14 주차 실습

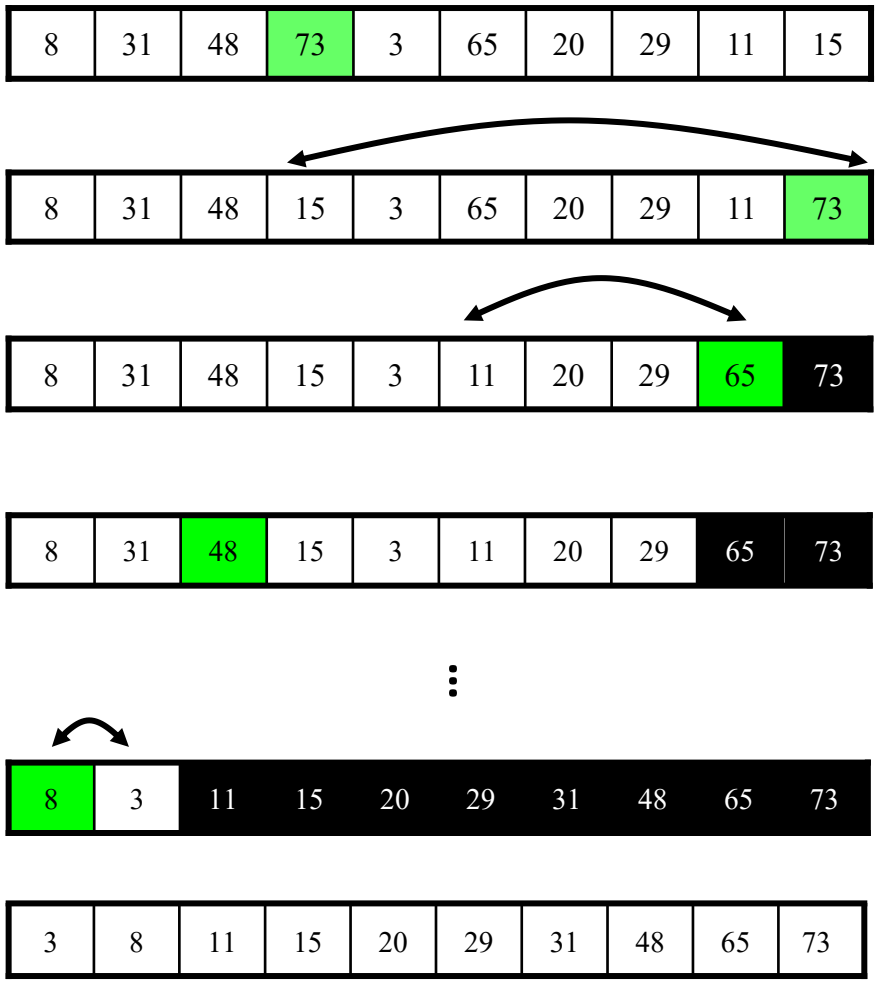
정렬 통합 실습

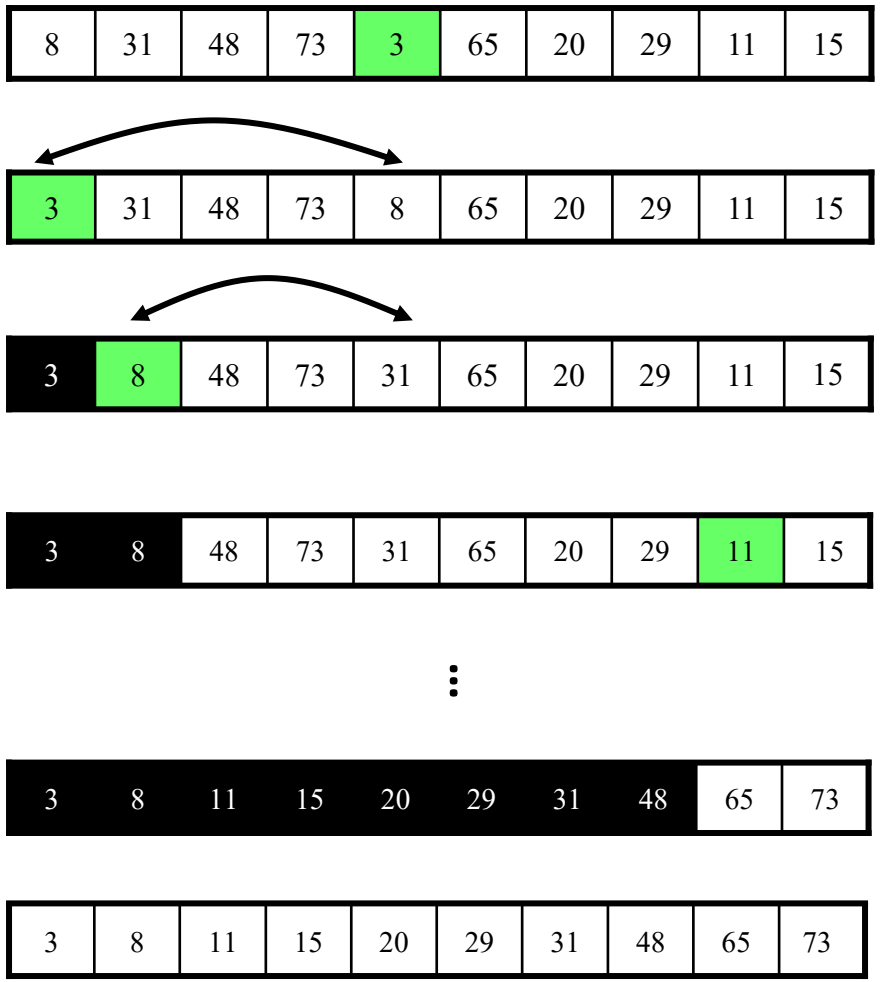
선택, 삽입, 버블, 쉘, 합병, 퀵

- 간편 테스트를 위해 전역 변수로 각 변수 선언

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define SWAP(x, y, t) ( (t)=(x), (x)=(y), (y)=(t) )
#define MAX_SIZE 100000L
int sel[MAX_SIZE];
int in[MAX_SIZE];
int bubble[MAX_SIZE];
int shell[MAX_SIZE];

int merge[MAX_SIZE];
int sorted[MAX_SIZE];

