동국대학교
dongguk university

실습: Week 13

Data Structures

Contents

- 프로젝트 관련 안내
- Sorting
  - Sorting algorithms
- 실습
  - 실습 13. Sorting - insertion sort, quick sort, merge sort

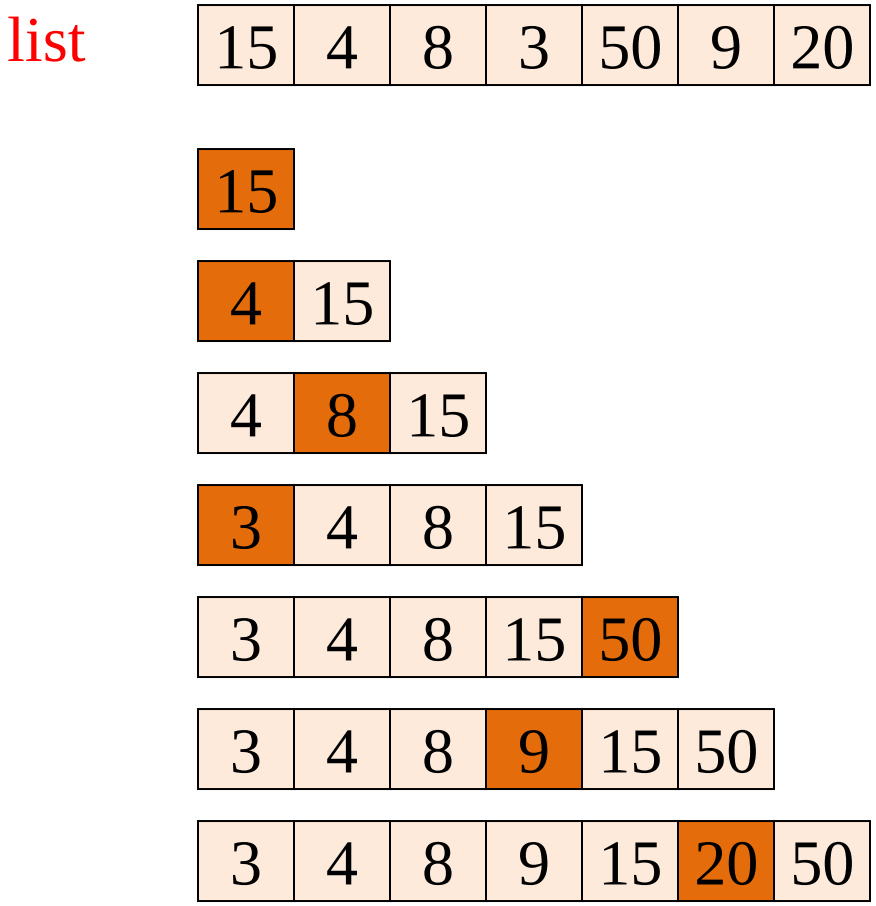
프로젝트

- 제출 기한 :
  - 6/8 09:59 (수업시간 전까지)
  - 결과보고서, 소스코드, 실행파일
- 데모 방식 :
  - 14주 실습시간에 진행
    - 1부(10~11시) , 2부(11~12시) 로 나눠서 진행
  - 한 팀 씩 실시간으로 프로그램 실행 (5분 소요 예상)
    - 이클래스에서 다운로드한 소스코드로 진행
    - 간단한 코드 설명 (자료 구조, 색인, 탐색 알고리즘 , 소요시간 등)

Sorting

- 정렬
  - 자료를 특정 키 값에 따라 오름차순(내림차순)으로 정리
- 정렬방법
  - 내부정렬(internal sorting)
    - 메인 메모리 내에서 정렬
    - Bubble sort, Insertion sort, Quick sort, Shell sort, Heap sort, Radix sort, ...
  - 외부정렬(external sorting)
    - 자료의 양이 방대하여 보조기억 장치를 활용하여 정렬

