12.1 정렬

정렬 물건을 오름차순이나 내림차순으로 나열하는 것

자료가 정렬되어 있다면 탐색의 효율성 up

레코드 정렬시켜야 할 대상

필드 키 레코드를 식별해주는 역할

ex) 이름 학번 주소 연락처 필드 중 학번

정렬 = 레코드를 키값의 순서로 재배열하는 것

정렬 알고리즘 현재 상황에서 가장 효율적인 알고리즘을 골라야함

효율성의 기준 비교연산의 횟수, 이동연산의 횟수

효율로 구분 단순하지만 비효율적인 방법 삽입정렬, 선택정렬, 버블정렬

복잡하지만 효율적인 방법 퀵 정렬, 히프정렬, 합병정렬, 기수정렬

자료의 개수가 커지면 반드시 효율적인 알고리즘 사용

데이터가 저장되는 위치에 따른 구분 내부정렬 정렬하기 전에 모든 데이터가 메인 메모리에 올라와 있는 정렬

외부정렬 외부기억장치에 대부분의 데이터가 있고 일부만 메모리에 올려놓은 상태에서 정렬을 함

안정성 데이터에 동일한 키값을 가진 레코드가 여러 개 존재하는 경우, 레코드의 상대적인 위치가 바뀌지 않는 것

ex) 30 30 10 20🡪 10 20 30 30

안정성 충족 삽입정렬, 버블정렬, 합병정렬

12.2 선택 정렬

선택 정렬의 원리 정렬환경 숫자필드만 가지고 있는 레코드들

레코드들은 1차원 배열에 기록

준비물 2개의 리스트 리스트1, 리스트2

리스트1은 정렬안된 숫자들이 들어있고 리스트2는 비어있음

정렬방법 리스트2에서 가장 작은 숫자를 골라 하나씩 리스트2로 이동

리스트2가 공백에 될 때까지 반복

이점 입력배열 외에 추가적인 메모리공간 필요 X

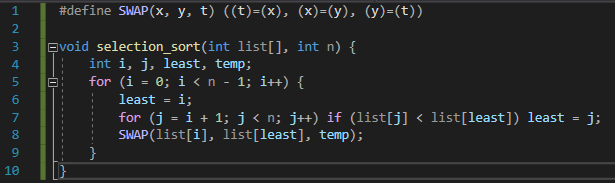
선택 정렬의 알고리즘 알고리즘12.1 selection\_sort(A, n) :

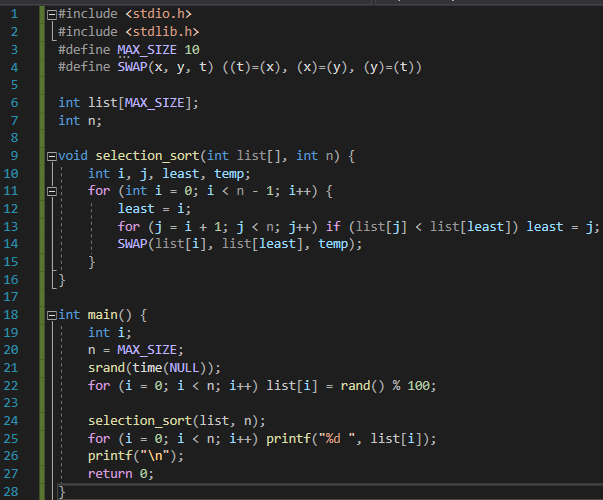
for (int i = 0; i < n - 2; i++) {

least = (배열에서 가장 작은 값의 인덱스);

A[i]와 A[least]와 교환

}

프로그램12.1 

전체 프로그램 프로그램12-2 

선택 정렬의 분석 시간복잡도 O(n2)

Quiz 1. 1 3 4 9 7 6 🡪 1 3 4 7 9 6 🡪 1 3 4 6 9 7 🡪 1 3 5 6 7 9

2. 1 4 4 2 3

12.3 삽입 정렬

삽입 정렬의 원리 정렬되어 있는 리스트에 새로운 레코드를 적절한 위치에 삽입 반복

입력배열을 정렬된 부분과 정렬되지 않은 부분으로 나눠서 사용하면 됨

삽입 정렬의 알고리즘 알고리즘12.2 insertion\_sort(A, n) :

for (i = 1 to i = n-1) //인덱스 1부터 시작

key = A[i]; //현재 삽입될 숫자인 i번째 정수를 key변수로 복사

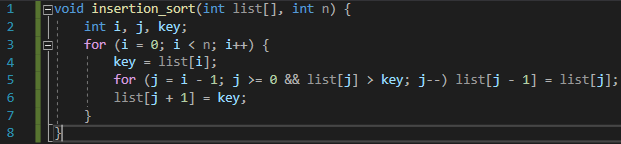
j=i-1 //현재 정렬된 배열은 i-1까지 이므로 i-1부터 역순으로 조사

while(j>=0 && A[j] > key) //j값이 음수가 아니어야 되고, key값보다 정렬된 배열에 있는 값이 크면

A[j+1] = A[j] //j번째 정수를 j+1번째로 이동

j=j-1 //j를 하나 감소

A[j+1] = key //j번째 정수가 key보다 크다는 조건에서 탈출 --> j+1번째에 key값이 들어감

삽입 정렬의 C언어 구현 프로그램12.3 

삽입 정렬의 복잡도 분석 시간복잡도 O(n2)

