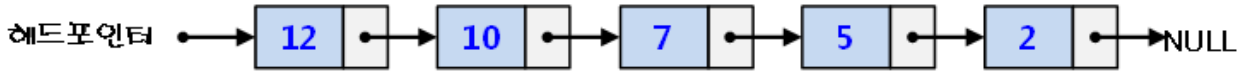
우선순위큐 구현방법



- 배열을 이용한 구현

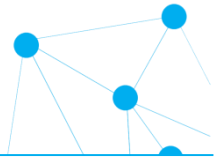


- 연결리스트를 이용한 구현



- 힙(heap)을 이용한 구현
  - 완전이진트리
  - 우선순위 큐를 위해 만들어진 자료구조
  - 일종의 반 정렬 상태를 유지

우선순위 큐 구현방법 비교



표현 방법	삽 입	삭 제
순서없는 배열	O(1)	O(n)
순서없는 연결 리스트	O(1)	O(n)
정렬된 배열	O(n)	O(1)
정렬된 연결 리스트	O(n)	O(1)
힙	**O(logn)**	**O(logn)**

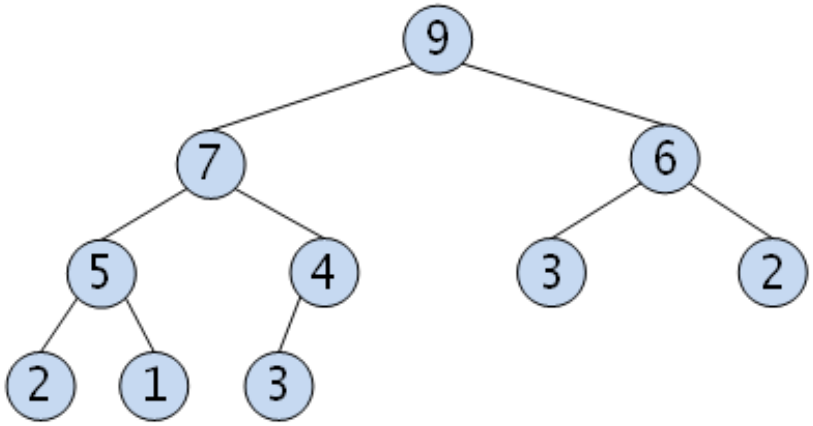
힙(heap)이란?

- Heap
  - 더미
  - 완전이진트리
  - 최대 힙, 최소 힙

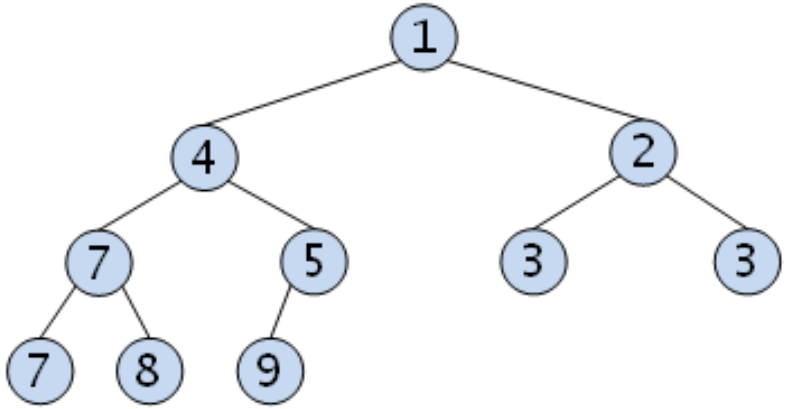
- 최대 힙(max heap)
  - 부모 노드의 키값이 자식 노드의 키값보다 크거나 같은 완전 이진 트리
- 최소 힙(min heap)
  - 부모 노드의 키값이 자식 노드의 키값보다 작거나 같은 완전 이진 트리

최대힙과 최소힙

- 최대 힙(max heap)
  - key(부모노드) ≥key(자식노드)