Insertion Sort

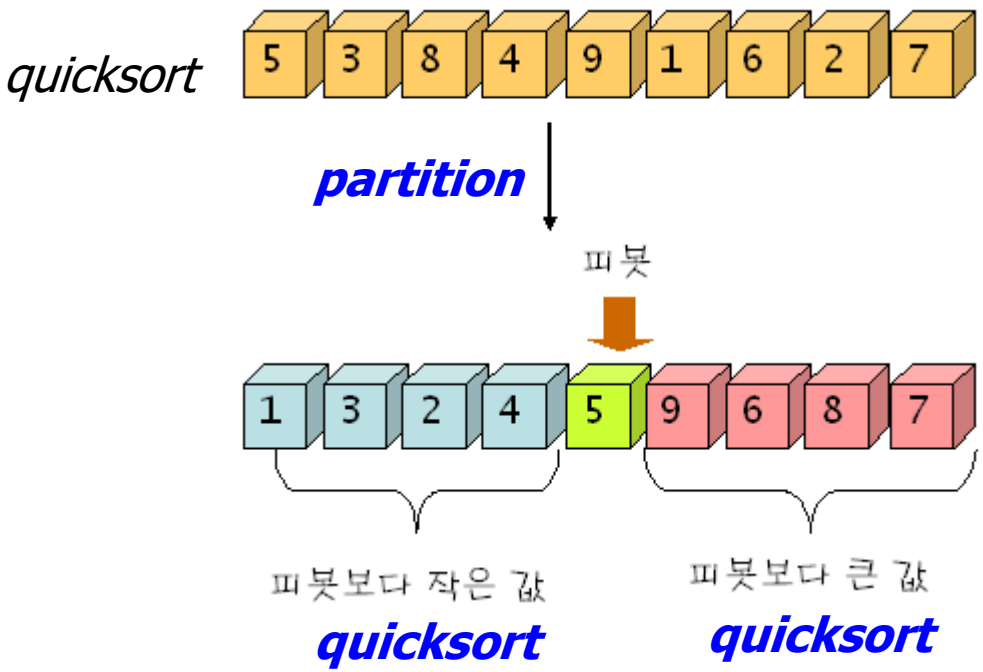


Insertion Sort

- Time complexity
  - worst case: $1 + 2 + \ldots + (n-1) = O(n^2)$
  - Best case : $O(n)$
- 장점
  - 알고리즘이 간단함
  - 안정성이 있음
  - 거의 정렬된 리스트에서는 매우 효율적임