레코드 양, 크기가 클 경우엔 적합하지 않음

Quiz 1. 379416 🡪 347916 🡪 134796 🡪 134679

2. 레코드 값이 같으면 삽입하지 않고 그 다음 인덱스로 넘어가기 때문

12.4 버블 정렬

버블 정렬의 원리 인접한 2개의 레코드르 서로 비교 🡪 크기가 순서대로 되어있지 않으면 교환

왼쪽부터 오른쪽으로 진행

1번 진행하면 가장 큰 레코드가 오른쪽으로 이동됨

정렬안된 왼쪽 리스트에 반복해 적용하면 정렬 완료

버블 정렬의 알고리즘 알고리즘12.3 Bubble\_sort(A, n) :

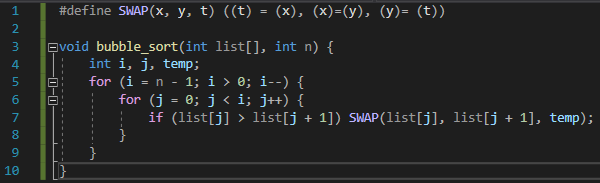
for (i = n - 1 -- > i = 0)

for (j = 0 -- > j = i - 1)

A[j]와 A[j + 1]이 크기 순이 아니면 교환

j++;

i++;

버블 정렬의 C언어 구현 프로그램12.4 

버블 정렬의 복잡도 분석 시간복잡도 O(n2)

Quiz의 복잡도 분석 1. 379416 🡪 374169 🡪 341679 🡪 314679 🡪 134679

2. 순서에 맞지 않는 요소를 인접요소와 교환함

12.5 쉘 정렬

쉘 정렬의 원리 삽입정렬이 어느정도 정렬된 배열에선 대단히 빠른 것에서 착안

정렬할 리스트를 일정한 기준에 따라 분류 🡪 여러 개의 부분리스트를 만듬 🡪 각 부분리스트를 삽입정렬로 정렬 🡪 전체리스트를 다시 부분리스트로 만듬 🡪 부분리스트가 1개가 될때까지 반복

부분리스트 구성 전체리스트의 k(간격)번째 요소를 추출해 만듬

각 패스마다 간격 k를 줄여감 🡪 하나의 부분리스트에 속하는 레코드의 개수는 증가됨

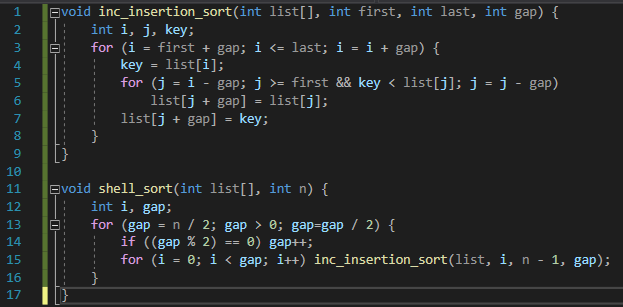
보통 간격은 처음에 n/2로 하고 갈수록 절반으로 줄임

쉘 정렬의 구현 gap 간격, 부분리스트의 개수

짝수이면 1을 더해줌

shall\_sort 간격이 1이 될 때까지 간격을 1/2로 줄이면서 반복하는 함수

inc\_insertion\_sort 각 부분리스트에 대해 일정한 간격으로 떨어져있는 요소들을 삽입정렬하는 함수

프로그램 12.5 

쉘 정렬의 분석 시간복잡도 보통 O(n1.5)

최악 O(n2)

Quiz 1. 48576213 🡪 13246578 🡪 12345678

2. 100배

12.6 합병 정렬

합병정렬의 개념 분할정복 기법 이용 하나의 리스트를 두개의 균등한 크기로 분할 🡪 분할된 부분리스트 정렬 🡪 두개의 정렬된 부분리스트 합함

순환호출 이용

합병정렬의 단계 분할 입력 배열을 같은 크기의 2개의 부분배열로 분할

정복 부분배열을 정렬, 부분배열의 크기가 충분히 작지 않으면(배열의 크기가 1일 때 까지) 순환호출로 다시 분할

각 부분배열을 정렬할 때도 합병정렬을 순환적으로 적용

결합 정렬된 부분 배열을 하나의 배열에 통합

합병정렬 알고리즘 알고리즘12.4 merge\_sort(list, left, right) :

if (left < right)

mid = (left + right) / 2

merge\_sort(list, left, mid)

merge\_sort(list, mid + 1, right)

merge(list, left, mid, right)

