

자료구조의 기초

Lab 4. Heap Sort / Merge Sort

Taewhan Kim

Lab Introduction

■ Visual Studio

- 모랩에 설치된 **Visual Studio 2017** 사용
- 개인 노트북에 설치된 Visual Studio 사용 가능

■ 출석

- **출석부에 서명 + eTL에 실습 코드 업로드**로 출석 체크
- 둘 중 하나라도 누락 시 **결석** 처리

Priority Queue

■ Priority Queue

- 각 Element가 **우선순위**를 갖는 자료 구조
- 우선순위가 높은 Element는 낮은 Element보다 먼저 처리됨
- 우선순위가 동일한 경우, Queue에서의 순서에 따라 처리됨

■ Priority Queue 구현 방법

- Array
- Linked List
- Heap
 - Array/Linked List로 구현하면 Enqueue/Dequeue에 $O(n)$ 의 Complexity 필요
 - Priority Queue 구현 시 주로 Heap의 형태로 구현

Heap

- Heap

- Complete Binary Tree

- Max-heap

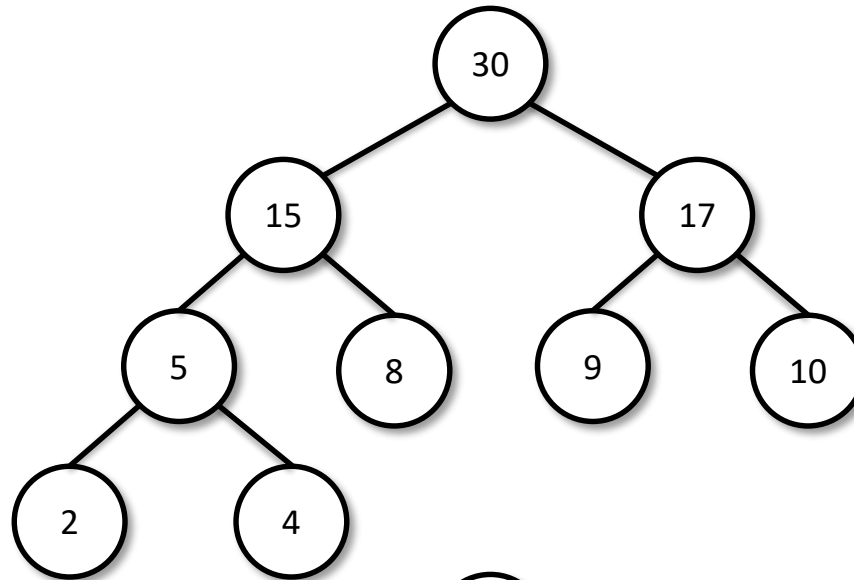
- 부모 노드에 저장된 값이 자식 노드에 저장된 값보다 크거나 같아야 함
 - Root 노드에 저장된 값이 전체 노드 중 가장 큰 값

- Min-heap

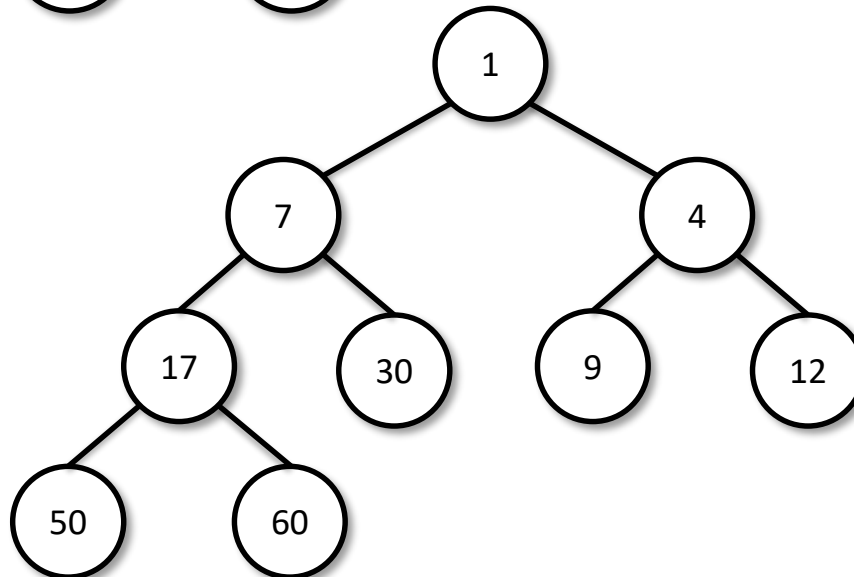
- 부모 노드에 저장된 값이 자식 노드에 저장된 값보다 작거나 같아야 함
 - Root 노드에 저장된 값이 전체 노드 중 가장 작은 값

