알고리즘 3주차
: 힙과힙정렬2,합병정렬,퀵정렬

2조 21011948 신아진, 21011954 이은서

00

목차

힙과 힙정렬(2) 실습문제

힙 정렬(heap sort)은 **힙**을 이용한 정렬이다. 무순 리스트(unordered list)에 대한 힙 정렬은 두 단계로 진행된다. **1단계**는 **힙 생성 단계**로서 초기 리스트로부터 **최대힙**을 생성한다. **2단계**는 **힙 정렬 단계**로서 전 단계에서 생성된 최대힙으로부터 **오름차순 정렬**을 수행한다. 힙 정렬은 O(n log n) 시간에 수행한다.

[문제 1] 힙 정렬 – 유일 키

다음 조건에 맞추어 힙 정렬 프로그램을 작성하라.

1) **순차힙**으로 구현한다. 즉, 배열을 이용한 순차트리 형식으로 힙을 저장한다.
2) 연산 효율을 위해 배열 인덱스 0 위치는 사용하지 않고 비워둔다.
3) 데이터구조를 단순화하기 위해 힙의 항목으로써 (**키**, **원소**) 쌍에서 원소를 생략하고 **키만** 저장하는 것으로 한다.
4) 키들은 중복이 없는 1 이상의 정수로 전제한다 – 즉, 중복 키에 대한 처리는 불필요하다.
5) 최대 키 개수 < 100 으로 전제한다.
6) 1단계(힙 생성 단계)에서 **삽입식** 또는 **상향식** 가운데 어떤 방식을 사용해도 좋다.
7) O(n lon n) 시간, O(1) 공간에 수행해야 한다.

inPlaceHeapSort() 함수
printArray() 함수
downHeap(i) 함수
insertItem(key) 함수
upHeap(i) 함수

힙과 힙정렬(2) 실습문제

```
int H[100];
int n;      //힙의 크기(총 키 개수)
```

필요 데이터구조 선언

```
int main(void) {
    int num, k;
    scanf("%d", &k);
    getchar();
    for (int i = 1; i <= k; i++) {
        scanf("%d", &num);
        insertItem(num);
    }
    inPlaceHeapSort();
    printArray();

    return 0;
}
```

main 함수

1. 초기 배열 값을 입력받음
2. 힙 정렬의 1단계 2단계 수행
3. 오름차순 정렬 배열을 인쇄하고 종료

힙과 힙정렬(2) 실습문제

```
void inPlaceHeapSort() {
    for (int i = n; i >= 2; i--) {
        int tmp = H[1];
        H[1] = H[i];
        H[i] = tmp;
        downHeap(1, i - 1);
    }
}
```

inPlaceHeapSort() 함수

: n개의 키로 구성된 무순 배열을 제자리 힙 정렬

```
void printArray() {
    for (int i = 1; i <= n; i++)
        printf(" %d", H[i]);
    printf("₩n");
}
```

printArray() 함수

: 배열에 저장된 키들을 차례로 인쇄

힙과 힙정렬(2) 실습문제

```
void downHeap(int i, int last) {
    if ((last < i * 2) && (last < i * 2 + 1))       // 단말노드인경우( 크기가 자식노드 위치보다 작은 경우 )
        return;
    int bigger = i * 2;
    if (last >= i * 2 + 1) {        // 오른쪽 자식 노드가 존재한다면
        if (H[i * 2 + 1] > H[bigger])       // 오른쪽 자식 노드 값이 크다면 bigger 갱신
            bigger = i * 2 + 1;
    }
    if (H[i] >= H[bigger])      // bigger보다 현재(i) 값이 더 크다면 힙 유지
        return;
    int tmp = H[i];
    H[i] = H[bigger];
    H[bigger] = tmp;
    downHeap(bigger, last);
}
```