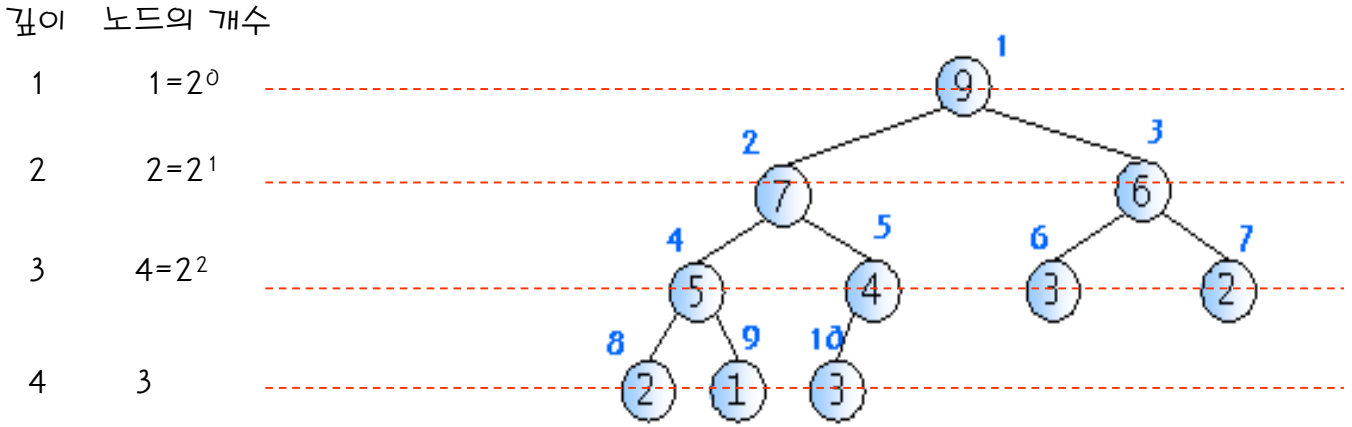


- 최소 힙(min heap)
  - key(부모노드) ≤key(자식노드)

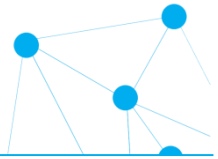


힙의 높이

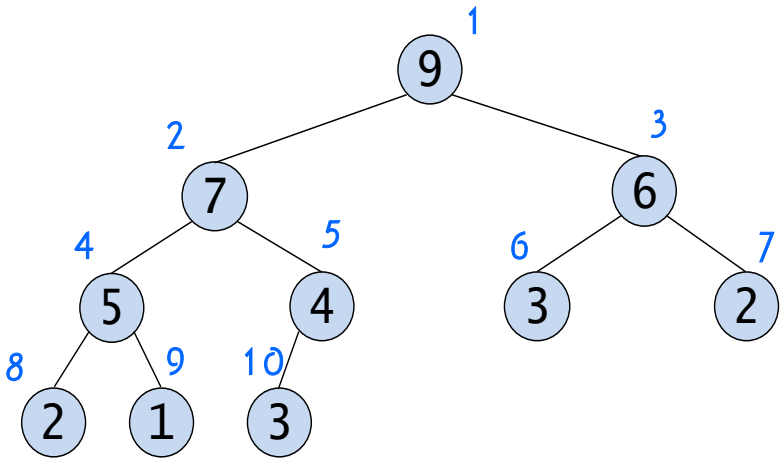
- n개의 노드를 가지고 있는 힙의 높이는 O(logn)
  - 힙은 완전이진트리
  - 마지막 레벨을 제외하고 각 레벨 i에 $2^i - 1$개의 노드 존재



힙의 구현: 배열을 이용



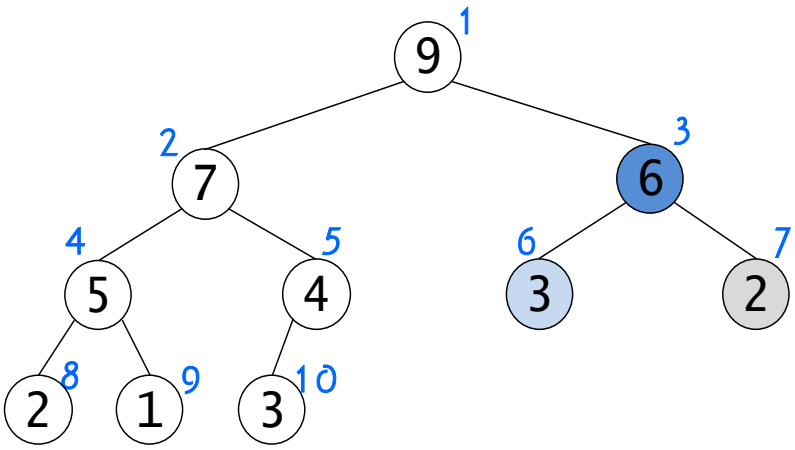
- 힙은 보통 배열을 이용하여 구현
  - 완전이진트리 ➔ 각 노드에 번호를 붙임 ➔ 배열의 인덱스

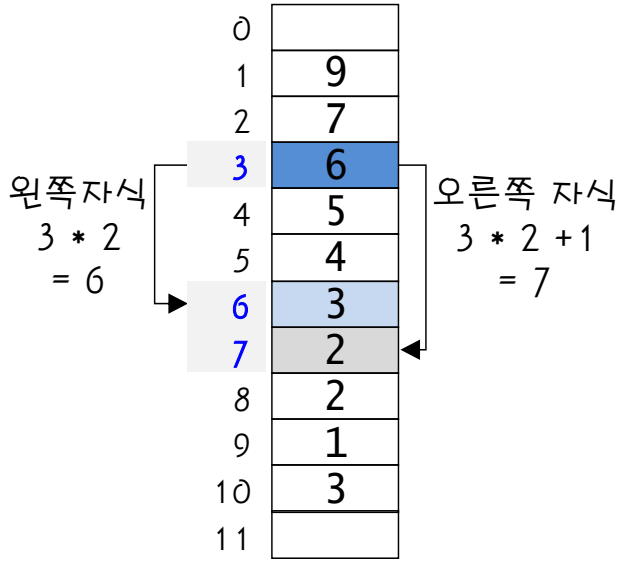


0	
1	9
2	7
3	6
4	5
5	4
6	3
7	2
8	2
9	1
10	3
11	

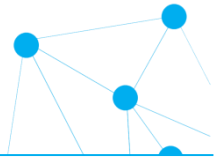
힙의 구현

- 부모노드와 자식노드의 관계
  - 왼쪽 자식의 인덱스 = (부모의 인덱스)*2
  - 오른쪽 자식의 인덱스 = (부모의 인덱스)*2 + 1
  - 부모의 인덱스 = (자식의 인덱스)/2

