[CSE3081(2반)] 알고리즘 설계와 분석

2020학년도 2학기

강의자료

(2020.10.01 목요일)

서강대학교 공과대학 컴퓨터공학과
임 인 성 교수

- 본 강의에서 제작하여 제공하는 PDF 파일, 동영상, 그리고 예제 코드 등의 강의 자료의 저작권은 특별히 명기되어 있지 않은 한 서강대학교에 있습니다.
- 본인의 학습 목적 외에 공개된 장소에 올리거나 타인에게 배포하는 등의 행위를 금합니다. 협조 부탁합니다.

[주제 3]
Divide-and-Conquer Techniques
and
Sorting Techniques

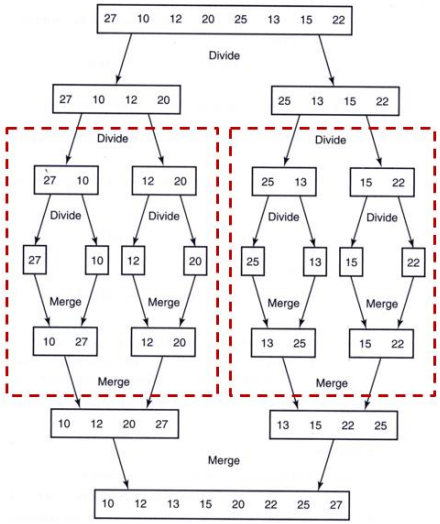
Quick Sort

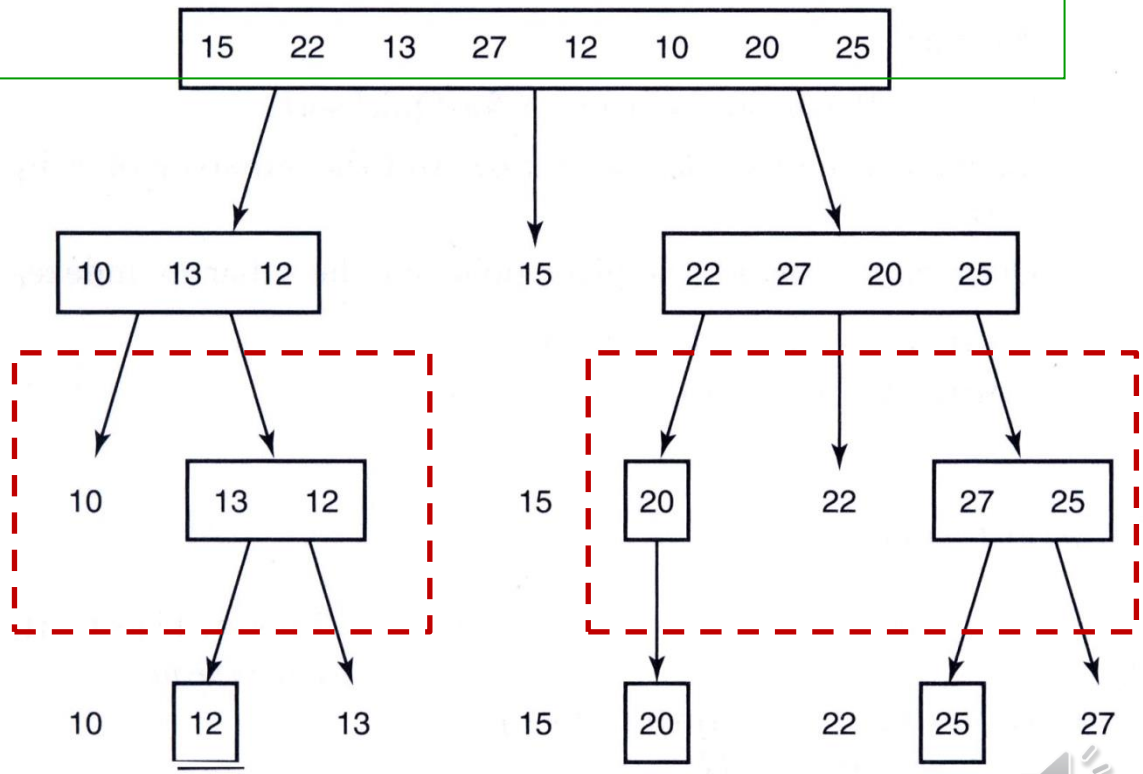
Pivot strategy

① **Divide:** Select a **pivot element**, and then divide the array into two subarrays such that ….