downHeap(i,last) 함수

: 힙 내 위치 i에 저장된 키를 크기에 맞는 위치로 하향 이동

힙과 힙정렬(2) 실습문제

```
void insertItem(int key) {
    n++;          // n 갱신
    H[n] = key;   // 힙의 초기 삽입 위치는 n
    upHeap(n);    // 힙 조정
}
```

insertItem() 함수

: n위치에 key 삽입,

upHeap(n) 호출 수행 후 n(총 키 개수) 갱신

```
void upHeap(int i) {
    if (i == 1)
        return;
    if (H[i] <= H[i / 2])
        return;
    int tmp = H[i];
    H[i] = H[i / 2];
    H[i / 2] = tmp;
    upHeap(i / 2);
}
```

upHeap() 함수

: 힙 내 위치 i 에 저장된 키를 크기에 맞는 위치로 상향 이동

힙과 힙정렬(2) 실습문제

입력 예시 1

```
3                    ↦ 키 개수
209 400 77           ↦ 키들
```

출력 예시 1

```
□77 209 400          ↦ 정렬 리스트
```

입력 예시 2

```
6                    ↦ 키 개수
24 17 33 50 60 70    ↦ 키들
```

출력 예시 2

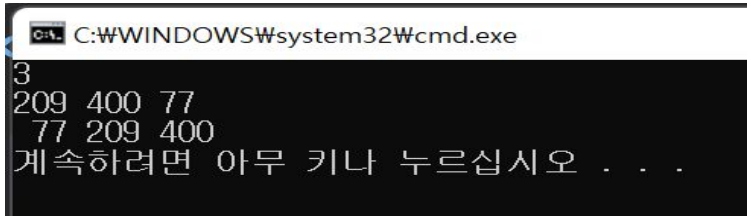
```
□17 24 33 50 60 70   ↦ 정렬 리스트
```

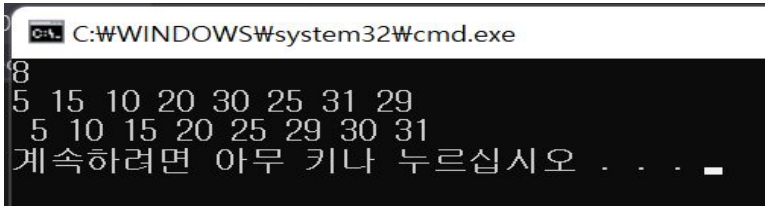
입력 예시 3

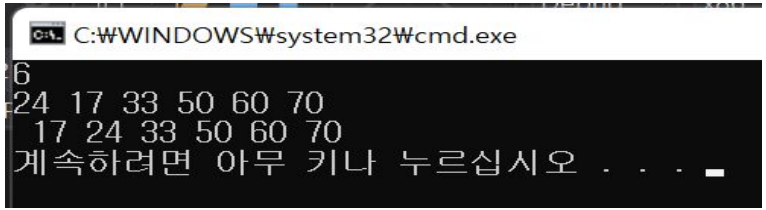
```
8                    ↦ 키 개수
5 15 10 20 30 25 31 29  ↦ 키들
```

출력 예시 3

```
□5 10 15 20 25 29 30 31  ↦ 정렬 리스트
```





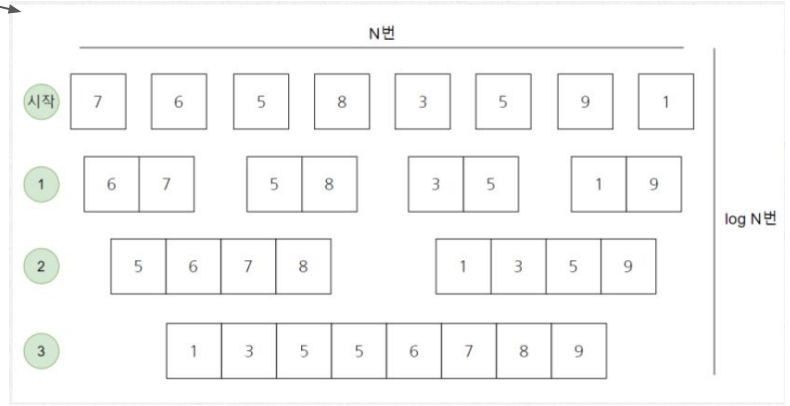


합병정렬 개념 정리

합병 정렬(merge-sort)이란?