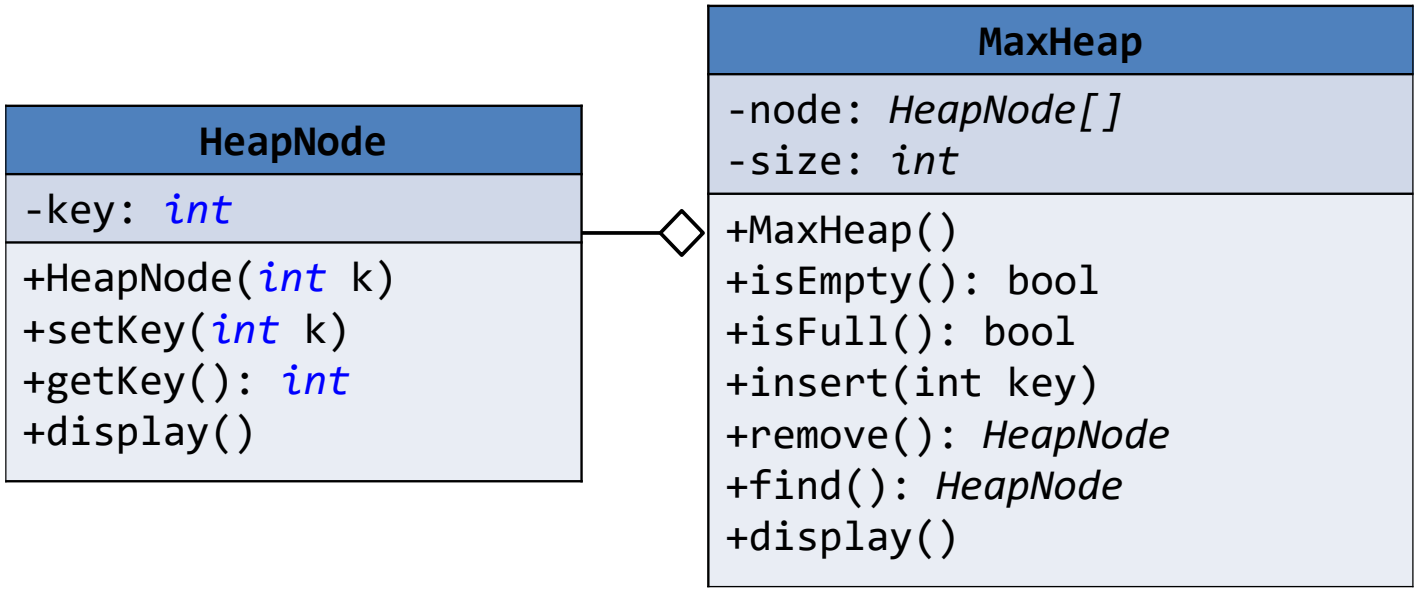




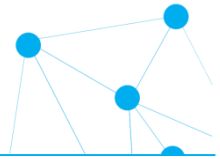
힙 구현의 기본틀



- 클래스 다이어그램
  - 힙 노드 클래스와 힙(Max Heap) 클래스

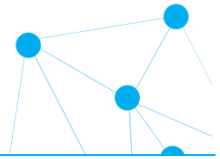


힙 노드 클래스



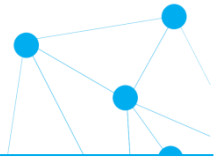
```
// 힙에 저장할 노드 클래스
class HeapNode
{
     int key; // Key 값
public:
     HeapNode( int k=0 ) : key(k) { }
     void setKey(int k) { key = k; }
     int getKey() { return key; }
     void display() { printf("%4d", key); }
};
```

최대 힙 클래스



```
// MaxHeap.h: 배열을 이용한 최대 힙 클래스
#include "HeapNode.h"
#define MAX_ELEMENT 200
class MaxHeap
{
     HeapNode node[MAX_ELEMENT];     // 요소의 배열
     int size;                       // 힙의 현재 요소의 개수
public:
     MaxHeap( ) : size(0) { }
     bool isEmpty() { return size == 0; }
     bool isFull() { return size == MAX_ELEMENT-1; }

     HeapNode& getParent(int i){ return node[i/2]; }  // 부모 노드
     HeapNode& getLeft(int i) { return node[i*2]; }   // 왼쪽 자식 노드
     HeapNode& getRight(int i) { return node[i*2+1];} // 오른쪽 자식 노드

     void insert(int key) {...}      // 삽입 함수: 프로그램 10.3
     HeapNode remove() {...}         // 삭제 함수: 프로그램 10.4
     HeapNode find() { return node[1]; }
};
```