int qu[MAX_SIZE];
```

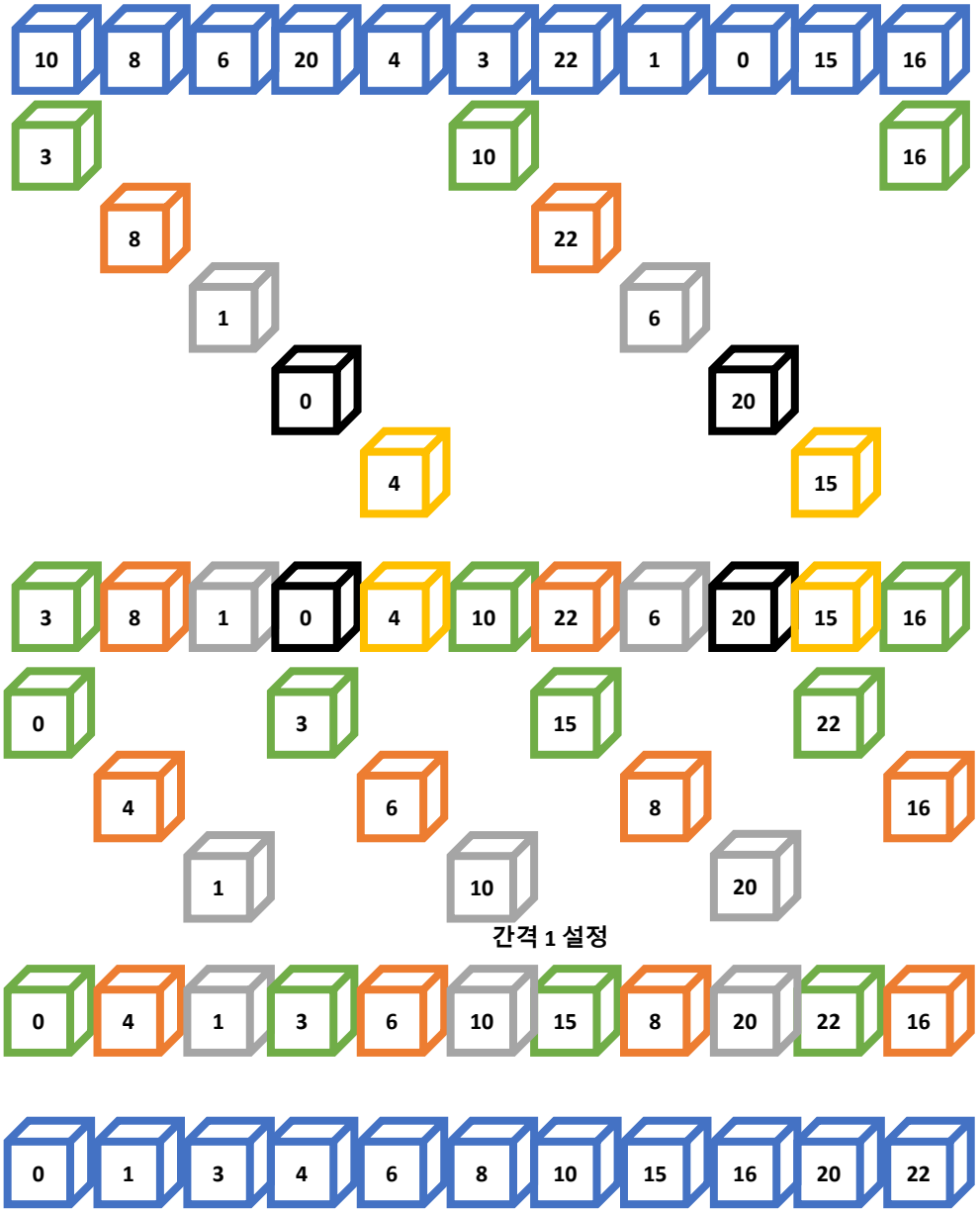
정렬 통합 실습(선택, 삽입, 버블, 쉘, 합병, 퀵)





정렬 통합 실습(선택, 삽입, 버블, 쉘, 합병, 퀵)

정렬 통합 실습(선택, 삽입, 버블, 쉘, 합병, 퀵)



정렬 통합 실습

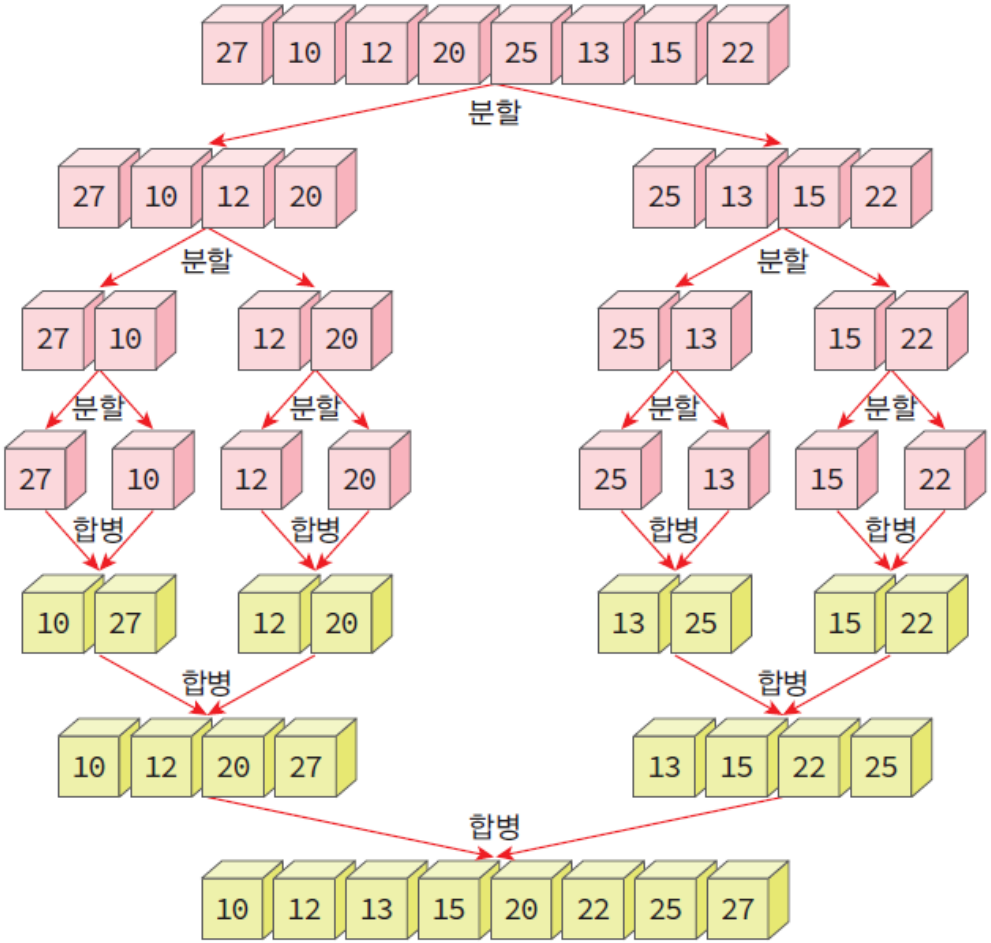
선택, 삽입, 버블, 쉘, 합병, 퀵

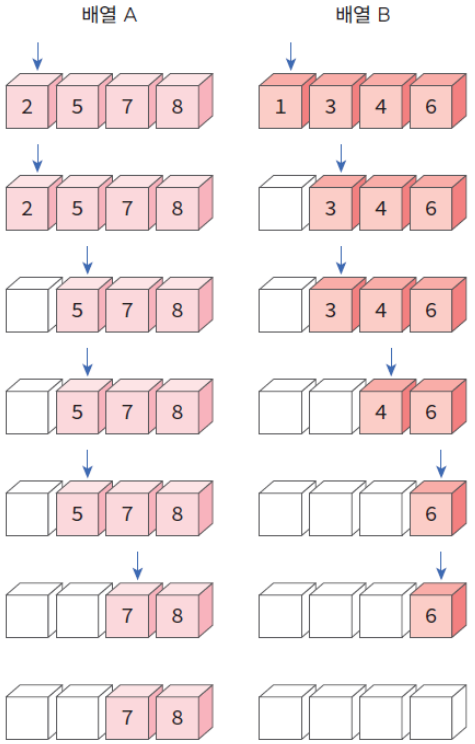
- 선택, 삽입, 버블, 쉘 소스

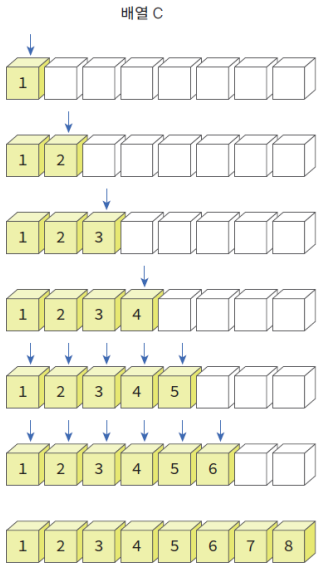
```
void selection_sort(int list[], int n){
   int i, j, least, temp;
   for (i = 0; i < n - 1; i++) {
      least = i;
      for (j = i + 1; j < n; j++)
         if (list[j] < list[least]) least = j;
      SWAP(list[i], list[least], temp);
   }
}
void insertion_sort(int list[], int n){
   int i, j, key;
   for (i = 1; i < n; i++) {
      key = list[i];
      for (j = i - 1; j >= 0 && list[j] > key; j--)
         list[j + 1] = list[j];
      list[j + 1] = key;
   }
}
void bubble_sort(int list[], int n) {
   int i, j, temp;
   for (i = n - 1; i > 0; i--) {
      for (j = 0; j < i; j++)
         if (list[j] > list[j + 1])
            SWAP(list[j], list[j + 1], temp);
   }
}
```

