알고리즘 과제(01)

201204025 김대래

과목명 : 알고리즘

담당 교수 : 김동일

분반 : 02

Index

1. 실습 환경

OS : Linux Ubuntu (Virtualbox) / Windows 10

Language : C

Tool : gcc(Linux) / Visual Studio 2015(in Windows)

- 파일 입출력을 확인하기에 Linux 환경이 더 적합하여 주로 Linux환경에서 작업
- 한글의 깨짐 현상에 따라 코드 이미지 캡쳐는 Visual Studio 2015에서 작업

2. 과제 개요

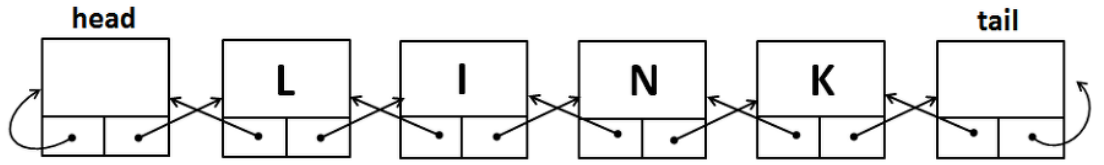
임의의 나열된 수가 저장된 파일(test_test case.txt)을 입력 받고 이중연결 리스트로 구현하여 각각의 정렬 방법(삽입, 버블, 선택)에 따라 오름차순으로 정렬하여 결과를 파일로 출력한다.

입력 예시 : test_100.txt / test_1000.txt / test_10000.txt

출력 예시 : insertion.txt / bubble.txt / selection.txt

3. 주요 개념

3.1. 자료구조 - 이중연결리스트<Doubly Linked List>



이중 연결 리스트는 이전 노드에 대한 링크(prev)와 다음 노드에 대한 링크(next)로 구성되어 있는 리스트이다.

장점은 양방향으로 연결되어 있기 때문에 양뱡향 탐색이 가능하다는 점이다.

DoublyLinkedList 에는 head 노드와 tail 노드로 리스트의 처음과 끝을 가리키며 각 노드에는 자료를 갖고있는 data 와 노드에 대한 링크 정보를 가지는 prev 와 next 가 있다.

3.2. 파일 입출력

C 언어에는 파일 입출력을 위한 다양한 방법이 존재한다.

fopen()/fclose(), fgets()/fputs() fread()/fwrite() 등의 함수를 통해 스트림형식으로 파일의 정보를 가져와 목적에 따른 mode 인자를 통해

읽기/쓰기 등의 일을 할 수 있다.

본 과제에서는 주로 표준 입/출력의 스트림을 여는데 쓰이는 freopen()함수로 좀 더 편하게 파일을 입출력하도록 했다.

freopen()함수는 파일 포인터 인자에 stdin, stdout 를 넣어 입력과 출력을 쉽게 할 수 있다.

3.3. 정렬 알고리즘

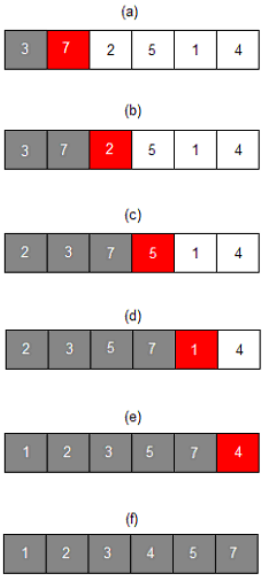
정렬 알고리즘이란 데이터를 번호순 사전 순서와 같이 일정한 순서대로(오름차순, 내림차순 등) 열거하는 방식의 알고리즘이다.

대부분 $O(n^2)$과 $O(n\log n)$의 복잡도를 가진다.

$O(n^2)$의 복잡도를 가진 기초적인 정렬 알고리즘은 삽입, 버블, 선택 정렬이 있다.

본 과제에서는 기초적인 정렬 알고리즘을 다룬다.

3.3.1. 삽입 정렬



삽입 정렬은 앞에서부터 차례대로 순서를 비교하여 자신의 위치를 찾아 삽입함으로써 정렬을 완성하는 알고리즘이다.

1. 처리할 index(빨간색)의 데이터를 앞에서부터 차례대로 크기를 비교하여 자신의 위치를 찾는다.
2. 해당 자리까지의 데이터를 오른쪽으로 한 칸씩 shift 한다.
3. 자신의 위치에 데이터를 삽입한다.