정렬이 이뤄지는 부분 2개의 리스트를 합병하는 단계

합병알고리즘 2개 리스트의 요소를 처음부터 하나씩 비교해 더 작은 요소를 새로운 리스트로 옮김

두 개의 리스트 중에서 하나가 먼저 끝날 때 까지 되풀이

둘 중 하나의 리스트가 먼저 끝나면 나머지 리스트의 요소들을 새로운 리스트로 복사

알고리즘12.5 merge(list, left, mid, last):

i=left

j=mid+1

k=left

sorted배열 생성

while (i <= mid and j <= right) {

if (list[i] <= list[j]) {

sorted[k] = list[i]

k++

i++

}

else {

sorted[k] = list[j]

k++

j++

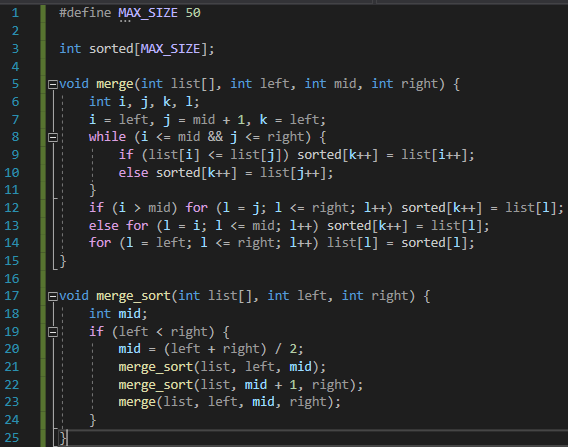
}

}

요소가 남아있는 부분배열을 sorted로 복사

sorted를 list로 복사

합병정렬의 C언어 구현 merge\_sort함수에서 배열을 2등분 🡪 각각에 부분배열에 merge\_sort를 순환호출 🡪 부분배열에 숫자가 1개 남을 때까지 반복 🡪 분할과정이 끝나면 merge함수를 이용해 합병

프로그램12.6 

합병정렬의 복잡도 분석 시간복잡도 O(nlog2n)

장점 안정적인 정렬방법

단점 임시 배열 필요

레코드의 크기가 큰 경우 큰 시간낭비 but 연결리스트로 레코드를 구성한다면 상관없음

Quiz 1. 8 2 5 7 6 4 1 3 🡪 2 8 5 7 6 4 1 3 🡪 2 8 5 7 4 6 1 3 🡪 2 5 7 8 1 3 4 6 🡪 1 2 3 5 6 7 8

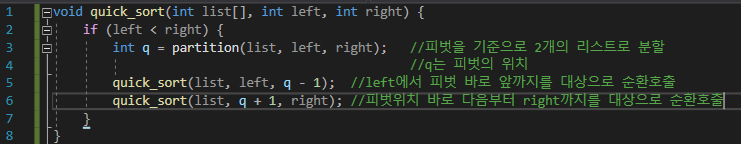
2. 비교연산, 이동연산에 그렇게 큰 차이가 없기 때문

12.7 퀵 정렬

퀵 정렬의 개념 매우 빠른 수행 속도

분할정복 기법 사용 but 합병정렬과는 다르게 비균등하게 분할함

리스트 안에 있는 한 요소를 피벗으로 정함 🡪 피벗보다 작은 요소는 왼쪽으로, 큰 요소는 오른쪽으로 이동 🡪 왼쪽∙오른쪽 요소에 대해 퀵정렬

퀵 정렬의 알고리즘 프로그램12.7 

low 왼쪽 부분리스트 인덱스

왼쪽에서 오른쪽으로 탐색하다가 피벗보다 큰 데이터를 찾으면 멈춤

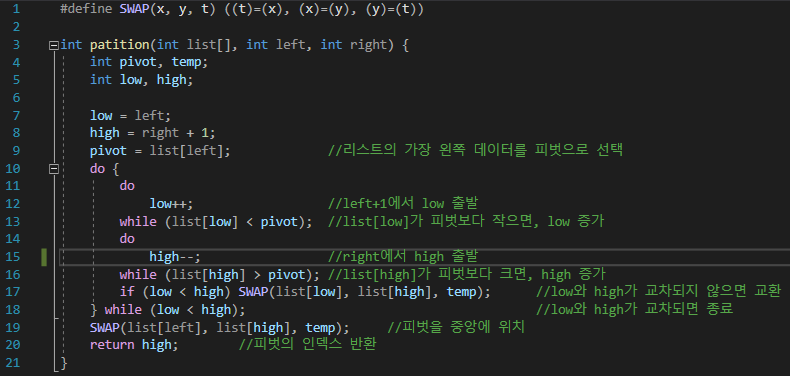
high 오른쪽 부분리스트 인덱스

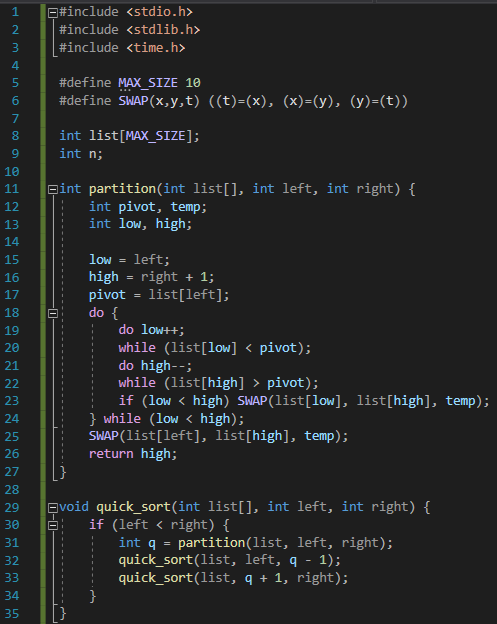
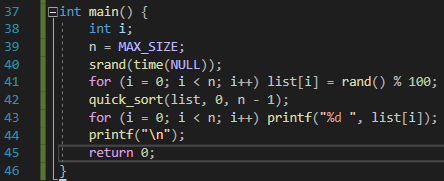
오른쪽에서 왼쪽으로 탐색하다가 피벗보다 작은 데이터를 찾으면 멈춤