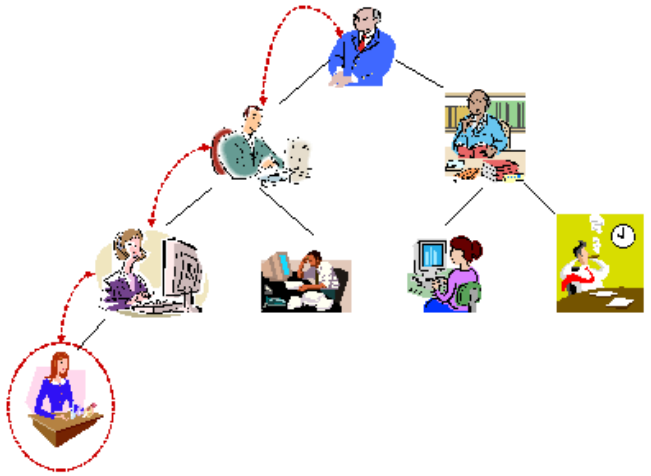
삽입 연산



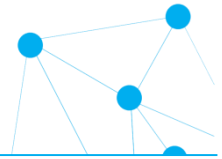
- Upheap
  - 회사에서 신입 사원이 들어오면 일단 말단 위치에 앉힘
  - 신입 사원의 능력을 봐서 위로 승진시킴

(1)힙에 새로운 요소가 들어 오면, 일단 새로운 노드를 히프의 마지막 노드에 이어서 삽입

(2)삽입 후에 새로운 노드를 부모 노드들과 교환해서 힙의 성질을 만족



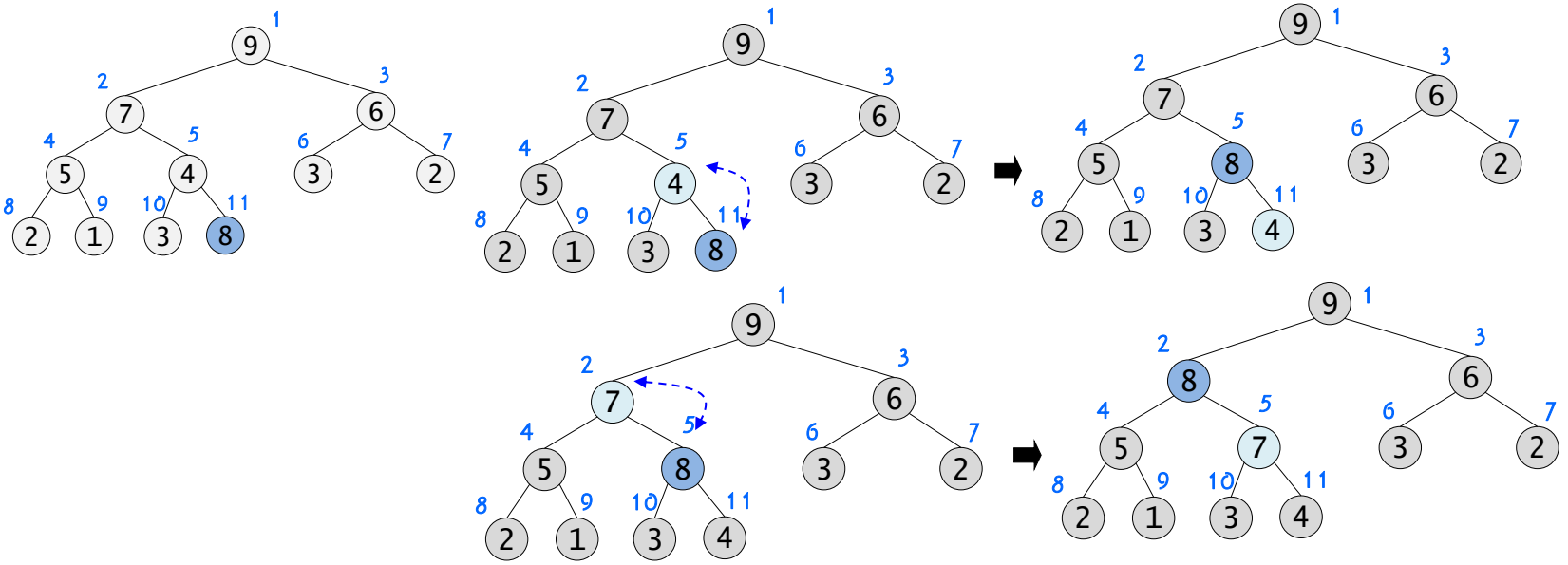
Upheap



- 삽입된 노드에서 루트까지의 경로에 있는 노드들을 비교/교환
- 히프의 성질을 복원
  - 키 k가 부모노드보다 작거나 같으면 upheap을 종료한다