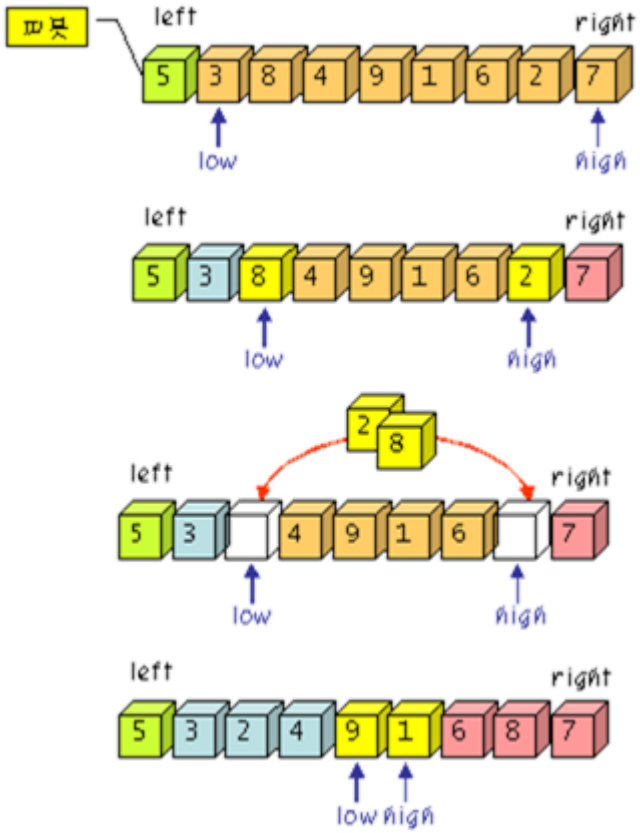
Quick Sort

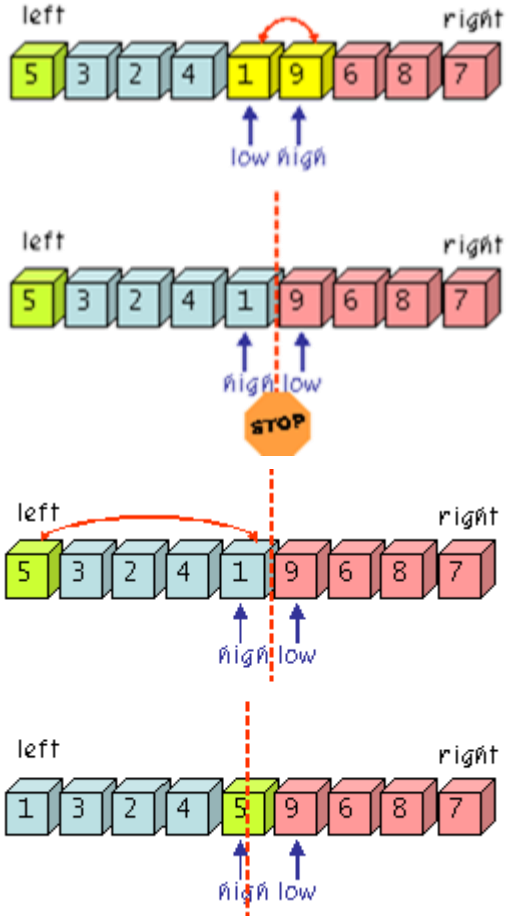
- Divide-and-conquer (분할정복)
  - 피봇 키를 사용하여 리스트를 분할
  - Recursive하게 두번의 quick sort 호출



Quick Sort

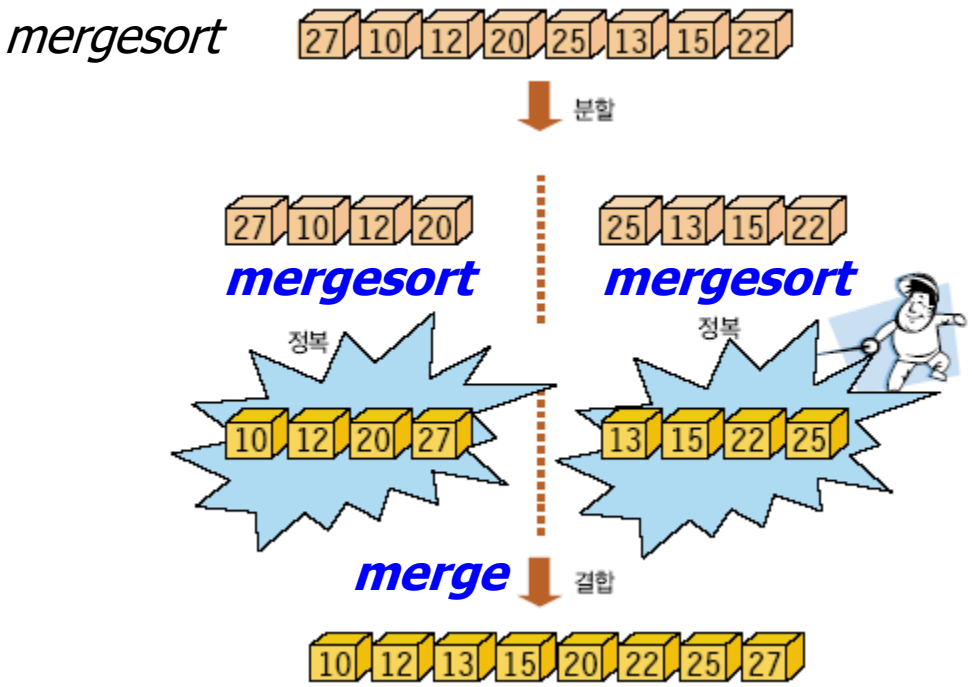
- 분할 (partition)
  - O(n)