데이터교환 탐색이 멈춰진 위치는 각 부분리스트에 적합하지 않은 데이터 🡪 low와 high의 데이터를 서로 교환

피벗데이터 위치 이동 초기 피벗위치는 중앙에 있지 않음

low와 high가 서로 엇갈리게 되면, 멈춰서 그 사이에 피벗을 위치시킴

프로그램12.8 

전체 프로그램 프로그램12.9  

퀵 정렬의 복잡도 분석 시간복잡도 O(nlog2n)

퀵 정렬 라이브러리 함수의 사용 함수의 원형 void qsort(void\* base, size\_t num, size\_t width, int (\*compare)(const void\*, const void\*));

base 배열의 시작 주소

num 배열요소의 개수

width 배열요소 하나의 크기(바이트 단위)

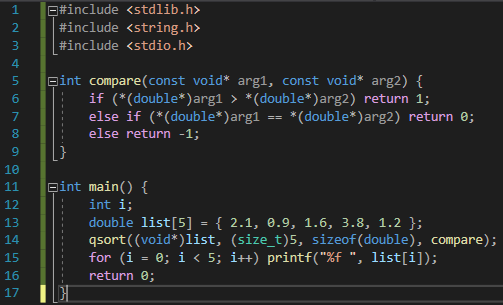
compare 포인터를 통해 두 개의 요소를 비교해 비교결과를 정수로 반환하는 함수

함수의 선언 int compare((void\*)elem1, (void\*)elem2);

결과 <0 elem1이 elem2보다 작음

0 elem1이 elem2와 같음

>0 elem1이 elem2보다 큼

함수의 사용예 프로그램12.10 

Quiz 1. 4 7 1 8 6 2 5 3 🡪 4 3 1 8 6 2 5 7 🡪 4 3 1 2 6 8 5 7 🡪 3 1 2 4 6 8 5 7 🡪 1 2 3 4 5 6 7 8

2. 50,000,000,000

12.8 히프정렬

히프정렬의 개념 우선순위 큐를 완전이진트리로 구현하는 방법

최대값, 최소값을 쉽게 추출

정렬한 배열을 먼저 최소 히프로 변환, 가장 작은 원소부터 차례대로 추출

Quiz 1. 1 2 4 6 3 7 5 8 🡪 8 1 2 4 6 3 7 5🡪 8 6 7 1 2 4 3 5 🡪 8 6 7 5 4 3 2 1

12.9 기수정렬

기수정렬의 원리 비교연산없이 데이터 정렬

시간복잡도 O(kn)

장점 매우 작은 시간복잡도

단점 정렬할 수 있는 레코드의 타입 제한

레코드가 동일한 길이를 가져야 함

기수 숫자의 자릿수

ex) 숫자 42 🡪 4,2

기수 = 기수정렬의 단계

10진수에선 0~9까지의 값만 가짐 🡪 10개의 버킷(0,1,2,3,4,5,6,7,8,9)을 만듬 🡪 입력데이터를 각 자리수의 값에 따라 상자에 넣음 🡪 순차적으로 버킷 안에 들어있는 상자를 읽음

ex) [8,2,7,3,5]

1=[], 2=[2], 3=[3], 4[], 5=[5]… 🡪 순서대로 읽으면 [2,3,5,7,8]

ex) [28,93,39,81,62,72,38,26]

1. 1의 자리에 넣음 1=[81], 2=[62,72], 3=[93], 6=[26], 8=[28,38], 9=[39]

2. 정렬 [81, 62, 72, 93, 26, 28, 38, 39]

3. 10의 자리에 넣음 2=[26, 28], 3=[38, 39], 6=[62], 7=[72], 8=[81], 9=[93]

4. 정렬 [26, 28, 38, 39, 62, 72, 81, 93]