Heap

- Max-heap



- Min-heap

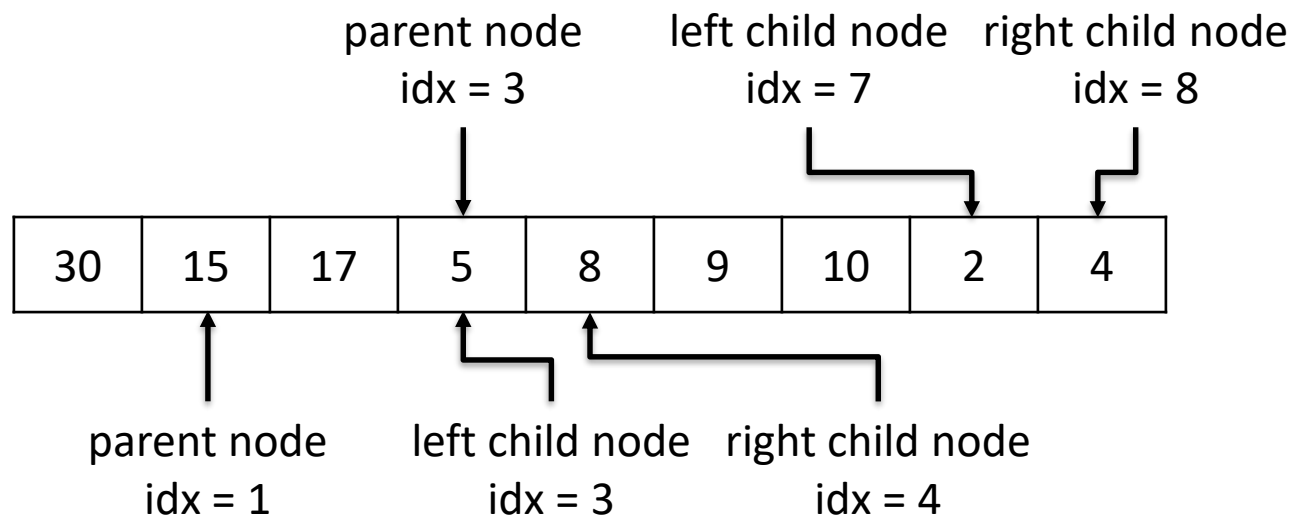
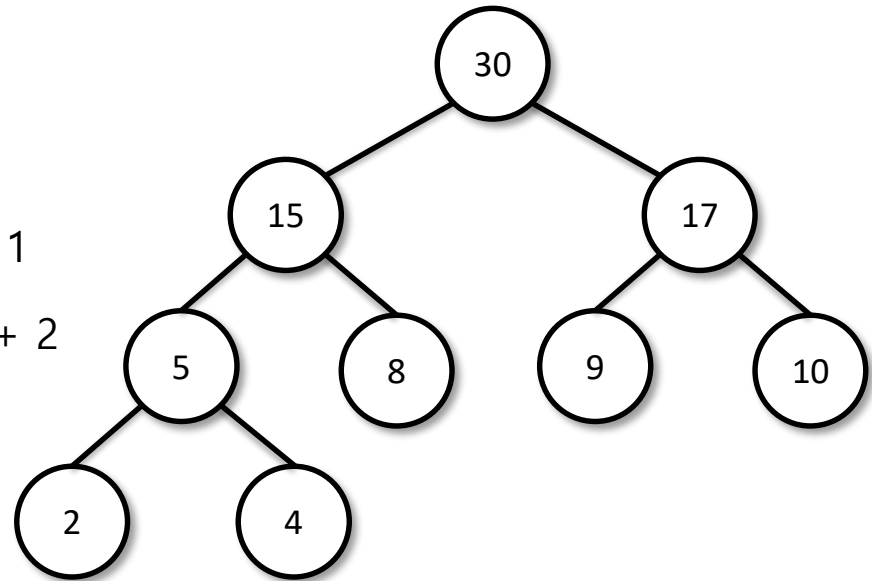