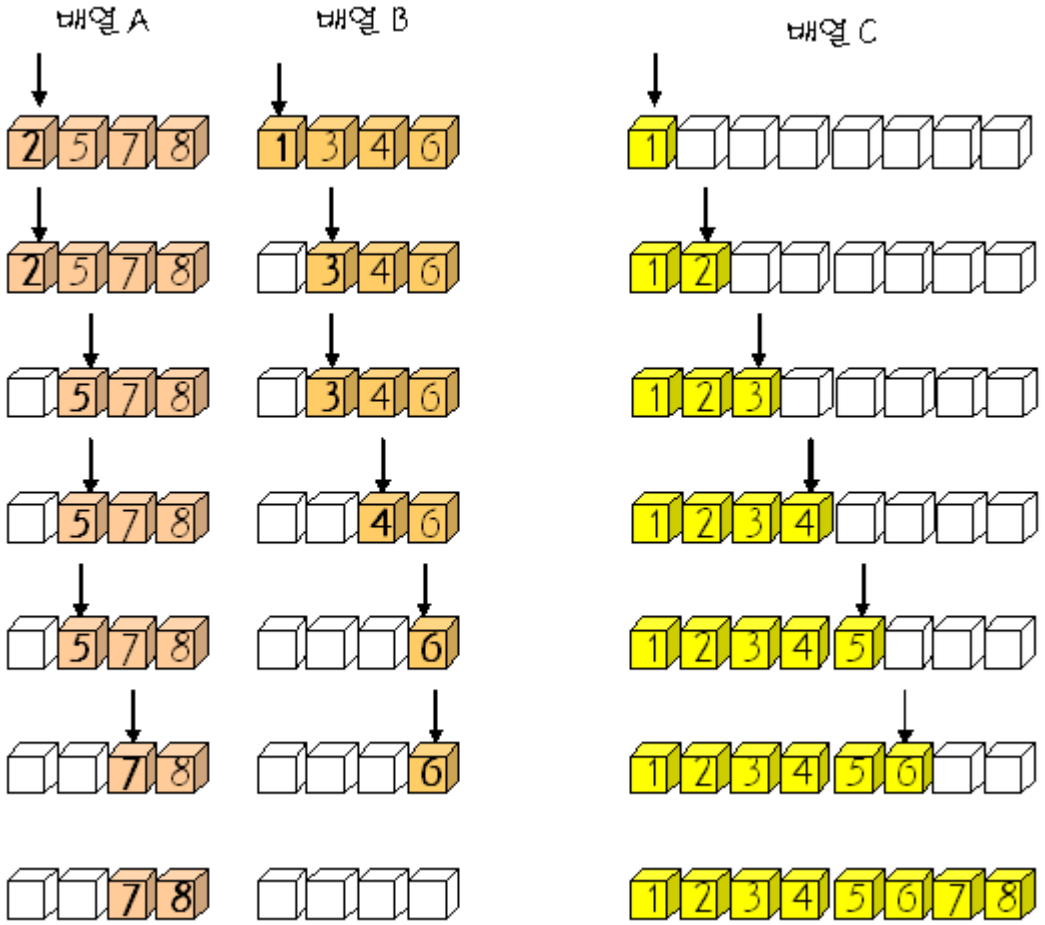
Merge Sort

- Divide-and-conquer (분할정복)
  - 리스트를 분할하여 두번의 merge sort 호출
  - 정렬된 리스트 두개를 합병(merge)