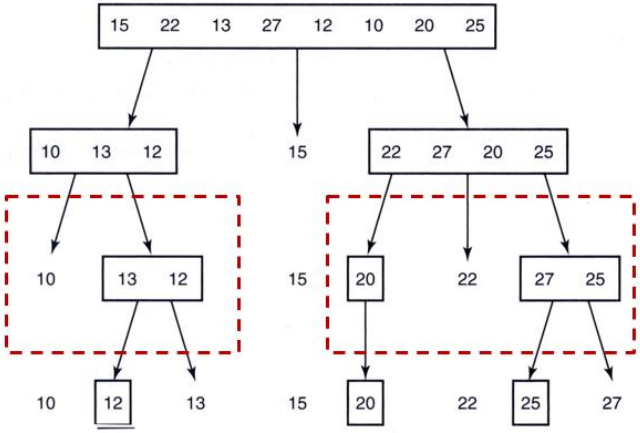
② **Conquer:** sort each subarray recursively.

③ **Combine:** do nothing.





- A **simple** implementation




```
// Sort a list from A[left] to A[right].
// Should be optimized for higher efficiency!!!

void quick_sort(item_type *A, int left, int right) {
  int pivot;

  if (right - left > 0) {
    pivot = partition(A, left, right);

    quick_sort(A, left, pivot - 1);
    quick_sort(A, pivot + 1, right);
  }
}
```

Divide

Conquer

```
#define SWAP(a, b) { item_type tmp; tmp = a; a = b; b = tmp; }

int partition(item_type *A, int left, int right) {
  int i, pivot;

  pivot = left;
  for (i = left; i < right; i++) {
    if (A[i] < A[right]) {
      SWAP(A[i], A[pivot]);
      pivot++;
    }
  }
  SWAP(A[right], A[pivot]);
  return(pivot);
}
```

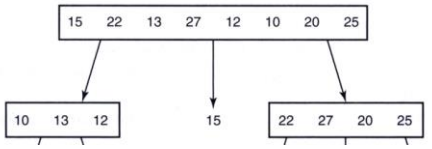
How is the pivot element chosen in this function?