Heap Implementation

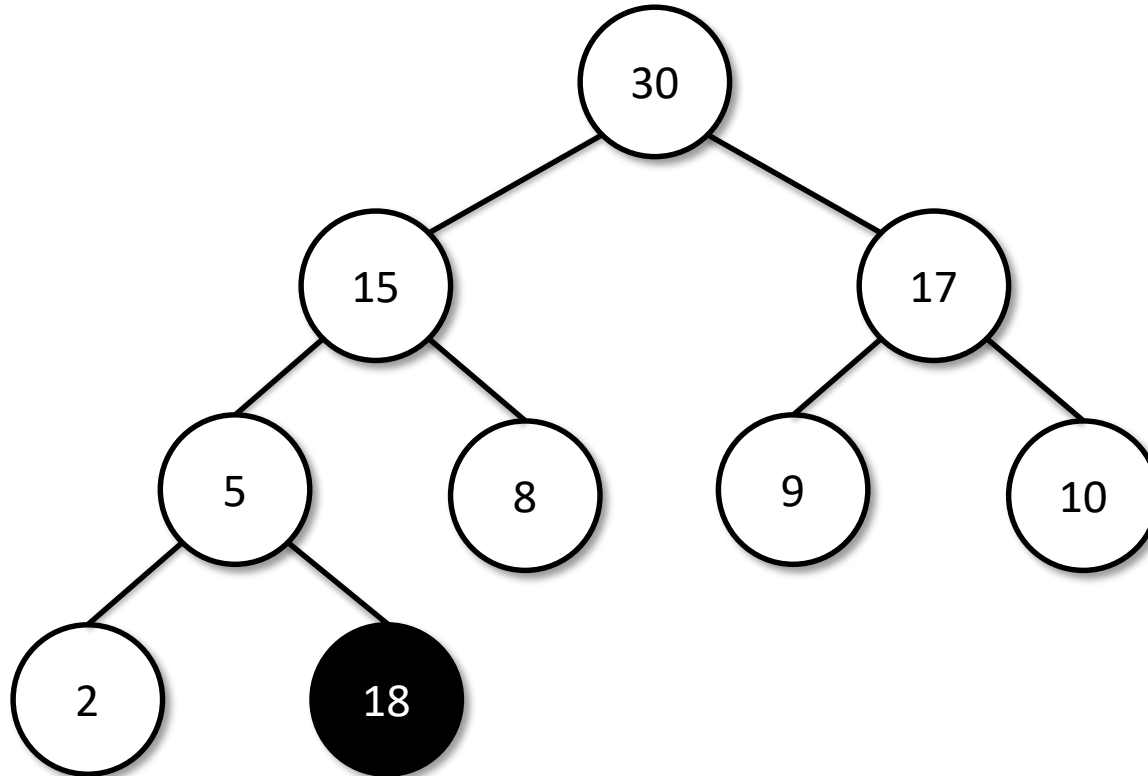
■ Array-based Implementation

□ Index

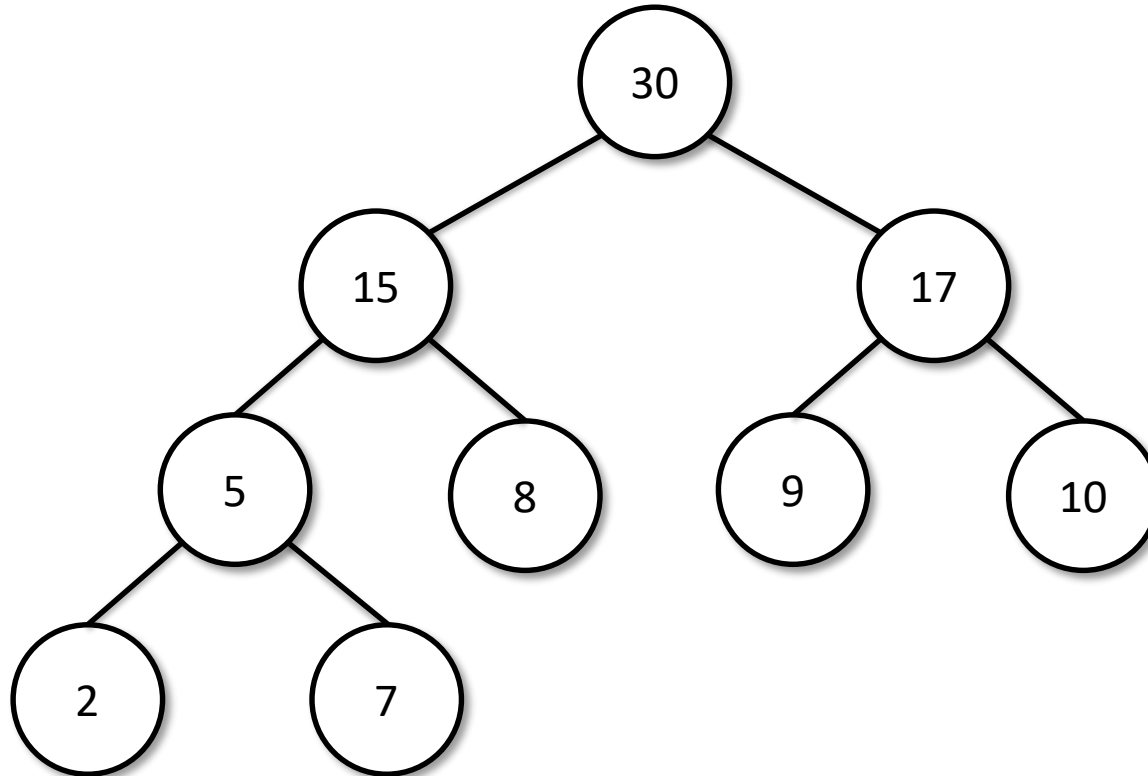
- Left Child : $(\text{Parent의 idx}) \times 2 + 1$
- Right Child : $(\text{Parent의 idx}) \times 2 + 2$