Merge Sort

- 합병 (merge)
  - O(n)



실습 13. Sorting

- Sorting 함수의 구현
  - Insertion Sort
  - Quick Sort
  - Merge Sort
  - 주어진 original[]에 대해 각 sorting의 중간 단계와 키 비교 횟수를 출력
- Random data로 test
  - 크기 1000~10000인 random integer array experiment[]를 생성하여 각 sorting의 키 비교 횟수를 출력
  - 중간 단계 출력 제외

자료구조 및 함수

- void insertion_sort(int list[], int n, int show);
  - insertion sort. 키 비교 횟수 기록. show=1 이면 중간 단계 출력
- void quick_sort(int list[], int left, int right, int show);
  - quick sort. 키 비교 횟수 기록. show=1 이면 중간 단계 출력
- int partitioin(int list[], int left, int right);
  - list[]를 pivot 보다 작은 키들과 큰 키들로 분할. pivot의 index를 반환
- void merge_sort(int list[], int left, int right, int show);
  - merge sort. 키 비교 횟수 기록. show=1 이면 중간 단계 출력
- void merge(int list[], int left, int mid, int right);
  - 정렬되어 있는 list[left..mid]와 list[mid+1..right]를 합병

자료구조 및 함수

- void copy_list(int original[], int list[], int n);
  - original을 list에 복사
- void print_list(int list[], int left, int right);
  - list를 left 에서 right 까지 출력
- void random_initialize(int list[], int n);
  - list[0]~list[n-1]을 random integer로 초기화

실습 13. 실행 예

```
----- insertion sort -----
 46   9  35  78  24  65  53  81  22  18
  9  46  35  78  24  65  53  81  22  18
  9  35  46  78  24  65  53  81  22  18
  9  35  46  78  24  65  53  81  22  18
  9  24  35  46  78  65  53  81  22  18
  9  24  35  46  65  78  53  81  22  18
  9  24  35  46  53  65  78  81  22  18
  9  24  35  46  53  65  78  81  22  18
  9  22  24  35  46  53  65  78  81  18
  9  18  22  24  35  46  53  65  78  81

  9  18  22  24  35  46  53  65  78  81

Total number of comparison = 23
```

```
----- quick sort -----
 46   9  35  78  24  65  53  81  22  18
 22   9  35  18  24  46  53  81  65  78
 18   9  22  35  24
  9  18
             24  35
                         53  81  65  78
                             78  65  81
                             65  78

  9  18  22  24  35  46  53  65  78  81

Total number of comparison = 17
```

```
----- merge sort -----
 46   9  35  78  24  65  53  81  22  18
  9  46
  9  35  46
             24  78
  9  24  35  46  78
                     53  65
                     53  65  81
                                 18  22
                     18  22  53  65  81
  9  18  22  24  35  46  53  65  78  81

  9  18  22  24  35  46  53  65  78  81

Total number of comparison = 23
```

실습 13. 실행 예

```
----- n = 1000 -----
No. of comparison (insertion sort) = 248145
No. of comparison (quick sort) = 17358
No. of comparison (merge sort) = 8715
----- n = 2000 -----
No. of comparison (insertion sort) = 990237
No. of comparison (quick sort) = 26261
No. of comparison (merge sort) = 19427
----- n = 3000 -----
No. of comparison (insertion sort) = 2245914
No. of comparison (quick sort) = 24252
No. of comparison (merge sort) = 30885
----- n = 4000 -----
No. of comparison (insertion sort) = 4024883
No. of comparison (quick sort) = 40745
No. of comparison (merge sort) = 42825
----- n = 5000 -----
No. of comparison (insertion sort) = 6235976
No. of comparison (quick sort) = 47379
No. of comparison (merge sort) = 55208
```

```
----- n = 6000 -----
No. of comparison (insertion sort) = 8981236
No. of comparison (quick sort) = 72418
No. of comparison (merge sort) = 67791
----- n = 7000 -----
No. of comparison (insertion sort) = 12192384
No. of comparison (quick sort) = 76736
No. of comparison (merge sort) = 80665
----- n = 8000 -----
No. of comparison (insertion sort) = 16089588
No. of comparison (quick sort) = 87279
No. of comparison (merge sort) = 93708
----- n = 9000 -----
No. of comparison (insertion sort) = 20447953
No. of comparison (quick sort) = 107763
No. of comparison (merge sort) = 106964
----- n = 10000 -----
No. of comparison (insertion sort) = 25279388
No. of comparison (quick sort) = 101952
No. of comparison (merge sort) = 120477
```