- 분할통치법에 기초한 정렬 알고리즘
- ‘일단 반으로 나누고 나중에 합쳐서 정렬하면 어떨까?’라는 아이디어
- 힙정렬처럼, O(n log n) 시간에 수행
- 힙정렬과는 달리, 외부의 우선순위큐 사용 X,

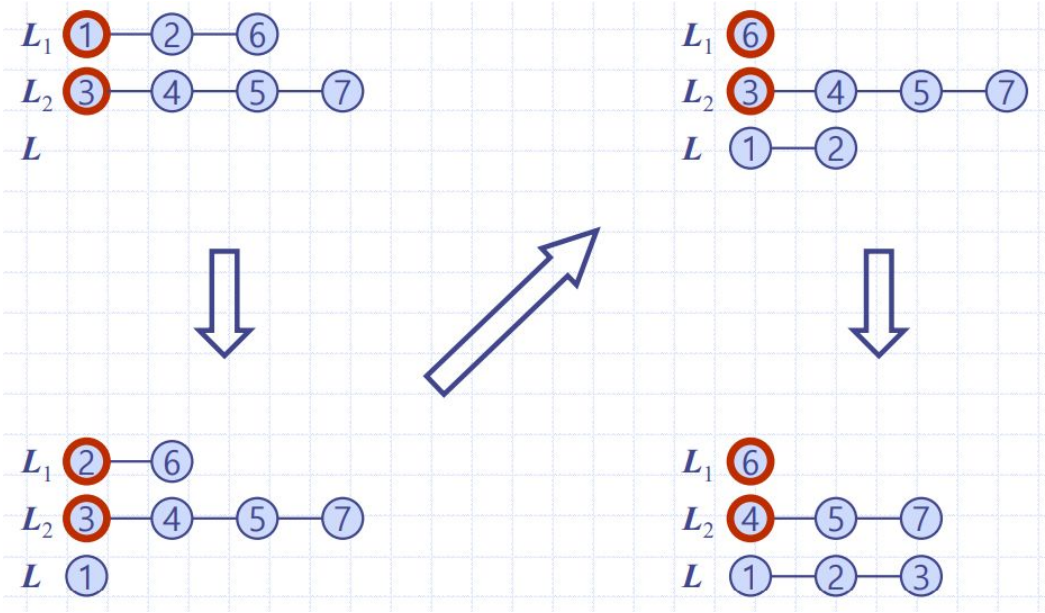
  데이터를 순차적 방식으로 접근



합병정렬 개념 정리

합병 정렬(merge-sort)의 세 단계

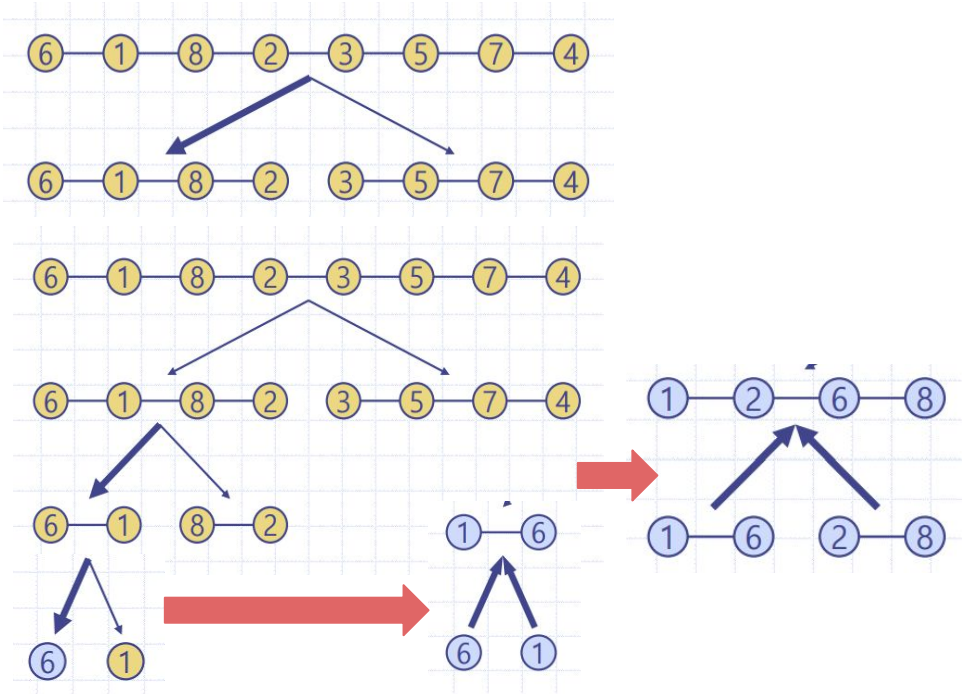
1. 분할(divide)
2. 재귀(recur)
3. 통치(conquer)



합병정렬 개념 정리

합병 정렬(merge-sort)의 세 단계

1. 분할(divide)
2. 재귀(recur)
3. 통치(conquer)



합병정렬 실습문제

[문제 1] (합병 정렬) N개의 양의 정수를 입력(중복 허용)받아 정렬하는 프로그램을 작성하시오. 정렬은 **단일연결리스트**를 이용하여 합병정렬을 구현하여 사용한다.

- 구현해야할 합병 정렬 알고리즘:
  - 크기가 N인 단일연결리스트를 **동적 할당**하여, 입력된 양의 정수 저장 (입력 정수는 중복 허용)