```
void inc_insertion_sort(int list[], int first, int last, int gap){
   int i, j, key;
   for (i = first + gap; i <= last; i = i + gap) {
      key = list[i];
      for (j = i - gap; j >= first && key < list[j]; j = j - gap)
         list[j + gap] = list[j];
      list[j + gap] = key;
   }
}

void shell_sort(int list[], int n){
   int i, gap;
   for (gap = n / 2; gap > 0; gap = gap / 2) {
      if ((gap % 2) == 0) gap++;
      for (i = 0; i < gap; i++)
         inc_insertion_sort(list, i, n - 1, gap);
   }
}
```

정렬 통합 실습(선택, 삽입, 버블, 쉘, 합병, 퀵)







정렬 통합 실습

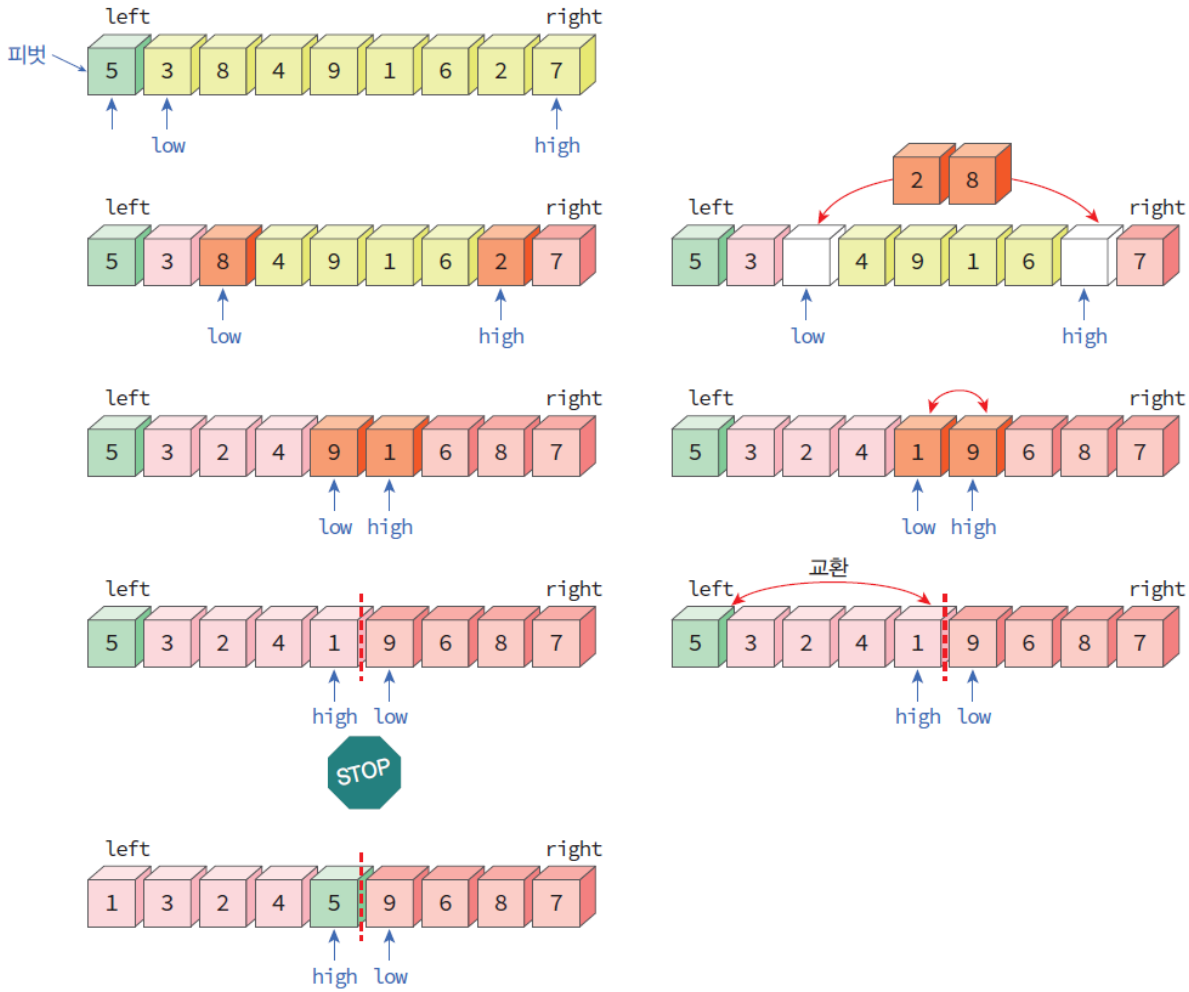
선택, 삽입, 버블, 쉘, 합병, 퀵

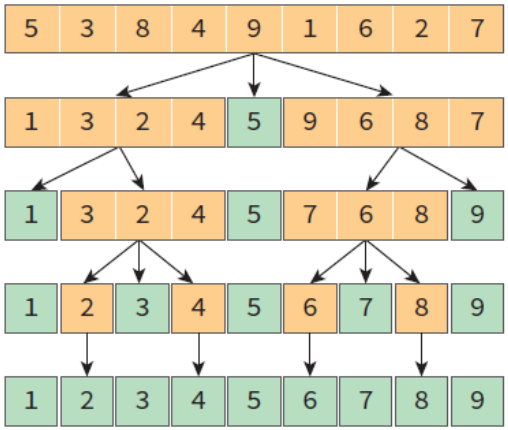
- 합병 소스