Node Insertion



Node Deletion



Heap Class

heap.cpp

```
#include <iostream>
using namespace std;

template <class T>
class heap {
private:
    T * arr;
    int capacity;
    int size;
public:
    heap(int c) : capacity(c), size(0), arr(new T[c]) {
        for (int i = 0; i < c; i++)
            arr[i] = NULL;
    }
    void swap(int pos_a, int pos_b);
    void insertData(T data);
    void deleteData();
    T getRoot();
    void print();
};
```

- Class heap
 - arr
 - heap 데이터 저장하는 배열
 - capacity
 - heap 배열 크기
 - size
 - 현재 heap에 저장된 데이터 수

Heap Class

heap.cpp

```
template <class T>
void heap<T>::swap(int a, int b) {
    T temp = arr[a];
    arr[a] = arr[b];
    arr[b] = temp;
}
```

```
template <class T>
T heap<T>::getRoot() {
    return arr[0];
}
```

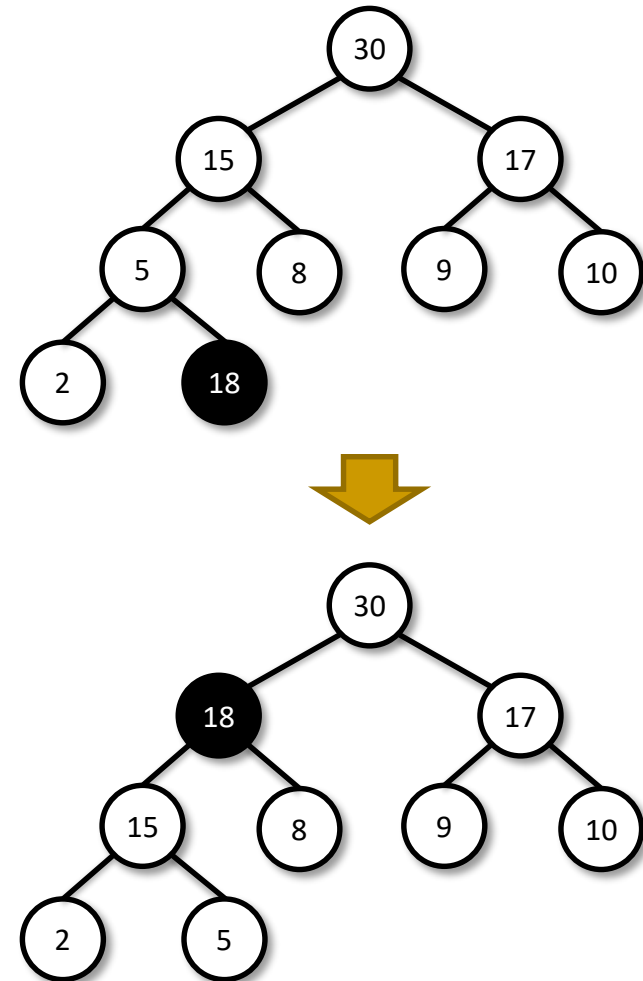
```
template <class T>
void heap<T>::print() {
    for (int i = 0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
}
```