18 20 28 0 38 8 2 16 10 14 24 30 34 12 32 22 6 4 36 26

18 20 0 8 2 16 10 14 24 12 22 6 4 **26** 32 38 30 34 36 28

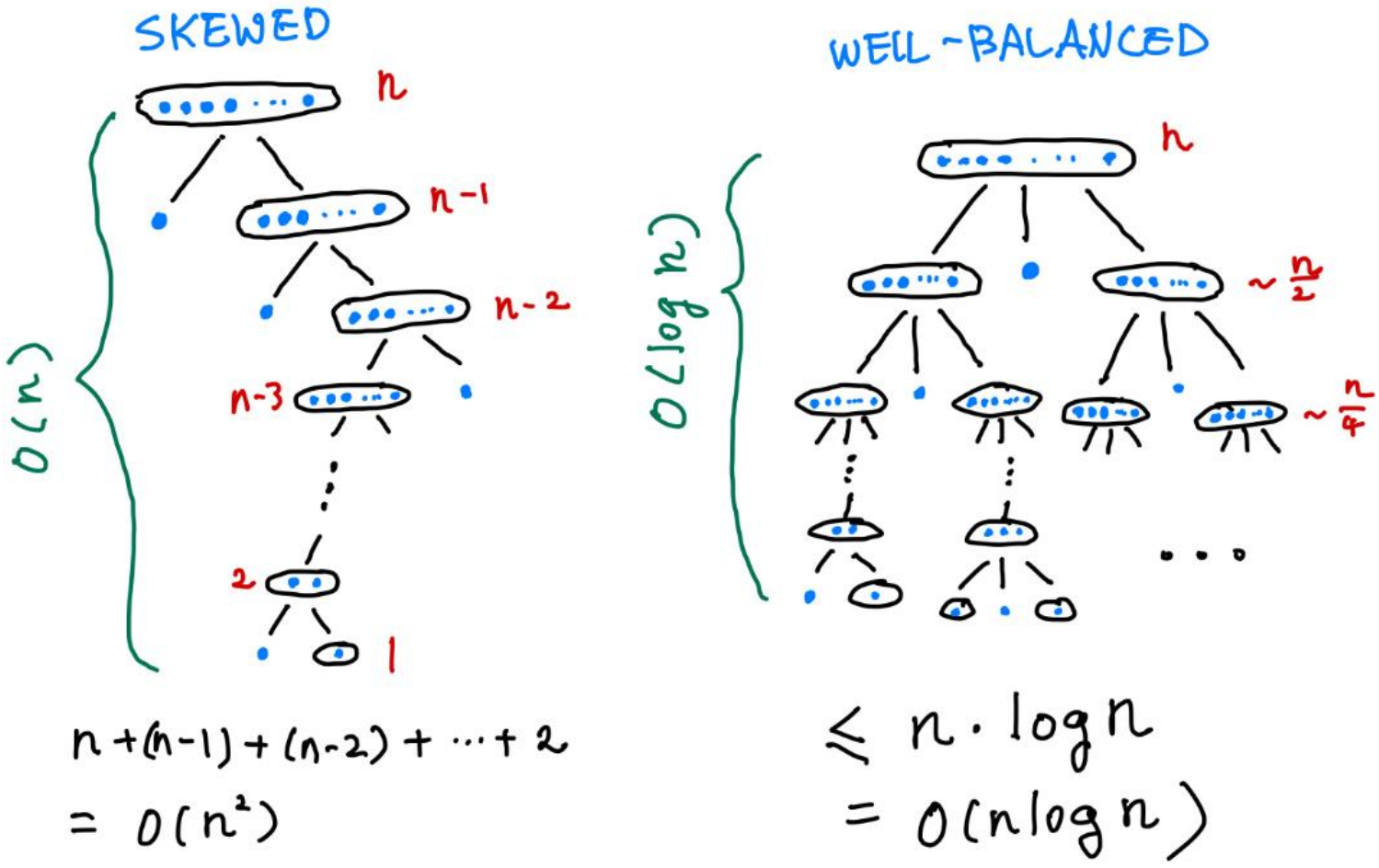
(13)

직관적인 시간 복잡도 추정



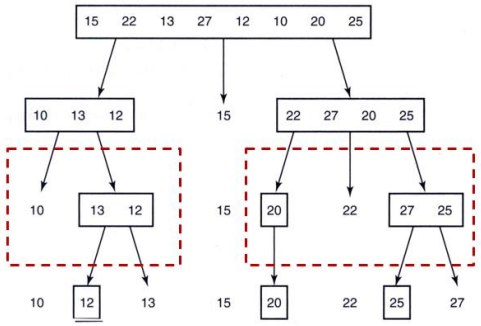
$$T(n) = T(m_1) + T(m_2) + cn \ (m_1 + m_2 = n - 1) \text{ if } n > 1$$
$$T(1) = 1$$

→ $O(n \log n) \leq T(n) \leq O(n^2)$



Cost Analysis

Quick Sort		
Divide	**Conquer**	**Combine**
O(n)	T(m1) +T(m2)	O(1) or 0



- **Cost**

$$T(n) = T(m_1) + T(m_2) + cn \ (m_1 + m_2 = n - 1) \text{ if } n > 1$$
$$T(1) = 1$$