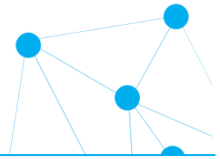
```
insert(key)

heapSize ← heapSize + 1;
i ← heapSize;
node[i] ← key;
while i ≠ 1 and node[i] > node[PARENT(i)] do
      node[i] ↔ node[PARENT(i)];
      i ← PARENT(i);
```

Upheap 과정



- 힙의 높이: O(logn) ➔ upheap연산은 O(logn)

삽입 함수

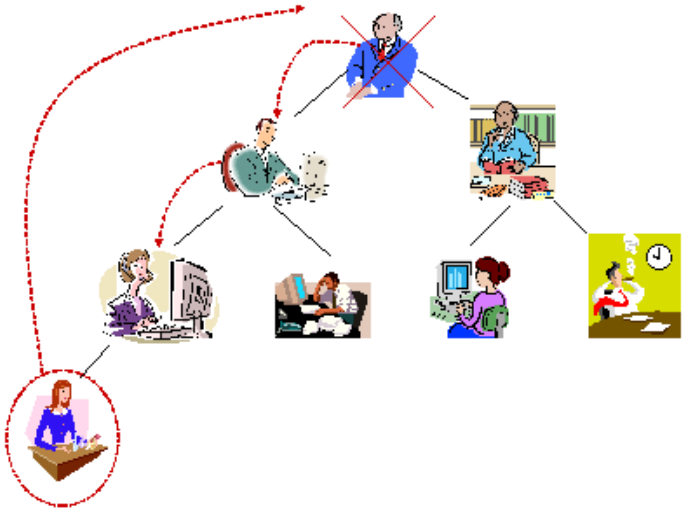


```
// 삽입 함수: 힙에 키값 key를 갖는 새로운 요소를 삽입한다.
void insert( int key )
{
        if( isFull() ) return;      // 힙이 가득 찬 경우
        int i = ++size;             // 증가된 힙 크기 위치에서 시작

        // 트리를 거슬러 올라가면서 부모 노드와 비교하는 과정
        while( i!=1                                 // 루트가 아니고
               && key>getParent(i).getKey()) {      // 부모보다 키값이 크면
                    node[i] = getParent(i);         // 부모를 끌어내림
                    i /= 2;                         // 한 레벨 위로 상승
         }
         node[i].setKey( key );                     // 최종 위치: 데이터 복사
}
```

삭제 연산

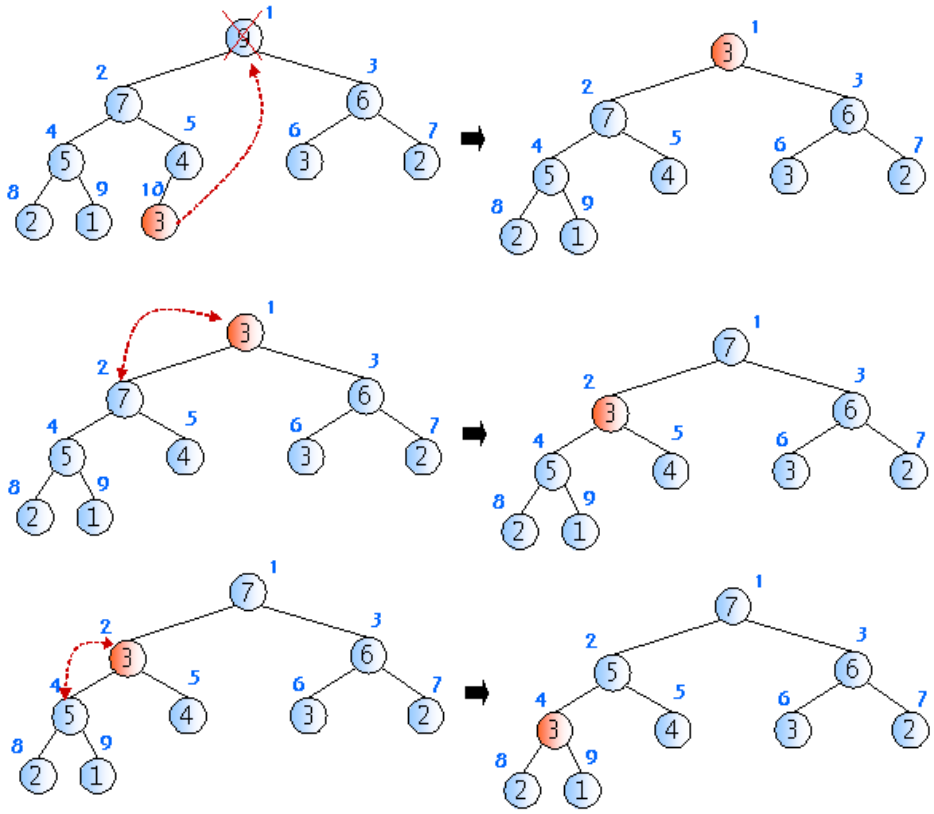
- 최대힙에서의 삭제 ➔ 항상 루트가 삭제됨
  - 가장 큰 키값을 가진 노드를 삭제하는 것

- 방법: downheap
  - 루트 삭제
  - 회사에서 사장의 자리가 비게 됨
  - 말단 사원을 사장 자리로 올림
  - 능력에 따라 강등 반복



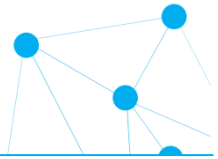
루트에서부터 단말노드까지의 경로에 있는 노드들을 교환하여 히프 성질을 만족시킨다.