- swap()
 - heap의 두 노드 교환
- getRoot()
 - root 노드 반환
- print()
 - heap에 저장된 데이터 출력

Node Insertion

heap.cpp

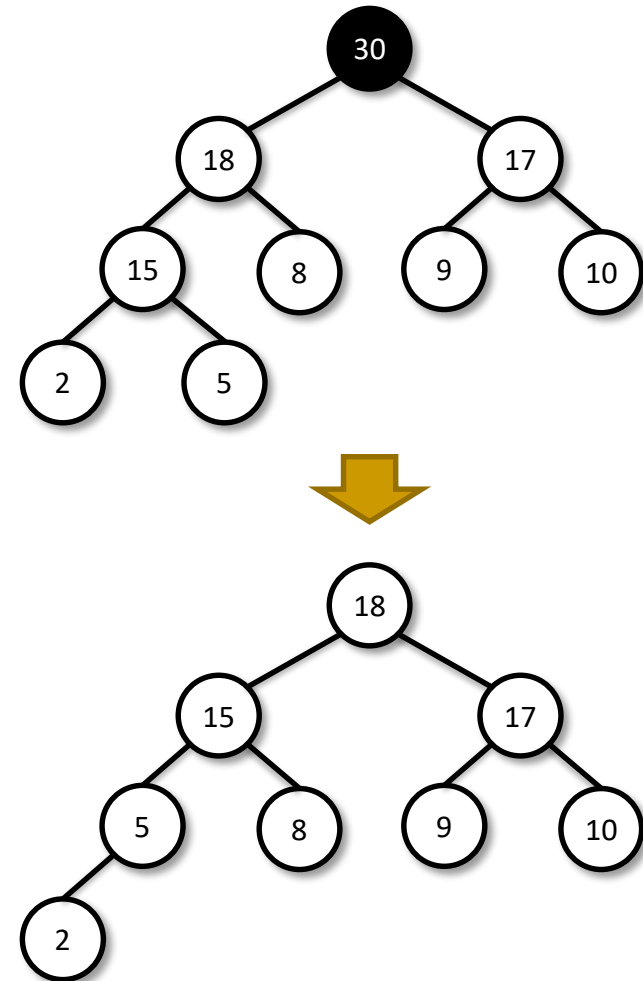
```
template <class T>
void heap<T>::insertData(T data) {
    if (capacity <= size) {
        cout << "heap is full" << endl;
        return;
    }
    int idx = size;
    int parent = (size - 1) / 2;
    arr[size++] = data;
    while (idx != 0) {
        if (arr[idx] > arr[parent]) {
            swap(idx, parent);
            idx = parent;
            parent = (idx - 1) / 2;
        }
        else break;
    }
}
```



Node Deletion

heap.cpp

```
template <class T>
void heap<T>::deleteData() {
    if (size == 0) {
        cout << "heap is empty" << endl;
        return;
    }
    swap(size - 1, 0);
    arr[size - 1] = NULL;
    size--;
    int idx = 0; int left = 1; int right = 2;
    while (left < size) {
        int max = idx;
        if (arr[left] > arr[max])
            max = left;
        if (right < size && arr[right] > arr[max])
            max = right;
        if (max != idx) {
            swap(idx, max);
            idx = max;
            left = 2 * idx + 1;
            right = 2 * idx + 2;
        }
        else break;
    }
}
```



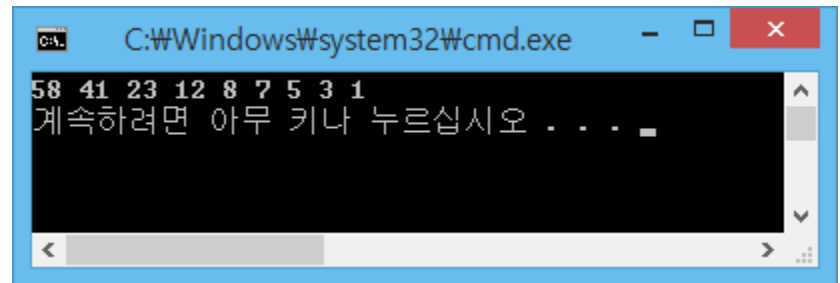
TODO : Heap Sort

- TODO : Heap Class를 이용하여 Heap Sort 구현
 - 데이터를 Heap에 넣고 빼는 방법으로 정렬
 - 수행 후 배열의 데이터는 내림차순으로 정렬