- **Worst-case time complexity**
  - 매 단계에서 선택한 pivot element가 가장 크거나 가장 작을 경우,

$$T(n) = T(0) + T(n-1) + cn$$

Skewed vs well-balanced trees

$$T(n) = T(n-1) + cn, \text{ if } n > 1$$
$$T(1) = 1$$

$$\rightarrow \quad T(n) = O(n^2)$$

- **Average-case time complexity**

$$T(n) = \Sigma_{p=1}^{n} \frac{1}{n} \{T(p-1) + T(n-p)\} + cn$$
$$T(0) = 1$$

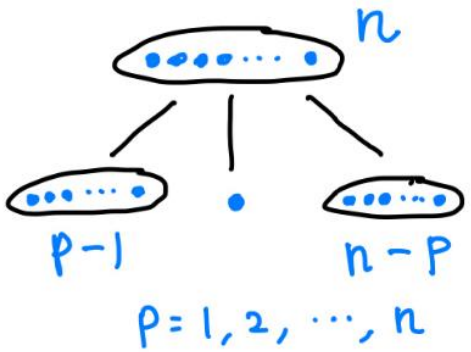
$$\rightarrow \quad T(n) = O(n \log n)$$

Average Case Time Complexity

첫 번째 사실: 0보다 같거나 큰 정수 n에 대해 $T_{ave}(n)$을 n개의 원소를 가지는 배열을 퀵정렬 방법을 사용하여 정렬하는데 걸리는 평균 수행시간이라고 하자. 그러면 어떤 양의 정수 b와 c에 대해 다음과 같은 재귀적인 관계가 존재한다.

$$\begin{aligned} T_{ave}(n) &\leq cn + \frac{1}{n}\sum_{p=1}^{n}\{T_{ave}(p-1) + T_{ave}(n-p)\} \\ &= cn + \frac{2}{n}\sum_{p=0}^{n-1} T_{ave}(p) \text{ for all } n \geq 2, \\ T_{ave}(1) &\leq b, \\ T_{ave}(0) &\leq b. \end{aligned}$$

$$Cost_{ave} = \sum_{p=1}^{n} Pr(p) \cdot Cost(p) = \frac{1}{n}\sum_{p=1}^{n}\{\cdots + \cdots\}$$



두 번째 사실: $k = 2(b+c)$라 할 때, 2보다 같거나 큰 모든 정수 n에 대해 $T_{ave}(n) \leq kn\log_e n$과 같은 관계가 존재한다.

증명: 위의 부등식을 수학적 귀납법을 사용하여 증명하자. 첫째, $n = 2$일 경우, 첫 번째 사실로 부터 다음과 같은 관계가 성립하며,

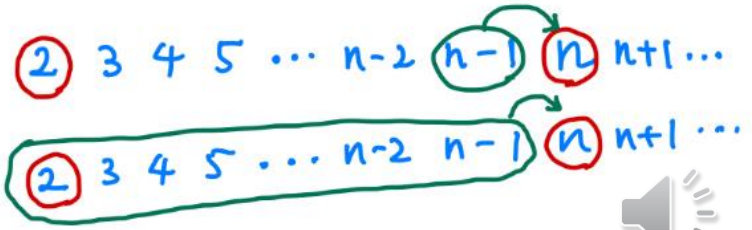
$$T_{ave}(2) \leq 2c + T_{ave}(0) + T_{ave}(1) \leq 2(b+c) \leq k \cdot 2 \cdot \log_e 2$$

따라서 두 번째 사실이 성립한다. 둘째, 3보다 같거나 큰 임의의 n이 주어졌을 때, $m < n$인 모든 m에 대하여 두 번째 사실이 성립한다고 가정하자. 그러면, 첫 번째 사실과 이 가정을 사용하여 다음과 같은 관계를 유도할 수 있으며,

$$\begin{aligned} T_{ave}(n) &\leq cn + \frac{2}{n}\sum_{m=0}^{n-1} T_{ave}(m) \\ &= cn + \frac{2}{n}\{T_{ave}(0) + T_{ave}(1)\} + \frac{2}{n}\sum_{m=2}^{n-1} T_{ave}(m) \\ &\leq cn + \frac{4b}{n} + \frac{2k}{n}\sum_{m=2}^{n-1} m\log_e m \end{aligned}$$