3.3.2. 버블 정렬

버블 정렬은 두 인접한 원소를 검사하여 정렬하는 방법으로 교환(swap)을 통해 데이터를 정렬한다.

3.3.3. 선택 정렬

리스트의 앞에서부터 리스트 중 최소값을 찾아 교체하며 진행하는 방법의 정렬 알고리즘이다.

제자리 정렬 알고리즘의 하나로 메모리가 제한적일 경우 성능상의 이점이 있는 알고리즘이다.

4. 구현 - 이중 연결 리스트 / 파일 입출력

4.1. 파일 입출력

4.1.1. 파일 포맷 지정

```
/* 테스트 케이스를 받기 위한 변수 선언 */
 char testcase[10];
 printf(">> test case num (100 / 1000 / 10000)₩n>> Input : ");
 scanf("%s", testcase);

 /* 테스트 케이스 만큼 반복하기 위한 변수 형변환 */
int test_num = atoi(testcase);

/* 파일명 포맷 */
 char inputFileName[100] = "test_";
 char *format = ".txt";

 /* 파일명 포맷에 맞춰 문자열 병합 */
strcat(inputFileName, testcase);
strcat(inputFileName, format);
```

파일명의 포맷을 맞추기 위하여 입력 받을 파일의 test case 를 입력받고

test_"test case".txt 의 파일명 포맷으로 strcat 함수를 이용하여 문자열을 병합

4.1.2. 파일 입출력

```
/* 스트림 파일 오픈 freopen() */
FILE *input_FD;
input_FD = freopen(inputFileName, "r", stdin);
```

freopen()함수를 통해 파일 포맷에 따라 저장된 파일명을 읽기 모드로 오픈

```
//출력 파일
FILE *output_FD;

start = clock();
/* 정렬 방식에 따른 정렬 */
if (mode == 0) {
    InsertionSort(list);
    end = clock();
    time = (float)(end - start) / CLOCKS_PER_SEC;
    output_FD = freopen("inesertion.txt", "w", stdout);
}
```

정렬 방식에 따른 파일명과 쓰기모드를 통해 출력 결과를 저장

freopen()함수의 인자 stdin, stdout 을 통해 입력과 출력 결과를 읽기/쓰기

4.1.3. 정렬 방법 선택

```
/* 정렬 방법 선택 */
int mode;
printf(">> Input Sort Mode ₩n [0 : Insertion, 1 : Bubble, 2 : Selection ] : ");
scanf("%d", &mode);
```

정렬 방법을 선택하기 위하여 정수형 변수 mode 선언

입력 값에 따라 작동할 수 있도록 저장

4.2. 이중 연결 리스트

4.2.1. 자료구조 정의 (Node, Doubly Linked List)

```c
/*
 Define structure Node, DoublyLinkedList
 */
typedef struct Node {
    int data;
    struct Node *next;
    struct Node *prev;

}Node;

typedef struct DoublyLinkedList
{
    unsigned int count;
    struct Node *head;
    struct Node *tail;

}DLL;
```

DoublyLinkedList 와 Node 자료구조를 정의

DoublyLinkedList 에는 리스트의 처음과 끝을 가리킬 head 와 tail 노드와 list 의 크기를 저장할 count 변수 선언

사용의 편의를 위해 DLL 이라 명칭

Node 에는 정수 data 변수와 이전/다음 노드의 링크를 저장할 prev, next 선언

4.2.2. 리스트 초기화 함수

```c
/* create (a list) */
DLL *createList() {
    DLL *list;
    if ((list = (DLL*)malloc(sizeof(DLL))) == NULL) {
        fprintf(stderr, "Could not allocate memory for a new DoublyLinkedList.₩n");
    }
    list->head = NULL;
    list->tail = NULL;

    list->count = 0;
    return list;
}
```

List 를 할당해주고 head 와 tail, count 를 초기화

4.2.3. 노드 생성 및 추가

```c
/* create (a node) */
Node *createNode(DLL *list, int data) {
    Node *newNode;
    if ((newNode = (Node*)malloc(sizeof(Node))) == NULL) {
        fprintf(stderr, "Could not allocate memory for a new Node.₩n");
    }
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    /* if this is the first node in the list, make it head, else add to end */
    if (list->count == 0) {
        list->head = newNode;
    }
    else {
        Node *move = list->head;
        while (move->next != NULL) {
            move = move->next;
        }
        move->next = newNode;
        newNode->prev = move;
    }
    list->tail = newNode;
    list->count++;
    return newNode;
}
```