heap.cpp

```
template <class T>
void heapSort(T* arr, int capacity) {
    // TODO
}

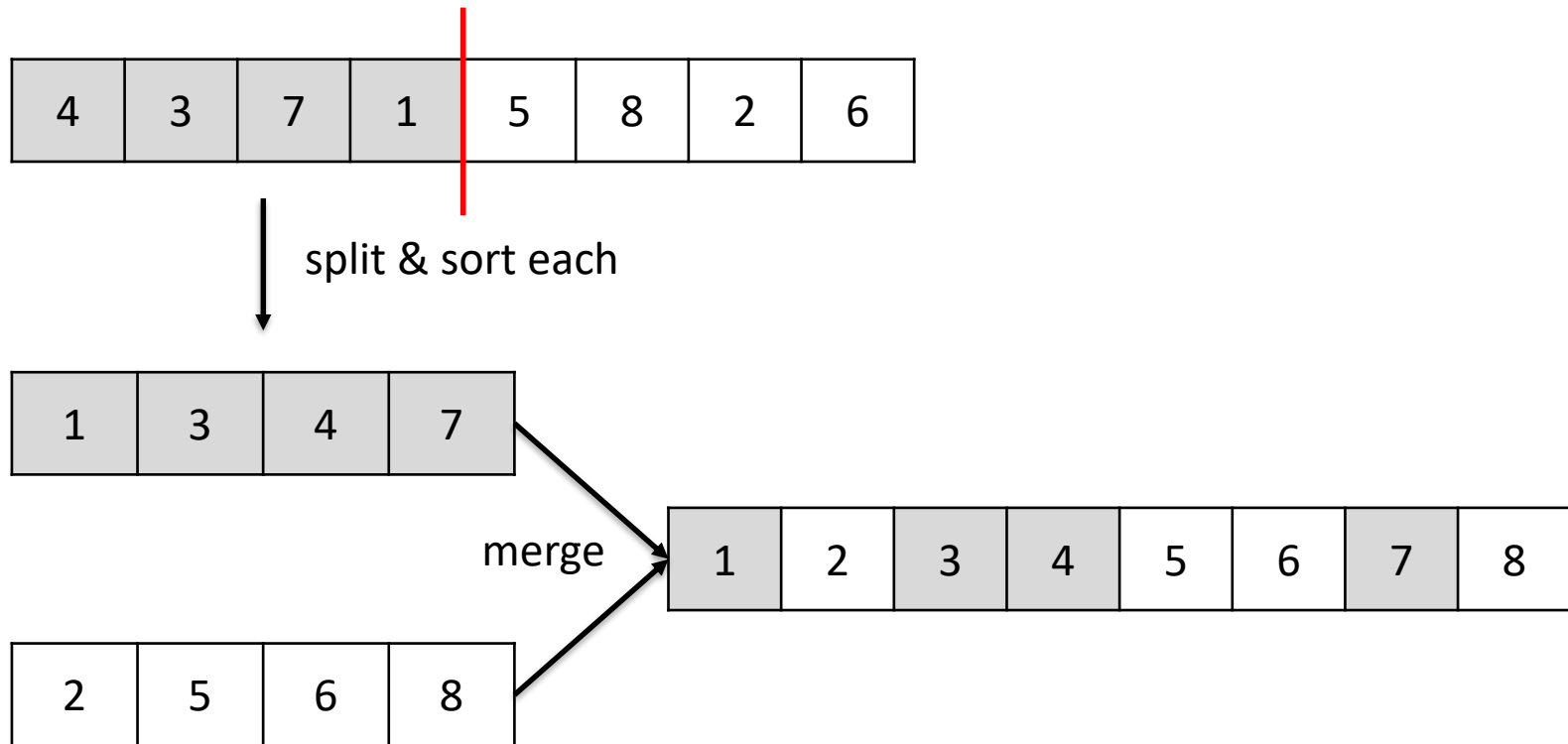
int main() {
    int a[9] = { 5, 7, 12, 3, 58, 8, 41, 23, 1 };
    heapSort<int>(a, 9);
    for (int i = 0; i < 9; i++)
        cout << a[i] << " ";
    cout << endl;
    return 0;
}
```