Downheap 과정



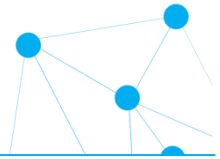
- 힙의 높이: O(logn) ➔ downheap연산은 O(logn)

Downheap 알고리즘



```
remove()

item ← A[1];
A[1] ← A[heapSize];
heapSize ← heapSize-1;
i ← 2;
while i ≤ heapSize do
      if i < heapSize and A[LEFT(i)] > A[RIGHT(i)]
         then largest ← LEFT(i);
         else largest ← RIGHT(i);
      if A[PARENT(largest)] > A[largest]
         then break;
      A[PARENT(largest)] ↔ A[largest];
      i ← LEFT(largest);return item;
```

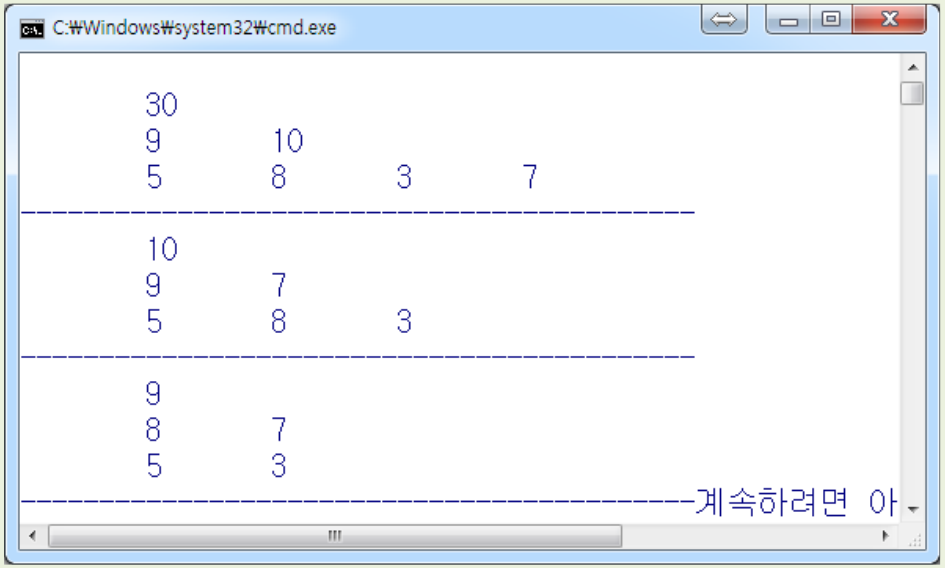
삭제 함수



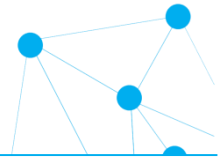
```
HeapNode remove() {
    if( isEmpty()) error();
    HeapNode item = node[1];     // 루트노드(꺼낼 요소)
    HeapNode last = node[size--]; // 힙의 마지막노드
    int parent = 1;              // 마지막 노드의 위치를 루트로 생각함
    int child = 2;               // 루트의 왼쪽 자식 위치
    while( child <= size ){      // 힙 트리를 벗어나지 않는 동안
        if( child < size
         && getLeft(parent).getKey() < getRight(parent).getKey())
                child++;
        if( last.getKey() >= node[child].getKey() ) break;

        // 한 단계 아래로 이동
        node[parent] = node[child];
        parent = child;
        child *= 2;
    }
    node[parent] = last;     // 마지막 노드를 최종 위치에 저장
    return item;             // 루트 노드 반환
}
```

전체 프로그램

```
void main() {
        MaxHeap heap;

        heap.insert(10);
        heap.insert( 5);
        heap.insert(30);
        heap.insert( 8);
        heap.insert( 9);
        heap.insert( 3);
        heap.insert( 7);
        heap.display( );

        // 삭제 테스트
        heap.remove();
        heap.display();
        heap.remove();
        heap.display();
        printf("\n");
}
```



힙의 복잡도 분석



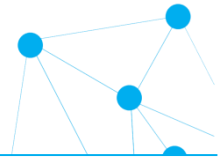
- 삽입 연산에서 최악의 경우
  - 루트 노드까지 올라가야 하므로 트리의 높이에 해당하는 비교 연산 및 이동 연산이 필요하다.

  ➔ O(logn)

- 삭제연산 최악의 경우
  - 가장 아래 레벨까지 내려가야 하므로 역시 트리의 높이 만큼의 시간이 걸린다.