노드 생성

newNode 를 할당하고 data 를 저장

만약 리스트가 비어 있으면 해당 노드를 헤드로 지정

그렇지 않으면 리스트의 마지막에 새로운 노드를 추가하고 prev/next 지정

4.2.4. 입력 받은 자료 노드 추가

```c
/* 이중연결리스트 생성(초기화) */
DLL *list = createList();

/* 파일 크기만큼 입력받은 값을 노드생성하여 list에 추가 */
int i;
for (i = 0; i < test_num; i++) {
    int x;
    scanf("%d", &x);
    createNode(list, x);
}
```

리스트를 생성하여 초기화 후

해당 리스트에 파일 크기 만큼의 자료를 읽어 createNode(list, data)를 통해 노드를 추가

4.2.5. 리스트 출력

```c
/* prints a node's data */
void print(Node *node) {
    fprintf(stdout, "[%d]", node->data);
}

/* traverses the list from head to tail, printing data (used in tests) */
void printAll(DLL *list) {
    Node *move = list->head;
    while (move != NULL) {
        print(move);
        move = move->next;
        if (move == list->tail) {
            break;
        }
        printf("<->");
    }
}
```

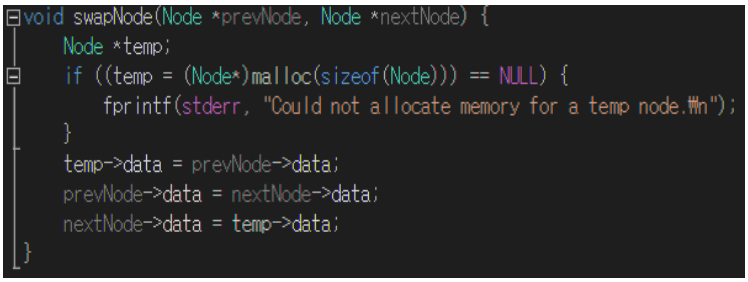
print()함수

노드를 인자로 받아 해당 노드의 data 를 출력

printAll()함수

리스트의 head 부터 다음 노드가 없을 때까지 출력

4.2.6. 데이터 교환



```
void swapNode(Node *prevNode, Node *nextNode) {
    Node *temp;
    if ((temp = (Node*)malloc(sizeof(Node))) == NULL) {
        fprintf(stderr, "Could not allocate memory for a temp node.₩n");
    }
    temp->data = prevNode->data;
    prevNode->data = nextNode->data;
    nextNode->data = temp->data;
}
```

데이터를 임시 저장할 노드 생성 후 교환할 노드인 prevNode 와 nextNode 의 데이터를 교환

5. 과제 1-1 Insertion Sort

5.1. Insertion Sort 의사 코드

```
Insertion_Sort(A)
    for j = 2 to A.length
        key = A[j]
        i = j - 1
        while i > 0 and A[i] > key
            A[i + 1] = A[i]
            i = i - 1
        A[i + 1] = key
```

5.2. InsertionSort(DLL *list)

```
void InsertionSort(DLL * list) {
    Node *i, *j;
    int key;
    //for j = 2 to A.length
    for (j = list->head->next; j != NULL; j = j->next) {
        key = j->data;  // key = A[j]
        i = j->prev;    // i = j - 1
        //while i > 0 and A[i] > key
        while (i != NULL && (i->data) > key) {
            i->next->data = i->data;    //A[i+1] = A[i]
            i = i->prev;                //i = i - 1
        }
        //만약 head가 최소값이면
        if (i == NULL) {
            list->head->data = key;
        }
        else
            i->next->data = key;    //A[i+1] = key
    }
}
```

의사 코드를 기반으로 하여 Doubly Linked List 구조에 맞도록 작성

반복 노드 i, j 선언

2 번째 노드부터 마지막까지 j 반복하며 key 값에 따라 j 의 자리에 맞는 위치를 찾아 삽입

만약, 첫 번째 원소가 가장 작은 수라면 i = i->prev 가 NULL 값이므로 i->next->data 를 정상적으로 받아올 수 없으므로 해당 부분에 대한 조건을 추가

6. 과제 1-2 Bubble Sort

6.1. Bubble Sort 의사 코드

```
Bubble_Sort(A)
    for i = 1 to A.length
        for j = A.length downto i + 1
            if A[j] < A[j - 1]
                swap(A[j], A[j - 1])
```