기수정렬의 알고리즘 알고리즘12.6 RadixSort(list, n) {

for (d = LSD의 위치 to MSD의 위치) {

d번째 자릿수에 따라 0번부터 9번 버킷에 넣음

버킷에서 숫자들을 순차적으로 읽어 하나의 리스트로 합침

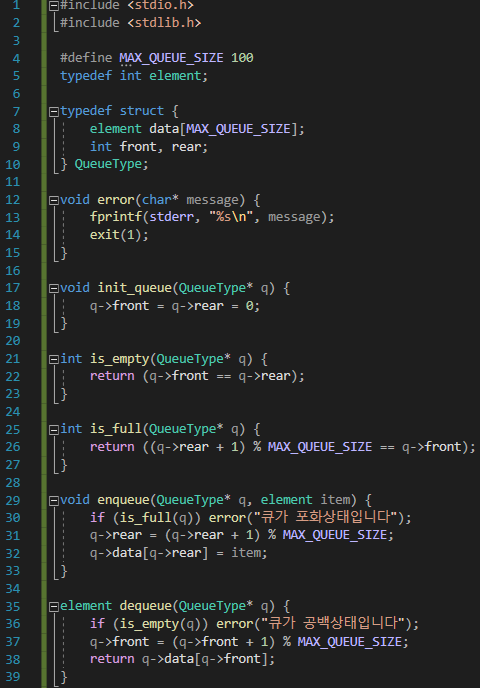
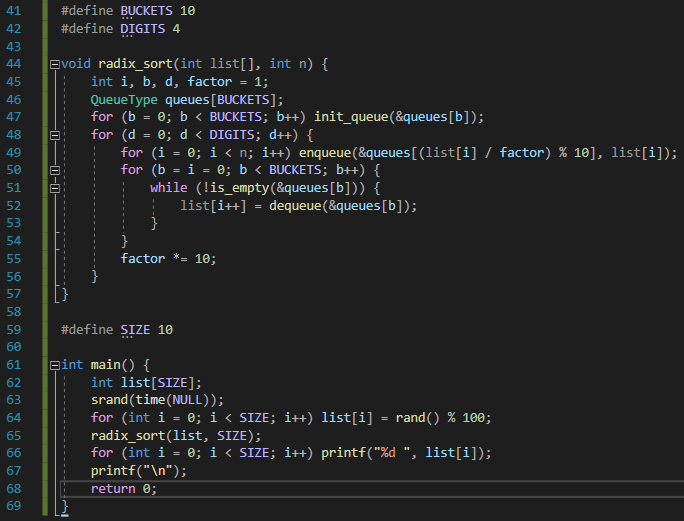
d++

}

}

각각의 버킷은 큐로 대체 큐의 삽입 버킷에 숫자를 집어넣음

큐의 삭제 버킷에서 숫자를 읽음

기수정렬의 구현 프로그램12.11  

기수정렬의 분석 시간복잡도 O(n)

Quiz 1. 52 87 42 77 53 47 85 72 🡪 52 42 72 53 85 87 77 37 🡪 37 42 52 53 72 77 85 87  
 2. 기수정렬

12.10 정렬 알고리즘의 비교

최선 평균 최악

삽입정렬 O(n) O(n2) O(n2)

선택정렬 O(n2) O(n2) O(n2)

버블정렬 O(n2) O(n2) O(n2)

쉘정렬 O(n) O(n1.5) O(n1.5)

퀵정렬 O(nlog2n) O(nlog2n) O(nlog2n)

히프정렬 O(nlog2n) O(nlog2n) O(nlog2n)

합병정렬 O(nlog2n) O(nlog2n) O(nlog2n)

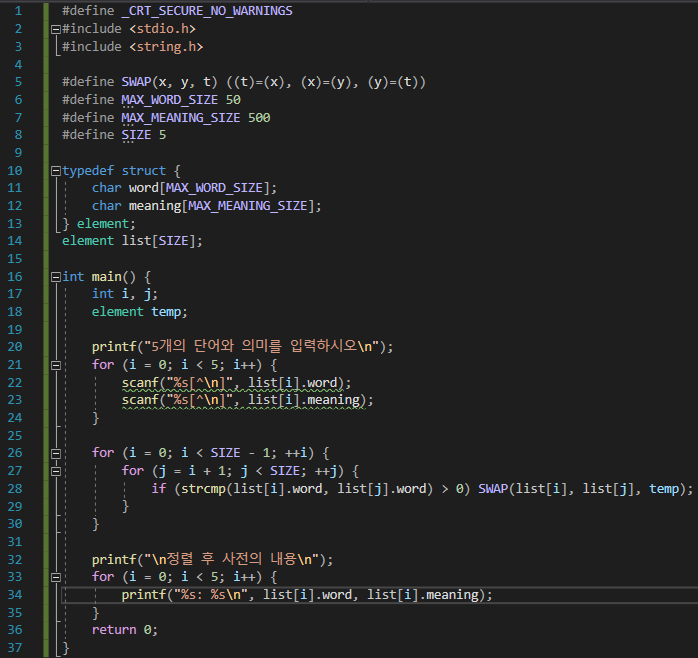
기수정렬 O(dn) O(dn) O(dn)

12.11 정렬의 응용: 영어 사전을 위한 정렬

영어사전을 배열로 구현한다 가정

구조체에 단어와 의미 저장 🡪 구조체의 배열 🡪 배열 정렬

버블정렬 사용

프로그램12.12 

연습문제

1. 4

2. 2

3. 1

4. 4

5. (1) 7 4 9 6 3 8 7 5 🡪 3 4 9 6 7 8 7 5 🡪 3 4 5 6 7 8 7 9 🡪 3 4 5 6 7 7 8 9

(2) 7 4 9 6 3 8 7 5 🡪 4 7 9 6 3 8 7 5 🡪 4 6 7 9 3 8 7 5 🡪 3 4 6 7 9 8 7 5 🡪 3 4 6 7 8 9 7 5 🡪 3 4 6 7 7 8 9 5 🡪 3 4 5 6 7 7 8 9