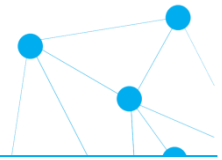
  ➔ O(logn)

히프정렬



- 힙을 이용하면 정렬 가능: 힙 정렬
  - 먼저 정렬해야 할 n개의 요소들을 최대 힙에 삽입
  - 한번에 하나씩 요소를 힙에서 삭제하여 저장하면 된다.
  - 삭제되는 요소들은 값이 증가되는 순서(최소힙의 경우)
- 시간 복잡도: O(nlogn)
  - 하나의 요소의 삽입 삭제가 O(logn)
  - 요소의 개수가 n개 ➔ O(nlogn)
- 특히 유용한 경우
  - 전체의 정렬이 아니라 가장 큰 값 몇 개만 필요할 때이다.

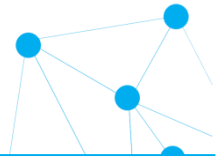
히프정렬



```
class HeapNode { ... };          // 프로그램 10.1의 HeapNode 클래스
class MaxHeap { ... };           // 프로그램 10.2의 MaxHeap 클래스

// 우선순위 큐인 힙을 이용한 정렬
void heapSort( int a[], int n)
{
    MaxHeap h;
    for(int i=0 ; i<n ; i++ )
        h.insert(a[i]);

    // 최대 힙에서는 삭제시 가장 큰 값이 반환되므로
    // 오름차순으로 정렬하기 위한 삭제한 항목을 배열의 끝부터 앞으로 채워 나감
    for(int i=n-1 ; i>=0 ;i-- )
        a[i] = h.remove()->getKey();
}
```

```
C:₩WINDOWS₩system32₩cmd.exe
   정렬 전:  41 67 34  0 69 24 78 58 62 64
   정렬 후:   0 24 34 41 58 62 64 67 69 78
계속하려면 아무 키나 누르십시오 . . .
```

STL의 우선순위큐



- STL에서 우선순위큐를 제공함
  - #include <queue>
  - using namespace std;

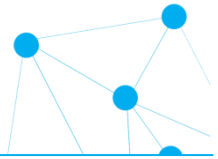
- 최대힙과 최소힙이 약간 다름
  - 최대힙 객체 생성 (less than operator 사용)

```
priority_queue< int > maxHeap;
```

  - 최소힙 객체 생성 (greater than operator 사용)

```
#include <functional>
...
priority_queue<int,vector<int>,greater<int>> minHeap;
```

STL 사용 힙정렬 (오름차순)



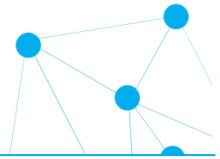
```
#include <queue>
#include <functional>
using namespace std;

void heapSortInc( int a[], int n)
{
    priority_queue<int, vector<int>, greater<int>> minHeap;
    for(int i=0 ; i<n ; i++ )
        minHeap.push(a[i]);

    // MinHeap을 이용해 오름차순으로 정렬하기 위한 반복문
    for(int i=0 ; i<n ;i++ )
        a[i] = minHeap.top();
        minHeap.pop();
    }
}
```