  - mergeSort(L) 함수: 단일연결리스트 L의 원소들을 합병정렬하여 정렬된 결과를 오름차순으로 정렬
  - merge(L1, L2) 함수: mergeSort에 호출되어 두 개의 정렬된 단일연결리스트 L1과 L2를 합병한 하나의 단일연결리스트를 반환. 합병을 위해서 새로운 공간을 할당하면 안되고, L1과 L2 노드들의 링크만 변화시켜서 합병.
  - mg-partition(L, k) 함수: 단일연결리스트 L과 양의 정수 k를 입력받아서 L을 크기가 k이고 |L|-k인 두 개의 부분리스트 L1과 L2로 분할하여 (L1, L2)를 반환. 여기서 |L|은 L의 크기. 분할 시에도 추가로 공간을 할당해서 사용하지 않고, L의 공간을 그대로 사용해서 분할.

init(ListType *L)함수
isEmpty(ListType *L)함수
insertFirst(ListType *L)함수
insertLast(ListType *L)함수
deleteFirst(ListType *L)함수
get(ListType *L)함수
print(ListType *L)함수

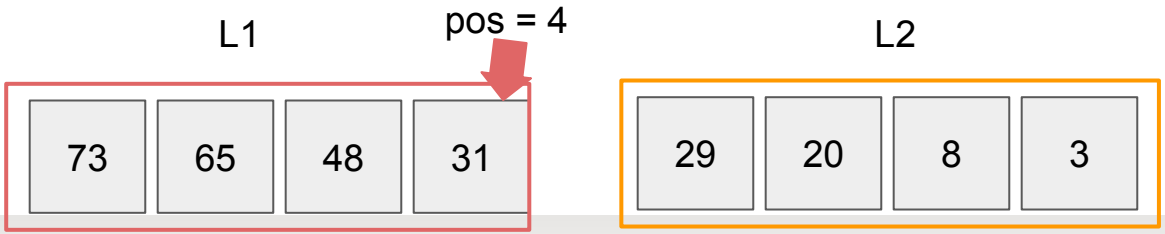
mergeSort(L)함수
merge(L1,L2)함수
mg-partition(L,k)함수

