Sorting (P.2)

used some pictures and codes from
http://people.cs.vt.edu/shaffer/Book/C++3elatest.pdf
Data Structures and Algorithm Analysis
by Clifford A. Schaffer

1 2 3 4 5 6 7 8

59 20 17 13 28 14 23 83 36 98 11 70 65 41 42 15

36 20 17 13 28 14 23 83 36 59 11 70 65 41 42 15

36 11 17 13 28 14 23 83 36 59 20 70 65 41 42 15

36 11 17 13 28 14 23 83 36 59 20 70 65 41 42 15

36 11 17 13 28 14 23 83 36 59 20 70 65 41 42 15

36 11 17 13 28 14 23 83 36 59 20 70 65 41 42 15

36 11 17 13 28 14 23 83 36 59 20 70 65 41 42 15

36 11 17 13 28 14 15 83 36 59 20 70 65 41 42 23

36 20 11 13 28 14 23 15 59 98 17 70 65 41 42 83

Stride : 8 (2개 짜리 리스트) × 8 각각 sorting 하는 과정

59 20 17 13 28 14 23 83 36 98 11 70 65 41 42 15

{59, 36} 59 20 17 13 28 14 23 83 36 98 11 70 65 41 42 15

{20, 98} 36 20 17 13 28 14 23 83 59 98 11 70 65 41 42 15

{17, 11} 36 20 17 13 28 14 23 83 59 98 11 70 65 41 42 15

{13, 70} 36 20 11 13 28 14 23 83 59 98 17 70 65 41 42 15

{28, 65} 36 20 11 13 28 14 23 83 59 98 17 70 65 41 42 15

{14, 41} 36 20 11 13 28 14 23 83 59 98 17 70 65 41 42 15

{23, 42} 36 20 11 13 28 14 23 83 59 98 17 70 65 41 42 15

{83, 15} 36 20 11 13 28 14 23 83 59 98 17 70 65 41 42 15

36 20 11 13 28 14 23 15 59 98 17 70 65 41 42 83

36 20 11 13 28 14 23 15 59 98 17 70 65 41 42 83

Stride : 4   4 element sublist X 4개

① 36 20 11 13 28 14 23 15 59 98 17 70 65 41 42 83

{36, 28, 59, 65}

② 28 20 11 13 36 14 23 15 59 98 17 70 65 41 42 83

{ 20, 14, 98, 41 }

③ 28 14 11 13 36 20 23 15 59 41 17 70 65 98 42 83

{ 11, 23, 17, 42 }

④ 28 14 11 13 36 20 17 15 59 41 23 70 65 98 42 83

{ 13, 15, 70, 83 }

28 14 11 13 36 20 17 15 59 41 23 70 65 98 42 83

Stride : 2 (8 element sublist) X 2

① 28 14 11 13 36 20 17 15 59 41 23 70 65 98 42 83

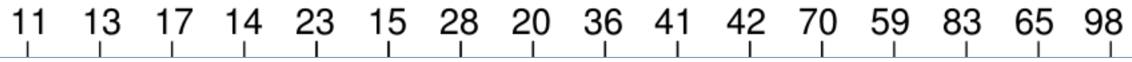
{28, 11, 36, 17, 59, 23, 65, 42}

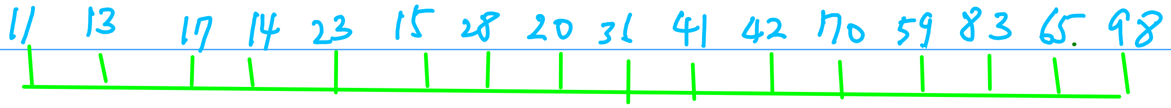
② 11 14 17 13 23 20 28 15 36 41 42 70 59 98 65 83