허프만 코드

- 이진 트리는 각 글자의 빈도가 알려져 있는 메시지의 내용을 압축하는데 사용될 수 있음
- 이런 종류의 이진트리 ➔ 허프만 코딩 트리

A	80
B	16
C	32
D	36
E	123
F	22
G	16
H	51
I	71
...	
Z	1

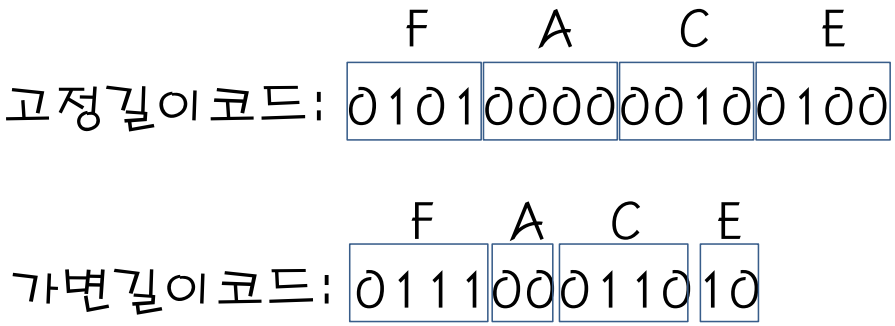
글자의 빈도수



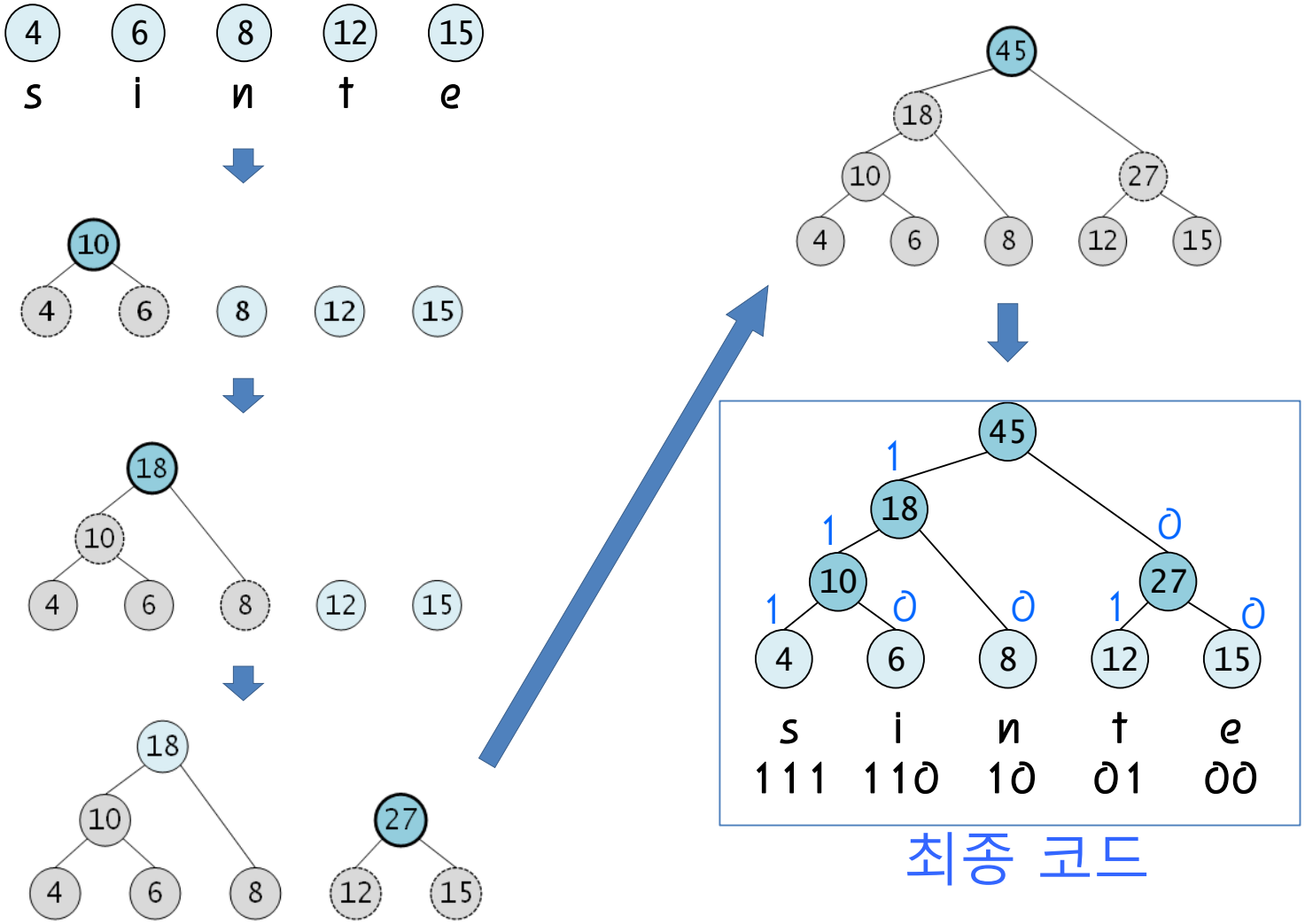
- 빈도수가 알려진 문자에 대한 고정길이코드와 가변길이코드의 비교

글자	빈도수	고정길이코드			가변길이코드		
		코드	비트수	전체 비트수	코드	비트수	전체 비트수
A	17	0000	4	68	00	2	34
B	3	0001	4	12	11110	5	15
C	6	0010	4	24	0110	4	24
D	9	0011	4	36	1110	4	36
E	27	0100	4	108	10	2	54
F	5	0101	4	20	0111	4	20
G	4	0110	4	16	11110	5	20
H	13	0111	4	52	010	3	39
I	15	1000	4	60	110	3	45
J	1	1001	4	4	11111	5	5
합계	100			**400**			**292**

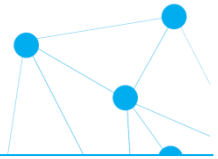
- 코드 읽기



허프만 코드 생성 절차


4 6 8 12 15
s i n t e
10
4 6 8 12 15
18
10
4 6 8 12 15
18
10 27
4 6 8 12 15
45
18
10 27
4 6 8 12 15
45
1 0
1 18 0
1 10 0 0 27 0
4 6 8 12 15
s i n t e
111 110 10 01 00
최종 코드

허프만 코드 프로그램



```
void MakeTree( int freq[], int n )
{
    MinHeap heap;
    for(int i=0;i<n;i++)
        heap.insert( freq[i] );

    for(int i=1;i<n;i++){
        HeapNode& e1 = heap.remove();              // 최소 노드 삭제
        HeapNode& e2 = heap.remove();              // 다음 최소 노드 삭제
        heap.insert(e1.getKey() + e2.getKey());    // 합한 노드 추가
        printf( " (%d+%d)\n", e1.getKey(), e2.getKey() );
    }
}
void main()
{
    int freq[] = { 15, 12, 8, 6, 4 };
    MakeTree( freq, 5 );
}
```