합병정렬 실습문제

L을 부리스트 L1, L2로 분할하는 함수

```
void mg_partition(ListType* L, ListType* L1, ListType* L2, int pos) {      // 리스트 분할
    // position까지가 L1, 그 뒤가 L2
    // L1, L2제외 추가 공간 사용 X, L의 공간 그대로 사용하여 분할
    ListNode* p = L->H;
    for (int i = 1; i <= pos; i++) {
        insert(L1, p->elem);
        p = p->next;
    }
    for (int i = pos + 1; i <= L->size; i++) {
        insert(L2, p->elem);
        p = p->next;
    }
    free(p);
}
```

N=8
pos = 8 / 2 = 4



합병정렬 실습문제

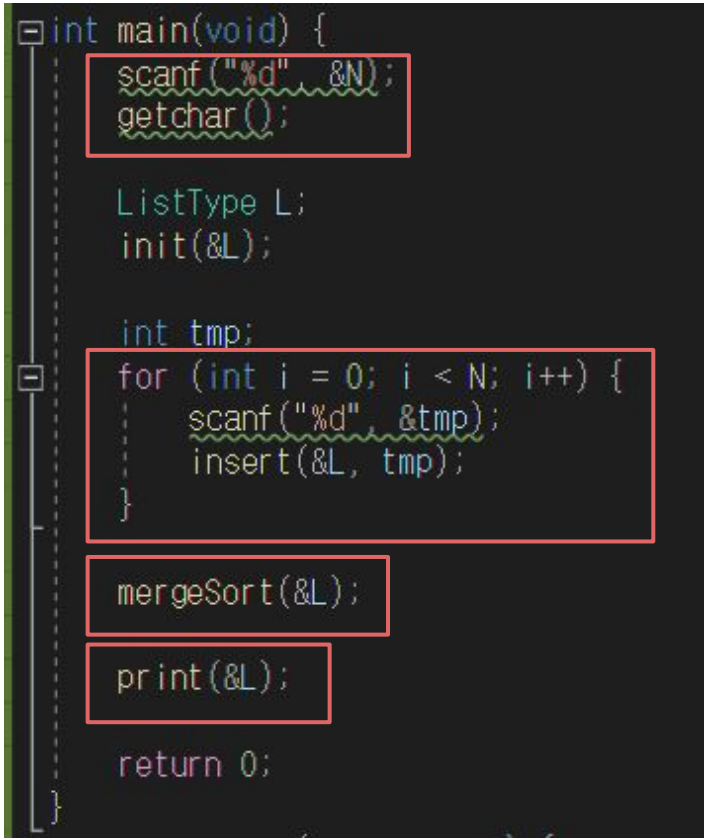
```
void merge(ListType* L, ListType* L1, ListType* L2) {
    init(L);
    while (!isEmpty(L1) && !isEmpty(L2)) {
        if (get(L1) <= get(L2))
            insertLast(L, deleteFirst(L1));
        else
            insertLast(L, deleteFirst(L2));
    }
    while (!isEmpty(L1))
        insertLast(L, deleteFirst(L1));
    while (!isEmpty(L2))
        insertLast(L, deleteFirst(L2));
}
```

- mergeSort 함수에 호출되는 함수
- 두개의 부리스트 L1,L2를 합병한 하나의 리스트를 만드는 함수

```
void mergeSort(ListType* L) {
    if (L->size > 1) {
        ListType L1, L2;
        init(&L1);  init(&L2);
        mg_partition(L, &L1, &L2, L->size / 2);
        mergeSort(&L1);
        mergeSort(&L2);
        merge(L, &L1, &L2);
    }
}
```

- L의 원소들을 합병정렬하여 정렬된 결과를 오름차순으로 정렬시키는 함수

합병정렬 실습문제


int main(void) {
scanf("%d", &N);
getchar();
ListType L;
init(&L);
int tmp;
for (int i = 0; i < N; i++) {
scanf("%d", &tmp);
insert(&L, tmp);
}
mergeSort(&L);
print(&L);
return 0;
}