```
void merge_s(int list[], int left, int mid, int right) {
  int i, j, k, l;
  i = left;  j = mid + 1;  k = left;
  while (i <= mid && j <= right) {
    if (list[i] <= list[j])
      sorted[k++] = list[i++];
    else
      sorted[k++] = list[j++];
  }
  if (i > mid)
    for (l = j; l <= right; l++)
      sorted[k++] = list[l];
  else
    for (l = i; l <= mid; l++)
      sorted[k++] = list[l];
  for (l = left; l <= right; l++)
    list[l] = sorted[l];
}
```

```
void merge_sort(int list[], int left, int right)
{
  int mid;
  if (left < right) {
    mid = (left + right) / 2;
    merge_sort(list, left, mid);
    merge_sort(list, mid + 1, right);
    merge_s(list, left, mid, right);
  }
}
```

정렬 통합 실습(선택, 삽입, 버블, 쉘, 합병, 퀵)





정렬 통합 실습

선택, 삽입, 버블, 쉘, 합병, 퀵

- 퀵 소스

```
int partition(int list[], int left, int right){
  int pivot, temp;
  int low, high;
  low = left;
  high = right + 1;
  pivot = list[left];
  do {
    do
      low++;
    while (list[low] < pivot);
    do
      high--;
    while (list[high] > pivot);
    if (low < high) SWAP(list[low], list[high], temp);
  } while (low < high);
  SWAP(list[left], list[high], temp);
  return high;
}
```