6.2. BubbleSort(DLL *list)

```
void BubbleSort(DLL * list) {
    Node *i, *j;
    //for i = 1 to A.length
    for (i = list->head; i != NULL; i = i->next) {
        //for j = A.lenght down to i + 1
        for (j = list->tail; j != i; j = j->prev) {
            //if A[j] < A[j - 1]
            if ((j->data) < (j->prev->data)) {
                //swap(A[j], A[j - 1]
                swapNode(j, j->prev);
            }
        }
    }
}
```

반복 노드 i, j 선언

i 가 리스트의 시작부터 끝까지 반복하며

j 가 끝에서부터 i+1 까지 반복할 때

인접한 두 노드의 data 를 비교하여 정렬(swap)

swapNode 는 4.2.6 참조

7. 과제 1-3 Selection Sort

7.1. Selection Sort 의사 코드

```
Selection_Sort(A)
    for i = 1 to A.length
        Min = i
        for j = i + 1 to A.length
            if A[j] < A[Min]
                Min = j
        swap(A[i], A[Min])
```

7.2. SelectionSort(DLL * list)

```
void SelectionSort(DLL * list) {
    Node *i, *j;
    Node *min;

    if ((min = (Node*)malloc(sizeof(Node))) == NULL) {
        fprintf(stderr, "Could not allocate memory for a min node.₩n");
    }

    //for i = 1 to A.length
    for (i = list->head; i != NULL; i = i->next) {
        min = i;    //Min = i
        //for j = i+1 to A.length
        for (j = i->next; j != NULL; j = j->next) {
            //if A[j] < A[Min]
            if (j->data < min->data)
                min = j;    //Min = j
        }
        //swap(A[i], A[Min])
        swapNode(i, min);
    }
}
```

반복 노드 i, j 선언

최솟값을 저장할 min 노드 선언

i 가 리스트의 처음부터 끝까지 반복할 때

min 을 i 로 저장

j 가 i+1 부터 끝까지 반복하며 최솟값을 찾으면 min 을 j 로 저장

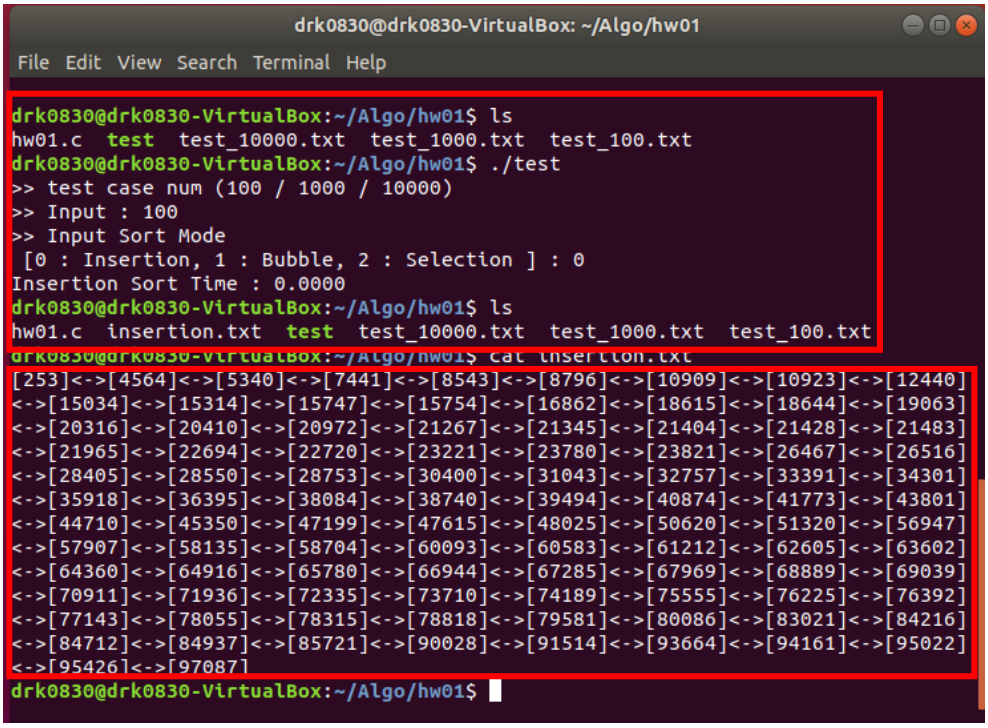
j 가 i+1 부터 끝까지 모두 반복한 후 찾은 최솟값을 i 와 데이터 교환