sorting.h

```
#include <stdio.h>

#define SIZE 10
#define EXP_SIZE 10000

#define boolean  int
#define true     1
#define false    0

// 정렬할 테스트 데이터
int      original[] = {46, 9, 35, 78, 24, 65, 53, 81, 22, 18};
int      experiment[];

// 키값 비교 횟수 카운트를 위한 변수
int      num_compare;
```

sorting.h

```
// insertion sort
void insertion_sort(int list[], int n, int show);

// quick sort
void quick_sort(int list[], int left, int right, int show);
int partition(int list[], int left, int right);

// merge sort
void merge_sort(int list[], int left, int right, int show);
void merge(int list[], int left, int mid, int right);

void copy_list(int original[], int list[], int n);
void print_list(int list[], int left, int right);

void random_initialize(int list[], int n);
```

sorting.c – main()

```
#include "sorting.h"

void main()
{
        int     list[SIZE], n = SIZE;
        int     exp_list[EXP_SIZE];

        printf("\n ----- insertion sort ----- \n");
        copy_list(original, list, n);
        print_list(list, 0, n - 1);
        num_compare = 0;
        insertion_sort(list, n, 1);
        printf("\n");
        print_list(list, 0, n - 1);
        printf("\n Total number of comparison = %d \n", num_compare);
```

sorting.c – main()

```
printf("\n ----- quick sort ----- \n");
copy_list(original, list, n);
print_list(list, 0, n - 1);
num_compare = 0;
quick_sort(list, 0, n - 1, 1);
printf("\n");
print_list(list, 0, n - 1);
printf("\n Total number of comparison = %d \n", num_compare);

printf("\n ----- merge sort ----- \n");
copy_list(original, list, n);
print_list(list, 0, n - 1);
num_compare = 0;
merge_sort(list, 0, n - 1, 1);
printf("\n");
print_list(list, 0, n - 1);
printf("\n Total number of comparison = %d \n", num_compare);
```

sorting.c – main()

```
        for (int i = 1; i <= 10; i++) {
                n = i * 1000;
                printf("\n ----- n = %d ----- \n", n);
                random_initialize(experiment, n);

                copy_list(experiment, exp_list, n);
                num_compare = 0; insertion_sort(exp_list, n, 0);
                printf("\n No. of comparison (insertion sort) = %d \n", num_compare);

                copy_list(experiment, exp_list, n);
                num_compare = 0; quick_sort(exp_list, 0, n - 1, 0);
                printf("\n No. of comparison (quick sort) = %d \n", num_compare);

                copy_list(experiment, exp_list, n);
                num_compare = 0; merge_sort(exp_list, 0, n - 1, 0);
                printf("\n No. of comparison (merge sort) = %d \n", num_compare);
        }
}
```

copy_list, print_list 함수

```
void copy_list(int original[], int list[], int n)
{
        for (int i = 0; i < n; i++)
                list[i] = original[i];
}

void print_list(int list[], int left, int right)
{
        for (int i = 0; i < left; i++)
                printf("    ");
        for (int i = left; i <= right; i++)
                printf("%4d", list[i]);
        printf("\n");
}
```

random_initialize 함수

```
void random_initialize(int list[], int n) {
        int i;
        for (i = 0; i < n; i++)
                list[i] = rand();
}
```