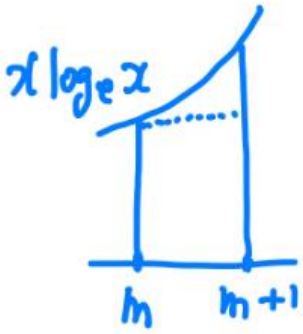
$$\begin{aligned} T_{ave}(n) &\leq cn + \frac{1}{n} \\ &= cn + \frac{2}{n} \\ T_{ave}(1) &\leq b, \\ T_{ave}(0) &\leq b. \end{aligned}$$

$$T_{ave}(n) \leq cn + \frac{2}{n}\sum_{p=0}^{n-1} T_{ave}(p) \text{ for all } n \geq 2$$



함수 $x \log_e x$가 x에 대하여 아래로 볼록인 함수이어서 $m \log_e m \leq \int_m^{m+1} x \log_e x \, dx$라는 사실을 이용하면 다음과 같은 관계식을 얻는다.

$$\begin{aligned} T_{ave}(n) &\leq cn + \frac{4b}{n} + \frac{2k}{n} \int_2^n x \log_e x \, dx \\ &\leq cn + \frac{4b}{n} + \frac{2k}{n} \{ \frac{n^2 \log_e n}{2} - \frac{n^2}{4} \} \\ &= kn \log_e n + \{ cn + \frac{4b}{n} - \frac{kn}{2} \} \end{aligned}$$



이때, $cn + \frac{4b}{n} - \frac{kn}{2} = (c - \frac{k}{2})n + \frac{4b}{n} = b(\frac{4}{n} - n)$과 같고, 이 값은 2보다 같거나 큰 n에 대해 항상 0보다 같거나 작으므로 $T_{ave}(n) \leq kn \log_e n$이 되어, 3보다 같거나 큰 임의의 n에 대해서도 두 번째 사실이 성립한다. 따라서 2보다 같거나 큰 모든 정수 n에 대해 두 번째 사실이 성립한다. □

$$= cn + \frac{1}{n}\{T_{ave}(0) + T_{ave}(1)\} +$$

$$\leq cn + \frac{4b}{n} + \frac{2k}{n} \sum_{m=2}^{n-1} m \log_e m$$

는 정수 n에 대해 $T_{ave}(n) \leq kn \log_e n$과 같은

$$\int_2^n x \log_e x dx = \frac{1}{2}x^2 \log_e x - \frac{x^2}{4}]_2^n = (\frac{n^2}{2} \log_e n - \frac{n^2}{4}) - (2 \log_e 2 - 1) \leq \frac{n^2}{2} \log_e n - \frac{n^2}{4}$$

Another Implementation

```
void quicksort(element list[], int left, int right)
/* sort list[left], . . . , list[right] into nondecreasing
order on the key field. list[left].key is arbitrarily
chosen as the pivot key.  It is assumed that
list[left].key ≤ list[right+1].key. */
{
   int pivot,i,j;
   element temp;
   if (left < right) {
      i = left;     j = right + 1;
      pivot = list[left].key;
      do {
      /* search for keys from the left and right sublists,
      swapping out-of-order elements until the left and
      right boundaries cross or meet */
         do
            i++;
         while (list[i].key < pivot);
         do
            j--;
         while (list[j].key > pivot);
         if (i < j)
            SWAP(list[i],list[j],temp);
      } while (i < j);
      SWAP(list[left],list[j],temp);
      quicksort(list,left,j-1);
      quicksort(list,j+1,right);
   }
}
```