Merge Sort

■ Merge Sort

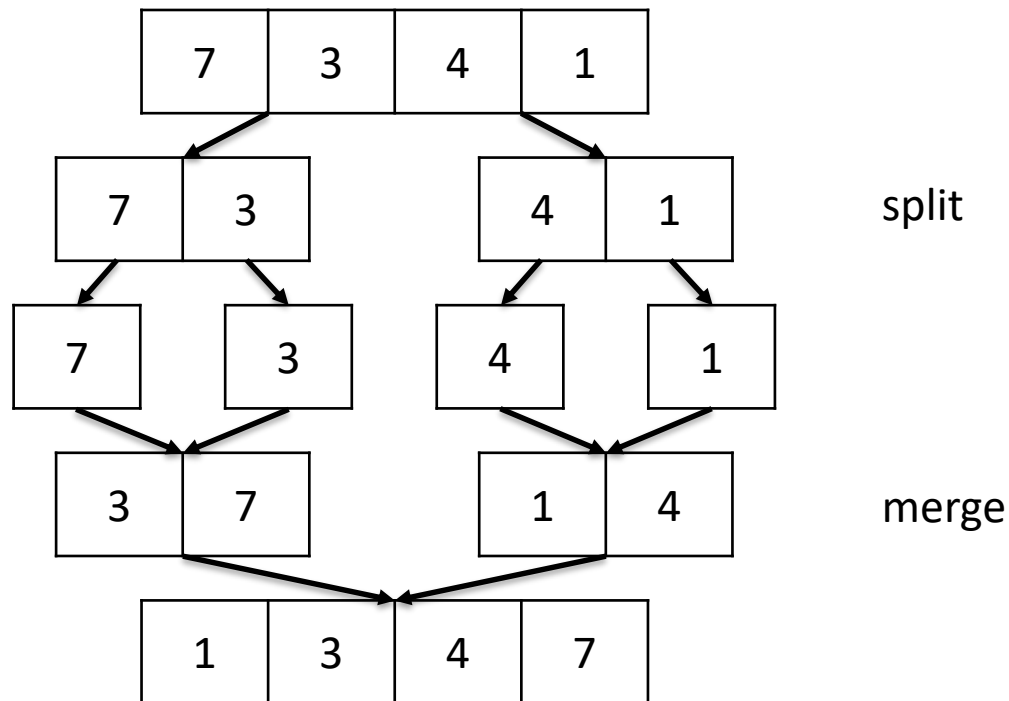
- 정렬된 두 부분 배열을 하나의 정렬된 배열로 합치는 과정을 반복하여 전체를 정렬된 형태로 만드는 방법



Merge Sort

- Top-down Approach

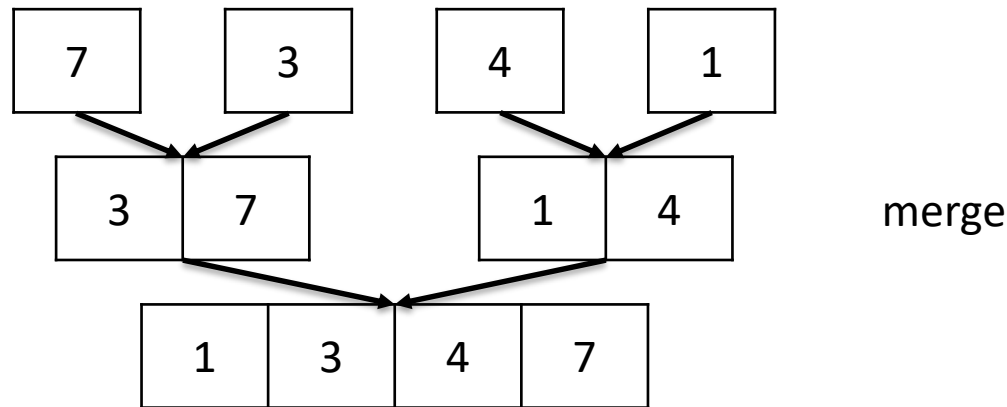
- Recursion 이용
- merge 호출 시 배열을 앞뒤로 절반으로 쪼개어 정렬 후 정렬된 두 배열을 하나의 정렬된 배열로 합치는 과정 반복



Merge Sort

- Bottom-up Approach

- Recursion 이용하지 않음
- 크기가 1인 배열부터 시작하여,
정렬된 두 배열을 하나의 정렬된 배열로 합치는 과정 반복



Merge Sort – Bottom-up Approach

merge.cpp

```
template <class T>
void copyArray(T* b, T* a, int n) {
    for (int i = 0; i < n; i++)
        a[i] = b[i];
}

template <class T>
void merge(T* a, int l, int c, int r, T* b) {
    int i = l, j = c;
    for (int k = l; k < r; k++) {
        if (i < c && (j >= r || a[i] <= a[j]))
            b[k] = a[i++];
        else
            b[k] = a[j++];
    }
}
```