합병정렬 실습문제

입력 예시 1

```
3          ↦ N
4 9 1
```

출력 예시 1

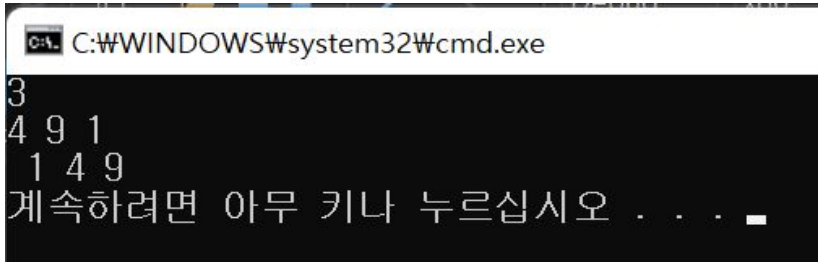
```
□1 4 9          ↦ 정렬 결과
```

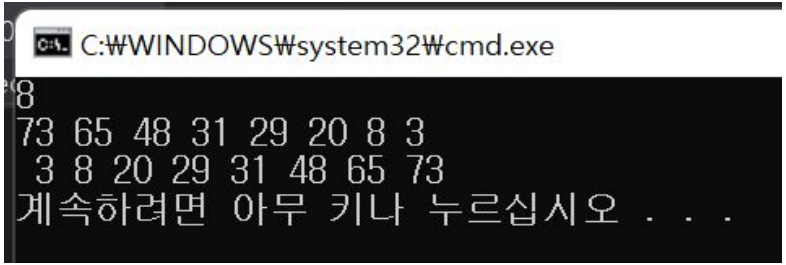
입력 예시 2

```
8          ↦ N
73 65 48 31 29 20 8 3
```

출력 예시 2

```
□3 8 20 29 31 48 65 73    ↦ 정렬 결과
```





퀵정렬 개념 정리

퀵 정렬(quick-sort)란?

- 분할통치법에 기초한 정렬 알고리즘
- 가장 빠른 정렬 알고리즘 중 하나
- 배열을 조금씩 나누어 보다 작은 문제를 해결하는 과정을 반복 → 시간 복잡도 O(n log n)
- 최악의 시간 복잡도: O(n^2)

퀵정렬 개념 정리

기준 원소(pivot) 선택

- **결정적**이면서 쉬운 방법: 맨 앞, 맨 뒤, 중간 원소
- **결정적**이면서 조금 복잡한 방법:

  1. 맨 앞, 중간, 맨 뒤 위치의 세 원소의 중앙값
  2. 0/4,1/4,2/4,3/4,4/4 위치 다섯 원소의 중앙값
  3. 전체 원소의 중앙값