(3) 7 4 9 6 3 8 7 5 🡪 4 7 6 3 8 7 5 9 🡪 4 6 3 7 7 5 8 9 🡪 4 3 6 7 5 7 8 9 🡪 3 4 6 5 7 7 8 9 🡪 3 4 5 6 7 7 8 9

(4) 7 4 9 6 3 8 7 5 🡪 3 4 7 5 7 8 9 6 🡪 3 4 7 5 6 8 9 7 🡪 3 4 6 5 7 7 9 8 🡪 3 4 5 6 7 7 8 9

6. (1) 71 49 92 55 38 82 72 53 🡪 38 49 53 55 71 82 72 92 🡪 38 49 53 55 71 72 82 92

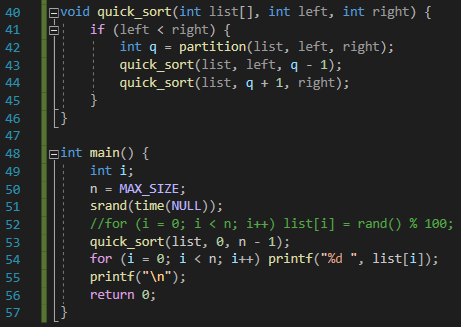
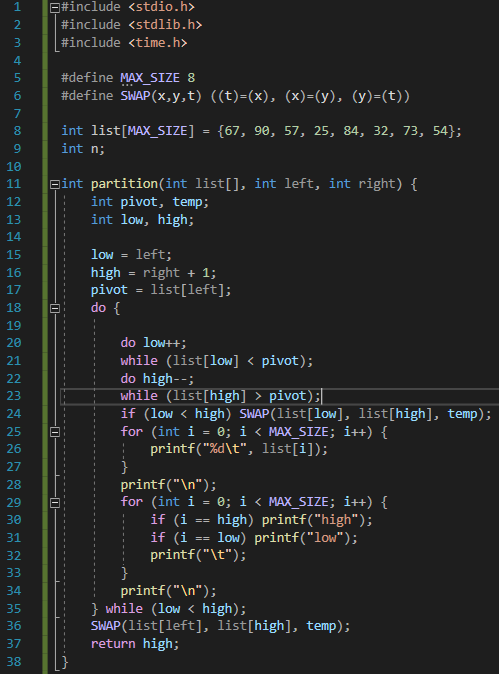
(2) 71 49 92 55 38 82 72 53 🡪

(3) 71 49 92 55 38 82 72 53 🡪 49 71 92 55 38 82 72 53 🡪 49 71 55 92 38 82 72 53 🡪 49 55 71 92 38 82 72 53 🡪 49 55 71 92 38 82 72 53 🡪 49 55 71 92 38 82 53 72 🡪 38 49 53 55 71 72 82 92

(4) 0 92 55 82 53 38 71 72 49 🡪 0 82 55 72 53 38 71 72 92 🡪 0 72 55 71 53 38 49 82 92 🡪 0 71 55 49 53 38 72 82 92 🡪 0 55 53 49 38 71 72 82 92 🡪 0 53 38 49 55 71 72 82 92 🡪 0 49 38 53 55 71 72 82 92 🡪 0 38 49 53 55 71 72 82 92

7. (1) 1 2 3 4 5 6 7 8 9 🡪 1 2 3 4 5 6 7 8 9 🡪 1 2 3 4 5 6 7 8 9 🡪 1 2 3 4 5 6 7 8 9 🡪 … 🡪 1 2 3 4 5 6 7 8 9 🡪 1 2 3 4 5 6 7 8 9

(2) 1 2 3 4 5 6 7 8 9 🡪 1 2 3 4 5 6 7 8 9 🡪 1 2 3 4 5 6 7 8 9 🡪 1 2 3 4 5 6 7 8 9

8. 

9. 3

10. 2

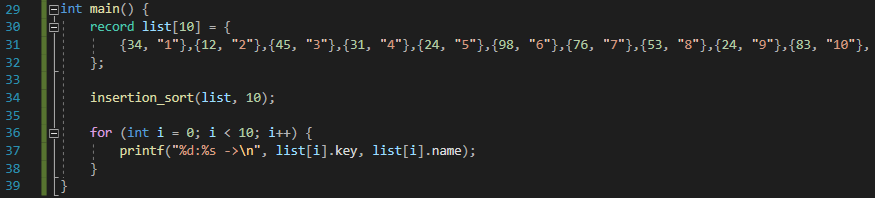
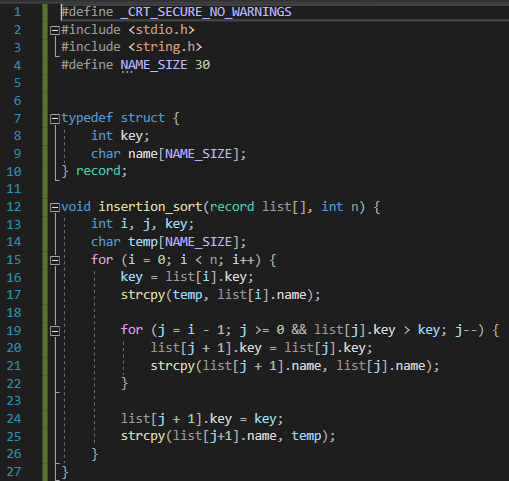
11. (a) 5 3 4 5 8 9 6 7