- copyArray()
 - 배열 b의 데이터를 a에 복사
- merge()
 - a[l] ~ a[c-1] 의 부분 배열과 a[c] ~ a[r-1] 의 부분 배열을 하나의 정렬된 배열로 merge 하여 b에 저장

Merge Sort – Bottom-up Approach

merge.cpp

```
template <class T>
void mergeSort_bottomup(T* a, int n) {
    T* b = new T[n];
    for (int w = 1; w < n; w *= 2) {
        for (int i = 0; i < n; i += 2 * w) {
            merge(a, i, __min(i + w, n), __min(i + 2 * w, n), b);
        }
        copyArray(b, a, n);
    }
}
```

- mergeSort_bottomup()
 - 크기가 1인 배열부터 시작하여 2, 4, ... 인 배열로 2개씩 merge 수행

TODO : Merge Sort – Top-down Approach

merge.cpp

```
template <class T>
void split_merge_topdown(T* b, int l, int r, T* a) {
    // TODO
}

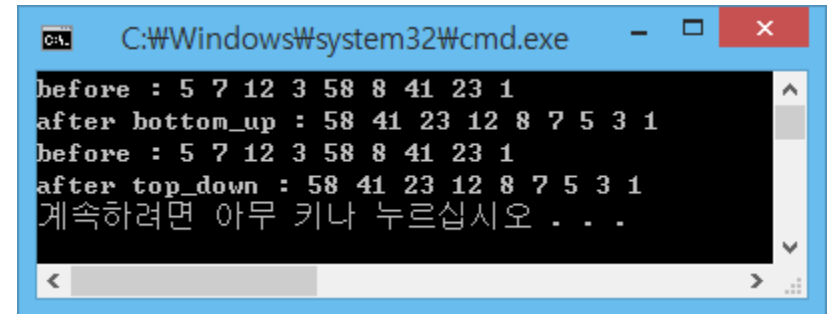
template <class T>
void mergeSort_topdown(T* a, int n) {
    T* b = new T[n];
    copyArray(a, b, n);
    split_merge_topdown(b, 0, n, a);
}
```

- split_merge_topdown()
 - 배열을 계속 잘게 쪼갠 후
잘게 쪼갠 배열에 대해
merge 수행
- mergeSort_topdown()
 - Top-down Approach를
사용하여 배열 정렬
 - Bottom-up Approach와
결과가 동일해야 함

TODO : Merge Sort – Top-down Approach

merge.cpp

```
void main() {  
    int a[9] = { 5, 7, 12, 3, 58, 8, 41, 23, 1 };  
    int* b = new int[9];  
    copyArray(a, b, 9);  
  
    cout << "before : ";  
    for (int i = 0; i < 9; i++) cout << a[i] << " ";  
    cout << endl;  
  
    mergeSort_bottomup<int>(a, 9);  
  
    cout << "after bottom_up : ";  
    for (int i = 0; i < 9; i++) cout << a[i] << " ";  
    cout << endl;  
  
    cout << "before : ";  
    for (int i = 0; i < 9; i++) cout << b[i] << " ";  
    cout << endl;  
  
    mergeSort_topdown<int>(b, 9);  
  
    cout << "after top_down : ";  
    for (int i = 0; i < 9; i++) cout << b[i] << " ";  
    cout << endl;  
}
```



The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The output of the program is displayed as follows:

```
before : 5 7 12 3 58 8 41 23 1  
after bottom_up : 58 41 23 12 8 7 5 3 1  
before : 5 7 12 3 58 8 41 23 1  
after top_down : 58 41 23 12 8 7 5 3 1  
계속하려면 아무 키나 누르십시오 . . .
```

Code Submission

■ 코드 제출

- 구현한 코드를 다음과 같이 압축
 - 제출할 코드 : heap.cpp, merge.cpp
 - 파일명 : lab4_홍길동_2017-10000.zip
- 오늘 (2018년 5월 21일) **오후 11시**까지 eTL에 제출
- 제출된 코드는 따로 채점하지 않음

■ 출석

- **출석부에 서명 + eTL에 실습 코드 업로드**로 출석 체크
- 둘 중 하나라도 누락 시 **결석** 처리