- **무작위**한 방법

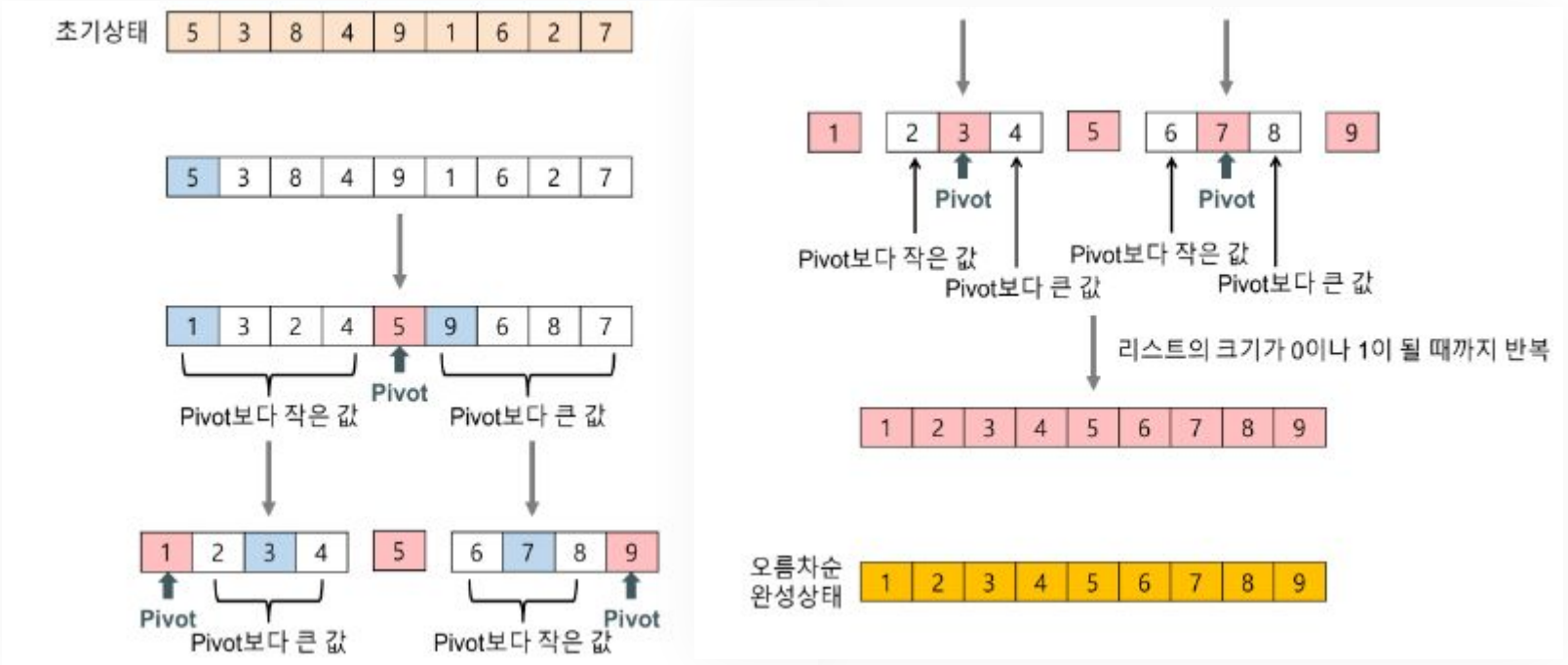
퀵정렬 개념 정리

퀵 정렬의 단계

- **분할** : 입력 배열을 피벗을 기준으로 비균등하게 2개의 부분 배열로 분할
- **정복**: 부분 배열을 정렬. 부분 배열의 크기가 충분히 작지 않으면 순환 호출을 이용해 다시 분할 정복 방법을 적용
- **결합**: 정렬된 부분 배열들을 하나의 배열에 합병

퀵정렬 개념 정리

퀵 정렬 수행



퀵정렬 실습문제

[문제 2] (퀵 정렬) N개의 양의 정수를 입력(중복 허용)받아 정렬하는 프로그램을 작성하시오. 정렬은 **아래에 명시된 퀵 정렬을** 구현하여 사용한다.

- 구현해야할 퀵 정렬 알고리즘:
  - 크기가 N인 배열을 **동적 할당**하여, 입력된 양의 정수 저장 (입력 정수는 중복 허용)
  - 기준값(pivot)을 정할 때, 다음의 방법을 이용한다:
    (1) 입력된 수들 중에서 3개의 수를 랜덤하게 선택한다. (즉, 입력배열의 l번째 수부터 r번째 수 중에서 3개의 수를 랜덤하게 선택)
    (2) 랜덤하게 선택된 3개의 수 중에서 중간값(median)을 구하여 이를 pivot으로 한다.
    (3) pivot을 정하는 부분을 partition함수 내에서 처리해도 된다. 혹은, 힌트에 주어진 알고리즘에서 처럼 pivot을 정하고 그 인덱스를 반환하는 함수 find_pivot_index를 따로 작성해서 partiton에 pivot index를 인자로 넘겨줘도 된다.
  - partition 함수의 반환 값은 두 인덱스인 (a,b)로 partition의 결과로, 배열의 l번째 수부터 a-1번째 수는 pivot보다 작은 값을 갖고, 배열의 a번째부터 b번째 수는 pivot과 같은 값을 갖고, b+1번째부터 r번째 수는 pivot보다 큰 값을 갖게 된다. (즉, 이후 호출되는 재귀함수는 l부터 a-1까지 배열에 대해서와 b+1부터 r까지의 배열에 대해서 다루고, pivot과 같은 값들인 a부터 b번째 값들은 재귀에서 제외된다.)