```
void quick_sort(int list[], int left, int right){
  if (left < right) {
    int q = partition(list, left, right);
    quick_sort(list, left, q - 1);
    quick_sort(list, q + 1, right);
  }
}
```

정렬 통합 실습

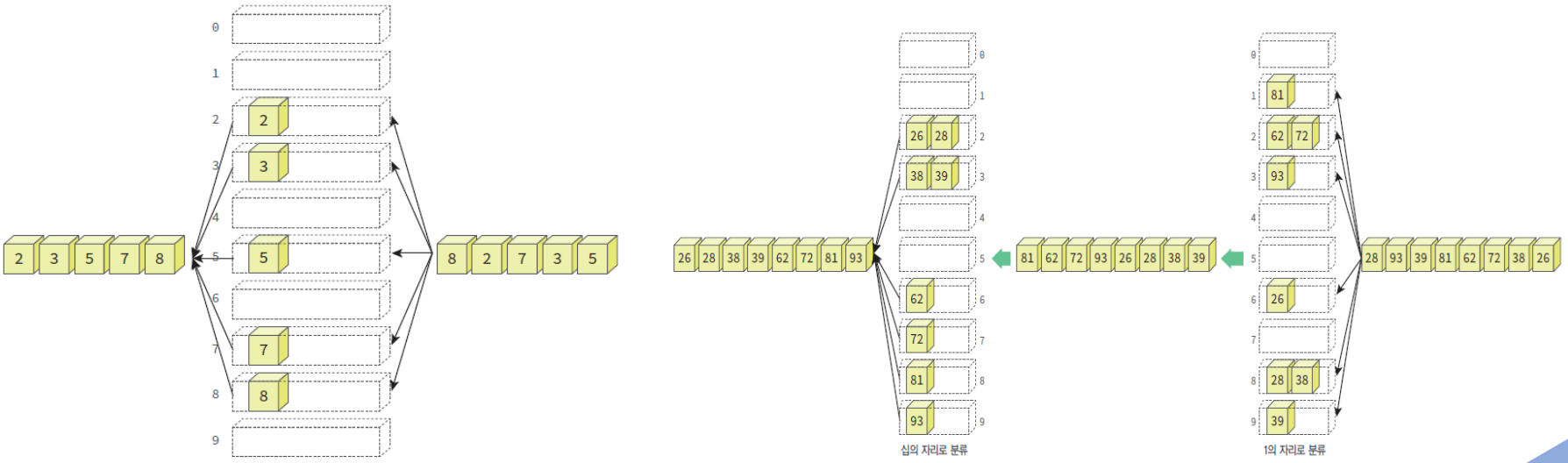
선택, 삽입, 버블, 쉘, 합병, 퀵

```
int main(void){
  int i;
  int n = MAX_SIZE;
  double start, end;

  srand(time(NULL));
  for (i = 0; i < n; i++)      // 난수 생성 및 출력
    sel[i] = rand() % 100; // 난수 발생 범위 0~99

  memmove(in, sel, sizeof(int) * MAX_SIZE);
  memmove(bubble, sel, sizeof(int) * MAX_SIZE);
  memmove(shell, sel, sizeof(int) * MAX_SIZE);
  memmove(qu, sel, sizeof(int) * MAX_SIZE);

  start = (double)clock() / CLOCKS_PER_SEC;
  selection_sort(sel, n); // 선택정렬 호출
  end = (((double)clock()) / CLOCKS_PER_SEC);
  printf("선택 정렬 수행시간 %lfWn", end - start);

  start = (double)clock() / CLOCKS_PER_SEC;
  insertion_sort(in, n); // 삽입정렬 호출
  end = (((double)clock()) / CLOCKS_PER_SEC);
  printf("삽입 정렬 수행시간 %lfWn", end - start);

  start = (double)clock() / CLOCKS_PER_SEC;
  bubble_sort(bubble, n); // 버블정렬 호출
  end = (((double)clock()) / CLOCKS_PER_SEC);
  printf("버블 정렬 수행시간 %lfWn", end - start);

  start = (double)clock() / CLOCKS_PER_SEC;
  shell_sort(shell, n); // 쉘정렬 호출
  end = (((double)clock()) / CLOCKS_PER_SEC);
  printf("쉘 정렬 수행시간 %lfWn", end - start);

  start = (double)clock() / CLOCKS_PER_SEC;
  merge_sort(merge, 0, n - 1); // 합병정렬 호출
  end = (((double)clock()) / CLOCKS_PER_SEC);
  printf("합병 정렬 수행시간 %lfWn", end - start);

  start = (double)clock() / CLOCKS_PER_SEC;
  quick_sort(qu, 0, n - 1); // 퀵정렬 호출
  end = (((double)clock()) / CLOCKS_PER_SEC);
  printf("퀵 정렬 수행시간 %lfWn", end - start);