Program 7.6: *quicksort* function

https://en.wikipedia.org/wiki/Sorting_algorithm#Comparison_of_algorithms

Comparison Sorts

Name	Best	Average	Worst	Memory	Stable	Method	Other notes
Quicksort	$n \log n$	$n \log n$	n^2	$\log n$	No	Partitioning	Quicksort is usually done in-place with $O(\log n)$ stack space.[5][6]
Merge sort	$n \log n$	$n \log n$	$n \log n$	n	Yes	Merging	Highly parallelizable (up to $O(\log n)$ using the Three Hungarians' Algorithm).[7]
Heapsort	$n \log n$	$n \log n$	$n \log n$	1	No	Selection	
Insertion sort	n	n^2	n^2	1	Yes	Insertion	$O(n + d)$, in the worst case over sequences that have d inversions.
Selection sort	n^2	n^2	n^2	1	No	Selection	Stable with $O(n)$ extra space or when using linked lists.[11]
Bubble sort	n	n^2	n^2	1	Yes	Exchanging	Tiny code size.

Insertion Sort: Example 1

[0]:	**15**	10	3	1	14	19	5	11	7	16	18	4	9	2	8	0	12	17	6	13
[1]:	**10**	**15**	3	1	14	19	5	11	7	16	18	4	9	2	8	0	12	17	6	13
[2]:	**3**	**10**	**15**	1	14	19	5	11	7	16	18	4	9	2	8	0	12	17	6	13
[3]:	**1**	**3**	**10**	**15**	14	19	5	11	7	16	18	4	9	2	8	0	12	17	6	13
[4]:	**1**	**3**	**10**	**14**	**15**	19	5	11	7	16	18	4	9	2	8	0	12	17	6	13
[5]:	**1**	**3**	**10**	**14**	**15**	**19**	5	11	7	16	18	4	9	2	8	0	12	17	6	13
[6]:	**1**	**3**	**5**	**10**	**14**	**15**	**19**	11	7	16	18	4	9	2	8	0	12	17	6	13
[7]:	**1**	**3**	**5**	**10**	**11**	**14**	**15**	**19**	7	16	18	4	9	2	8	0	12	17	6	13
[8]:	**1**	**3**	**5**	**7**	**10**	**11**	**14**	**15**	**19**	16	18	4	9	2	8	0	12	17	6	13
[9]:	**1**	**3**	**5**	**7**	**10**	**11**	**14**	**15**	**16**	**19**	18	4	9	2	8	0	12	17	6	13
[10]:	**1**	**3**	**5**	**7**	**10**	**11**	**14**	**15**	**16**	**18**	**19**	4	9	2	8	0	12	17	6	13
[11]:	**1**	**3**	**4**	**5**	**7**	**10**	**11**	**14**	**15**	**16**	**18**	**19**	9	2	8	0	12	17	6	13
[12]:	**1**	**3**	**4**	**5**	**7**	**9**	**10**	**11**	**14**	**15**	**16**	**18**	**19**	2	8	0	12	17	6	13
[13]:	**1**	**2**	**3**	**4**	**5**	**7**	**9**	**10**	**11**	**14**	**15**	**16**	**18**	**19**	8	0	12	17	6	13
[14]:	**1**	**2**	**3**	**4**	**5**	**7**	**8**	**9**	**10**	**11**	**14**	**15**	**16**	**18**	**19**	0	12	17	6	13
[15]:	**0**	**1**	**2**	**3**	**4**	**5**	**7**	**8**	**9**	**10**	**11**	**14**	**15**	**16**	**18**	**19**	12	17	6	13
[16]:	**0**	**1**	**2**	**3**	**4**	**5**	**7**	**8**	**9**	**10**	**11**	**12**	**14**	**15**	**16**	**18**	**19**	17	6	13
[17]:	**0**	**1**	**2**	**3**	**4**	**5**	**7**	**8**	**9**	**10**	**11**	**12**	**14**	**15**	**16**	**17**	**18**	**19**	6	13
[18]:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**14**	**15**	**16**	**17**	**18**	**19**	13
[19]:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

Insertion Sort: Example 2

Insertion	$O(n+d)$, in the worst case over sequences that have d inversions.

When does the insertion sort run fast?