퀵정렬 실습문제

```
int inPlaceQuickSort(int L[], int l, int r) {
    int a, b;
    if (l >= r)
        return 0;
    int k = find_pivot_index(L, l, r);
    a = b = inPlacePartition(L, l, r, k);
    inPlaceQuickSort(L, l, a - 1);
    inPlaceQuickSort(L, b + 1, r);

}
void print(int L[], int N) {
    for (int i = 0; i < N; i++)
        printf(" %d", L[i]);
}
int main() {
    int N;
    int* L;
    scanf("%d", &N);
    L = (int*)malloc(N * sizeof(int));
    for (int i = 0; i < N; i++) {
        scanf("%d", &L[i]);
    }
    inPlaceQuickSort(L, 0, N - 1);
    print(L, N);
    return 0;
}
```

(2) 기준값의 인덱스를 찾는 함수 호출

분할 함수를 통해 pivot의 값 찾음

pivot보다 작은 값과 큰 값들의 배열을 재귀적 호출하여 위와 같이 진행

(1) N개의 양의 정수를 입력 받은 후에 크기가 N인 배열을 동적 할당 받음
이후에 inPlaceQuickSort 함수를 호출

퀵정렬 실습문제

```
find_pivot_index(int L[], int l, int r) {
if (r - l <= 1)
    return l;
srand((unsigned)time(NULL));
int rValue1 = (rand() % (r - l)) + l;
int rValue2 = (rand() % (r - l)) + l;
int rValue3 = (rand() % (r - l)) + l;
if ((L[rValue1] >= L[rValue2] && L[rValue1] <= L[rValue3]) || (L[rValue1] <= L[rValue2] && L[rValue1] >= L[rValue3]))
    return rValue1;
else if ((L[rValue2] >= L[rValue1] && L[rValue2] <= L[rValue3]) || (L[rValue2] <= L[rValue1] && L[rValue2] >= L[rValue3]))
    return rValue2;
else
    return rValue3;
```

(3) 입력 배열의 l 번째 수부터 r번째 수 중에서 3개의 수를 랜덤하게 선택
랜덤하게 선택된 3개의 수 중에서 중간값을 구하여 이를 pivot으로 하고 그 인덱스 반환

퀵정렬 실습문제

```
inPlacePartition(int L[], int l, int r, int k) {
int p = L[k];
int tmp;
tmp = L[k];
L[k] = L[r];
L[r] = tmp;
int i = l;
int j = r - 1;
while (i <= j) {
    while (i <= j && L[i] < p)
        i++;
    while (j >= i && L[j] >= p)
        j--;
    if (i < j) {
        tmp = L[i];
        L[i] = L[j];
        L[j] = tmp;
    }
}
int a = i;
j = r - 1;
while (i <= j) {
    while (i <= j && L[i] == p)
        i++;
    while (j >= i && L[j] > p)
        j--;
    if (i < j) {
        tmp = L[i];
        L[i] = L[j];
        L[j] = tmp;
    }
}
tmp = L[i];
L[i] = L[r];
L[r] = tmp;
return a;
```

퀵정렬 실습문제

입력 예시 1	출력 예시 1
3 ↦ N 4 9 1	□1 4 9 ↦ 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