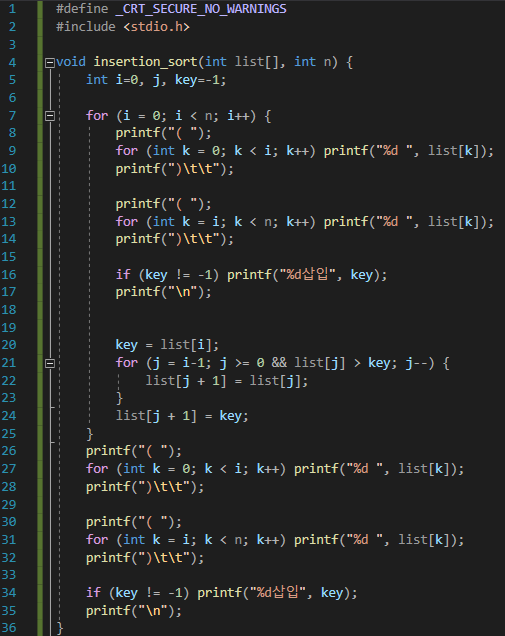
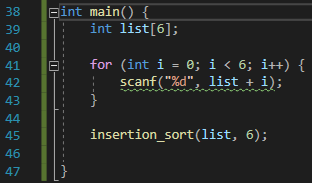
(b) 7번

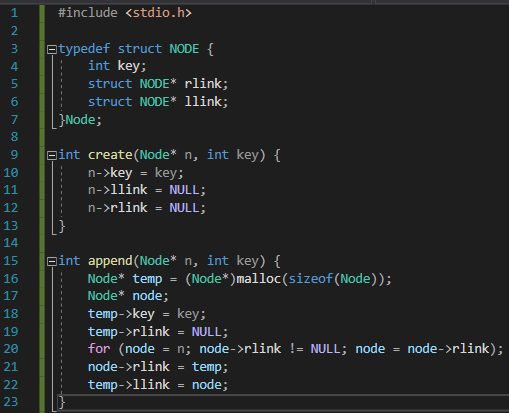
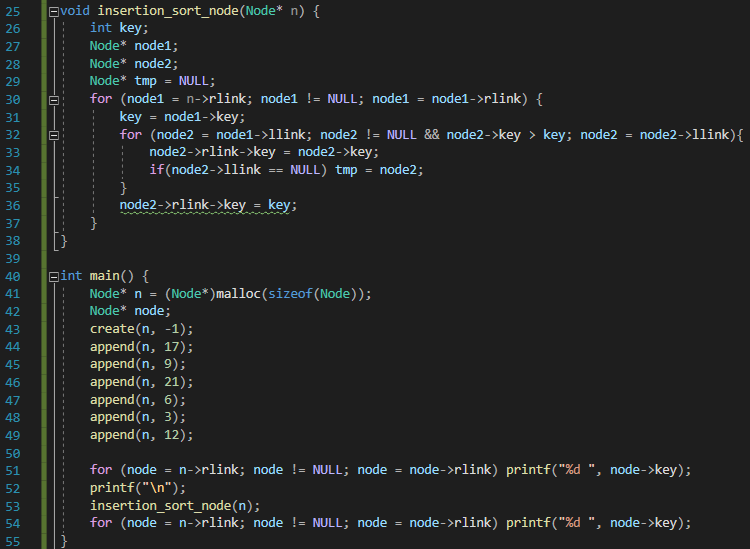
(c) 피벗값은 이미 정렬되어 있기 때문에 변하지 않음

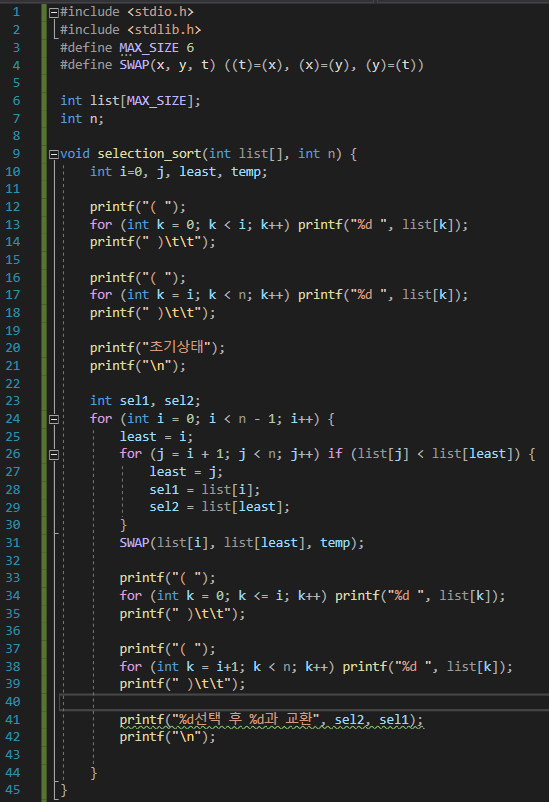
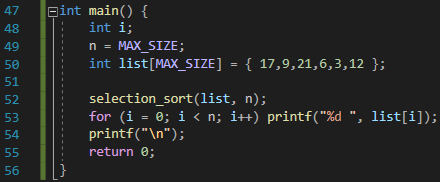
(d) quick\_sort(list, 0, 2), quick\_sort(list, 4, 7)