[0]:	**0**	1	4	3	2	7	6	5	8	11	10	9	13	12	14	17	16	15	18	19
[1]:	**0**	**1**	4	3	2	7	6	5	8	11	10	9	13	12	14	17	16	15	18	19
[2]:	**0**	**1**	**4**	3	2	7	6	5	8	11	10	9	13	12	14	17	16	15	18	19
[3]:	**0**	**1**	**3**	**4**	2	7	6	5	8	11	10	9	13	12	14	17	16	15	18	19
[4]:	**0**	**1**	**2**	**3**	**4**	7	6	5	8	11	10	9	13	12	14	17	16	15	18	19
[5]:	**0**	**1**	**2**	**3**	**4**	**7**	6	5	8	11	10	9	13	12	14	17	16	15	18	19
[6]:	**0**	**1**	**2**	**3**	**4**	**6**	**7**	5	8	11	10	9	13	12	14	17	16	15	18	19
[7]:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	8	11	10	9	13	12	14	17	16	15	18	19
[8]:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	11	10	9	13	12	14	17	16	15	18	19
[9]:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**11**	10	9	13	12	14	17	16	15	18	19
[10]:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**10**	**11**	9	13	12	14	17	16	15	18	19
[11]:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	13	12	14	17	16	15	18	19
[12]:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**13**	12	14	17	16	15	18	19
[13]:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	14	17	16	15	18	19
[14]:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	17	16	15	18	19
[15]:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**17**	16	15	18	19
[16]:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**16**	**17**	15	18	19
[17]:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	18	19
[18]:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**
[19]:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

이러한 insertion sort의 성질을 quick sort의 성능 향상에 활용하자.

Insertion Sort: Implementation

```
void insertion_sort(int *A, int n) {
  int i, j, tmp;

  for (i = 1; i < n; i++) {
    tmp = A[i];
    j = i;
    while ((j > 0) && (tmp < A[j - 1])) {
      A[j] = A[j - 1];
      j--;
    }
    A[j] = tmp;
  }
}
```

[0]:	**15**	10	3	1	14	19	5	11
[1]:	**10**	**15**	3	1	14	19	5	11
[2]:	**3**	**10**	**15**	1	14	19	5	11
[3]:	**1**	**3**	**10**	**15**	14	19	5	11
[4]:	**1**	**3**	**10**	**14**	**15**	19	5	11
[5]:	**1**	**3**	**10**	**14**	**15**	**19**	5	11
[6]:	**1**	**3**	**5**	**10**	**14**	**15**	**19**	11

Sort a list of elements by iteratively inserting a next element in a progressively growing sorted array.

$$T(n) = O(n^2)$$

Insertion Sort: Run-Time Analysis

- **Worst case**
  - No. of comparisons:

$$1+2+\cdots+n-1=O(\frac{n^2}{2})$$

  - No. of record assignments:

$$1+2+\cdots+n-1=O(\frac{n^2}{2})$$

```
[ 0]:  15  10   3   1  14  19   5  11
[ 1]:  10  15   3   1  14  19   5  11
[ 2]:   3  10  15   1  14  19   5  11
[ 3]:   1   3  10  15  14  19   5  11
[ 4]:   1   3  10  14  15  19   5  11
[ 5]:   1   3  10  14  15  19   5  11
[ 6]:   1   3   5  10  14  15  19  11
```

```
for (i = 1; i < n; i
  tmp = A[i];
  j = i;
  while ((j > 0) &&
    A[j] = A[j - 1];
    j--;
  }
  A[j] = tmp;
}
```

- **Average case**
  - No. of comparisons:

$$\sum_{i=1}^{n-1}\frac{1+2+\cdots+i+i}{i+1}=\sum_{i=1}^{n-1}(\frac{i}{2}+1-\frac{1}{i+1})\approx\frac{(n-1)(n+4)}{4}-\ln n=O(\frac{n^2}{4})$$

  - No. of record assignments

$$\sum_{i=1}^{n-1}(\frac{0+1+2+\cdots+i}{i+1}+2)$$

$$=\frac{n(n-1)}{4}+2(n-1)=O(\frac{n^2}{4})$$