  //for (i = 0; i < n; i++)
    //printf("%d ", list[i]);
  //printf("Wn");
  return 0;
}
```

정렬

기수 정렬(Radix sort)

- 대부분의 정렬 방법들은 레코드들을 비교함으로써 정렬 수행
- 기수 정렬(radix sort)은 레코드를 비교하지 않고 정렬 수행
  - 비교에 의한 정렬의 하한인 O(n*log(n)) 보다 좋을 수 있음
  - 기수 정렬은 O(dn) 의 시간적 복잡도를 가짐(대부분 d<10 이하)



정렬

기수 정렬(Radix sort)

```
RadixSort(list, n):
    for d←LSD의 위치 to MSD의 위치 do {
      d번째 자릿수에 따라 0번부터 9번 버켓에 넣는다.
      버켓에서 숫자들을 순차적으로 읽어서 하나의 리스트로 합친다.
      d++;
    }
```

- 버켓 = 큐로 구현
- 버켓 개수는 키의 표현 방법과 밀접한 관계가 될 수 있음
  - 이진법을 사용한다면 버켓 = 2개
  - 알파벳 문자를 사용한다면 버켓은 26개
  - 십진법을 사용한다면 버켓은 10개

실습

구현 방법 (기수 정렬)

```c
#include <stdio.h>
#include <stdlib.h>

#define MAX_QUEUE_SIZE 100
typedef int element;
typedef struct { // 큐 타입
  element  data[MAX_QUEUE_SIZE];
  int  front, rear;
} QueueType;

void error(char* message) {
  fprintf(stderr, "%s₩n", message);
  exit(1);
}

void init_queue(QueueType* q) {
  q->front = q->rear = 0;
}

int is_empty(QueueType* q) {
  return (q->front == q->rear);
}
```

```c
int is_full(QueueType* q) {
  return ((q->rear + 1) % MAX_QUEUE_SIZE == q->front);
}

void enqueue(QueueType* q, element item) {
  if (is_full(q))
    error("큐가 포화상태입니다");
  q->rear = (q->rear + 1) % MAX_QUEUE_SIZE;
  q->data[q->rear] = item;
}

element dequeue(QueueType* q) {
  if (is_empty(q))
    error("큐가 공백상태입니다");
  q->front = (q->front + 1) % MAX_QUEUE_SIZE;
  return q->data[q->front];
}
```

실습

구현 방법 (기수 정렬)

```
#define BUCKETS 10
#define DIGITS  4
void radix_sort(int list[], int n) {
  int i, b, d, factor = 1;
  QueueType queues[BUCKETS];
  for (b = 0; b < BUCKETS; b++)
    init_queue(&queues[b]);
  for (d = 0; d < DIGITS; d++) {
    for (i = 0; i < n; i++)
      enqueue(&queues[(list[i] / factor) % 10],
list[i]);

  for (b = i = 0; b < BUCKETS; b++)
    while (!is_empty(&queues[b]))
      list[i++] = dequeue(&queues[b]);
    factor *= 10;
  }
}
```

```
#define  SIZE 10

int main(void) {
  int list[SIZE];
  srand(time(NULL));
  for (int i = 0; i < SIZE; i++)
    list[i] = rand() % 100;

  radix_sort(list, SIZE);
  for (int i = 0; i < SIZE; i++)
    printf("%d ", list[i]);
  printf("₩n");
  return 0;
}
```