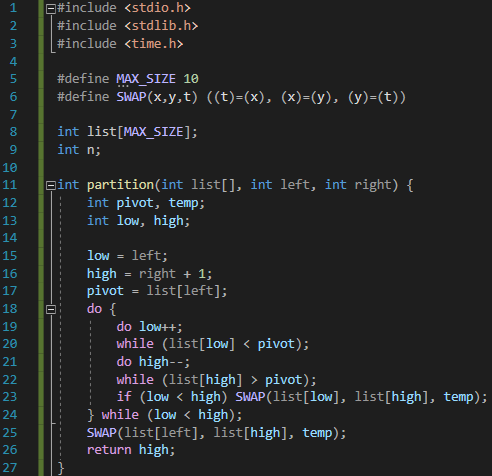
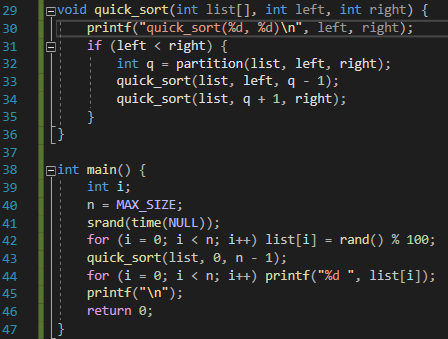
12. 210 220 123 003 513 294 398 528 019 129 🡪 003 210 513 019 220 123 129 294 398 🡪 003 019 123 129 210 220 294 398 513 528

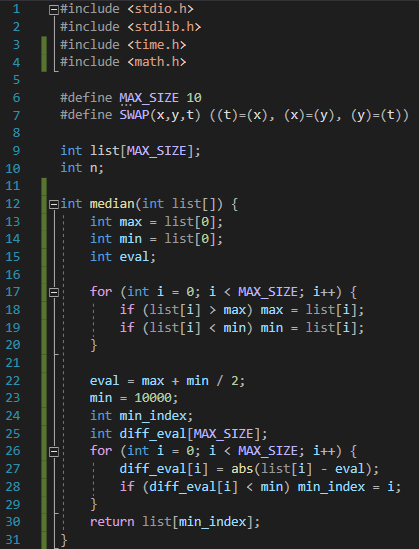
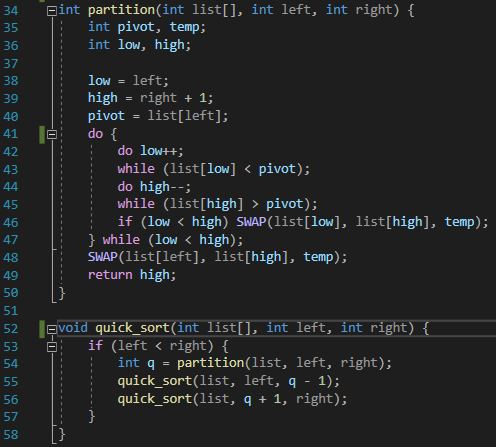
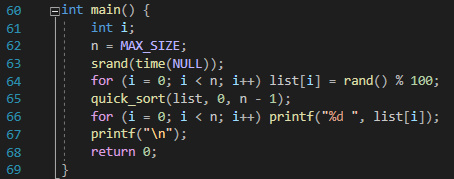
13. 

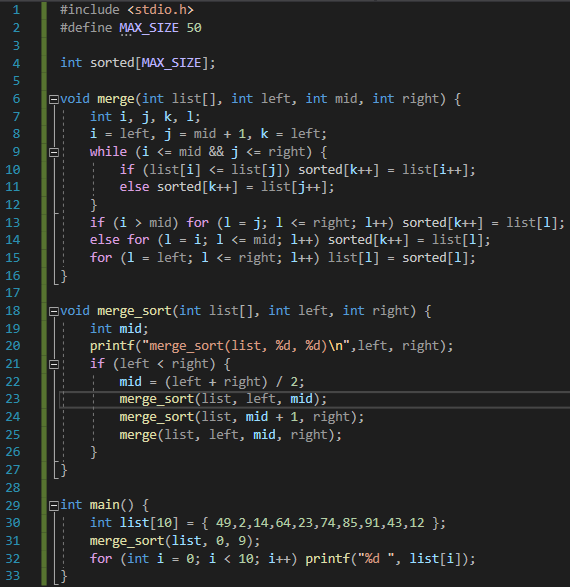
14.  

15.  

16.  

17.  

18.   

19. 

20.

21. 