데이터 교환은 4.2.6 참조

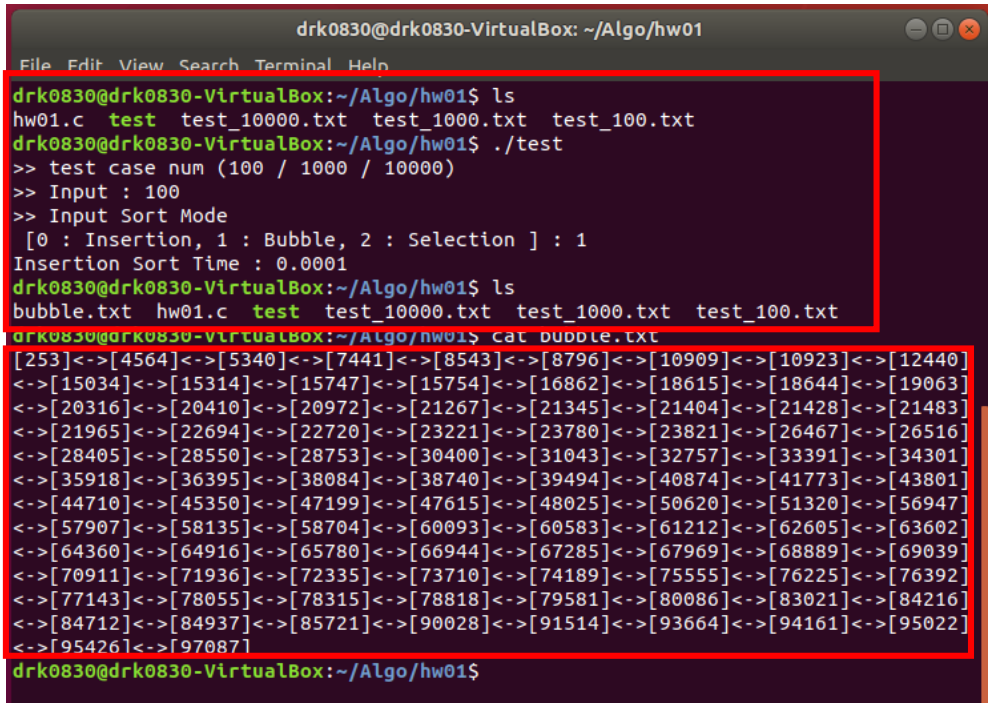
8. 실행 결과

8.1. 정렬 결과 확인

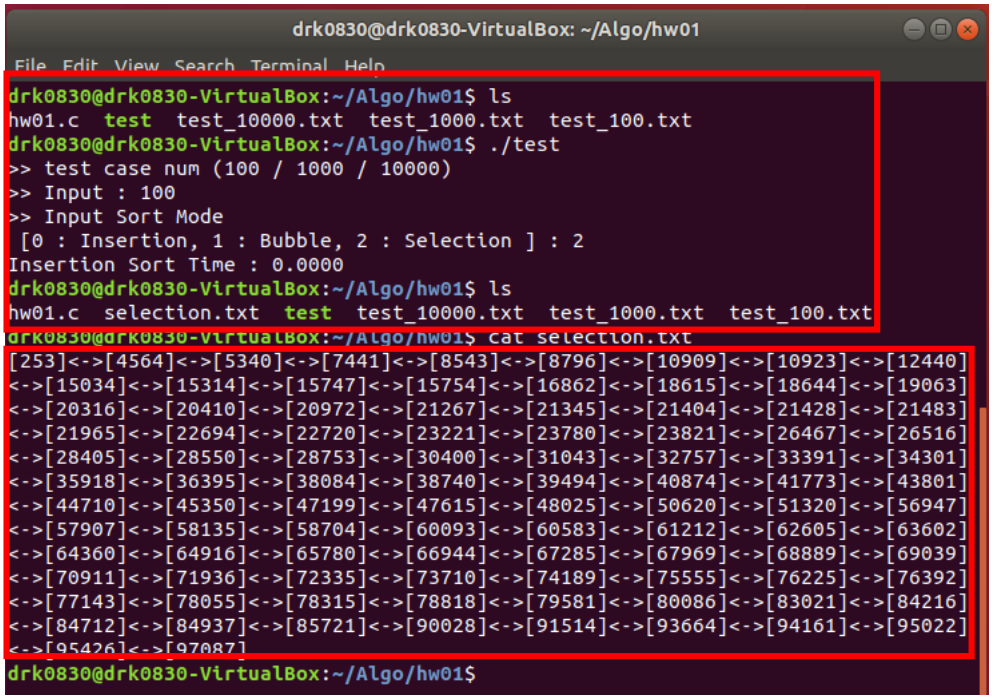
8.1.1. Insertion Sort



8.1.2. Bubble Sort

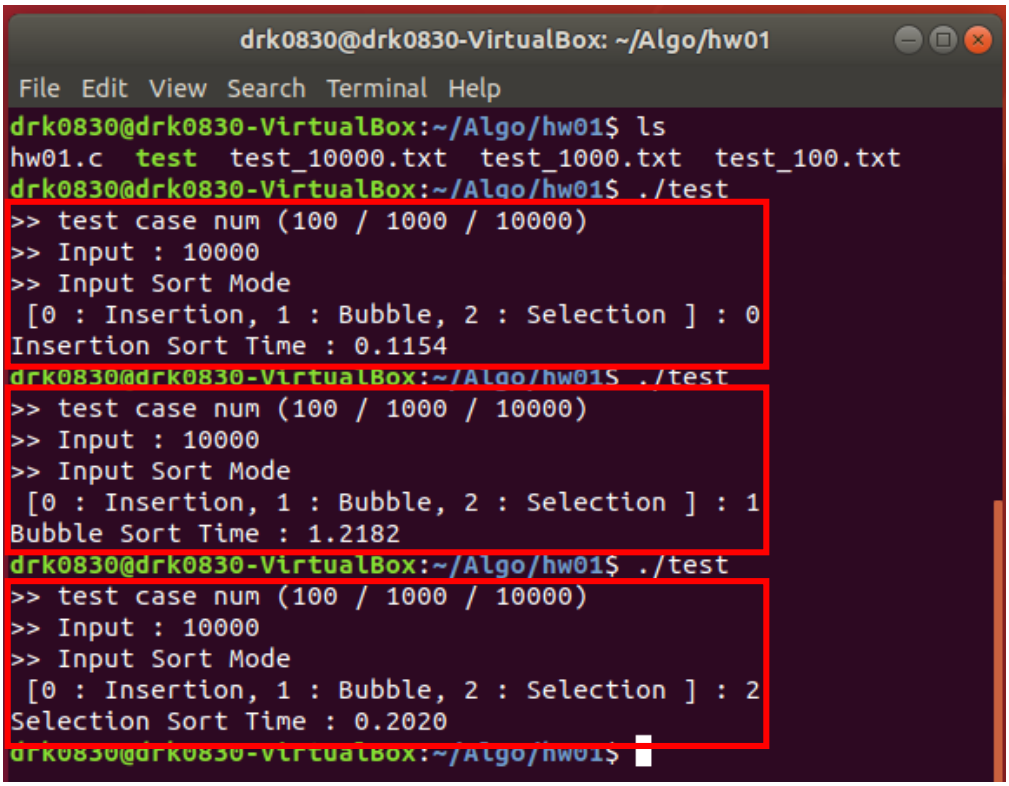


8.1.3. Selection Sort



8.2. 알고리즘 속도 비교

- 적은 양의 데이터로는 속도 비교가 어려워 10000 개의 데이터를 통해 테스트



8.3. 결과

- 삽입 정렬 수행 시간은 Worst case, Average case 모두 $O(n^2)$이다.
  단, 정렬되어 있는 경우 Best case 는 $O(n)$이다.
- 버블 정렬 수행 시간은 Worst case, Average case 모두 $O(n^2)$이다.
  하지만 이미 정렬되어 있어도 모두 순환하므로 Best case 도 $O(n^2)$이다.
- 선택 정렬 수행 시간은 Worst case, Average case 모두 $O(n^2)$이다.
  선택 정렬도 정렬되어 있어도 최소값을 찾는 과정에 따라 Best case 도 $O(n^2)$이다.
- 버블 정렬 vs. 선택 정렬
  두 가지 모두 Best case 는 $O(n^2)$이지만 swap 횟수에 따라 버블 정렬이 수행 시간이 더 길다.
- 이에 따라 위 알고리즘 속도 비교에 삽입 < 선택 < 버블의 결과가 나온다.