{14, 13, 20, 15, 41, 70, 98, 83}

11 13 17 14 23 15 28 20 36 41 42 70 59 83 65 98

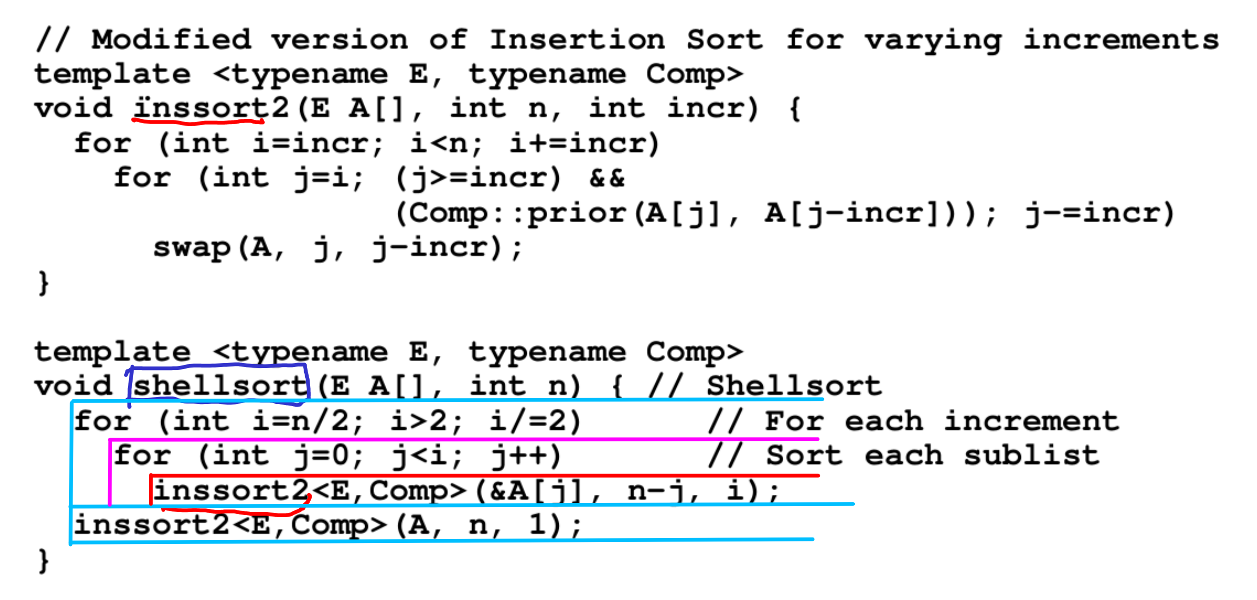
11 13 17 14 23 15 28 20 36 41 42 70 59 83 65 98





{11, 13, 17, 14, 23, 15, 28, 20, 36, 41, 42, 70, 59, 83, 65, 98}

11 13 14 15 17 20 23 28 36 41 42 59 65 70 83 98



```
// Modified version of Insertion Sort for varying increments
template <typename E, typename Comp>
void inssort2(E A[], int n, int incr) {
  for (int i=incr; i<n; i+=incr)
    for (int j=i; (j>=incr) &&
                  (Comp::prior(A[j], A[j-incr])); j-=incr)
      swap(A, j, j-incr);
}

template <typename E, typename Comp>
void shellsort(E A[], int n) { // Shellsort
  for (int i=n/2; i>2; i/=2)      // For each increment
    for (int j=0; j<i; j++)       // Sort each sublist
      inssort2<E,Comp>(&A[j], n-j, i);
  inssort2<E,Comp>(A, n, 1);
}
```

for (i= n/2; i > 2; i /= 2)

16 → 16 > 2 → 8 . Stride=8 { 2 elements } sub list 8개

8 > 2 → 4 . Stride = 4 { 4 elements } sub list 4개

4 > 2 → 2 . Stride = 2 { 8 elements } sub list 2개

2 > 2 ↘

i = i + 3 i += 3 i = i + 3;

i = i − 3 i −= 3 i = i − 3;

i = i * 3 i *= 3 i = i * 3;

i = i / 3 i /= 3 i = i / 3;

&A[j] : subarry starting from A[j] with the length of (n-j)
this subarray is accessed with the stride of i increment

for loop condition (i>2) ==> (i>1)

```
template <typename E, typename Comp>
void inssort(E A[], int n) { // Insertion Sort
  for (int i=1; i<n; i++)         // Insert i'th record
    for (int j=i; (j>0) && (Comp::prior(A[j], A[j-1])); j--)
      swap(A, j, j-1);
}
```

```
// Modified version of Insertion Sort for varying increments
template <typename E, typename Comp>
void inssort2(E A[], int n, int incr) {
  for (int i=incr; i<n; i+=incr)
    for (int j=i; (j>=incr) &&
                  (Comp::prior(A[j], A[j-incr])); j-=incr)
      swap(A, j, j-incr);
}
```

incr

Stride : 4   4 element sublist X 4개

i=0   i=4   i=8   i=12

j=4   j=8   j=12

j=4   j=8

j=4

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

① 36 20 11 13 28 14 23 15 59 98 17 70 65 41 42 83

{36, 28, 59, 65}

28   36   59   65

28   36   59   65

② 28 20 11 13 36 14 23 15 59 98 17 70 65 41 42 83

{20, 14, 98, 41}

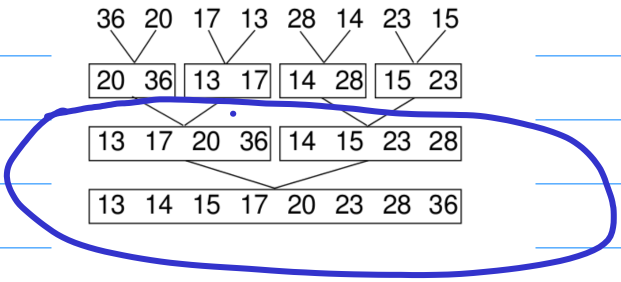
20   14   98   41

14 20 98 41

14 20 98 41

14 20 41 98

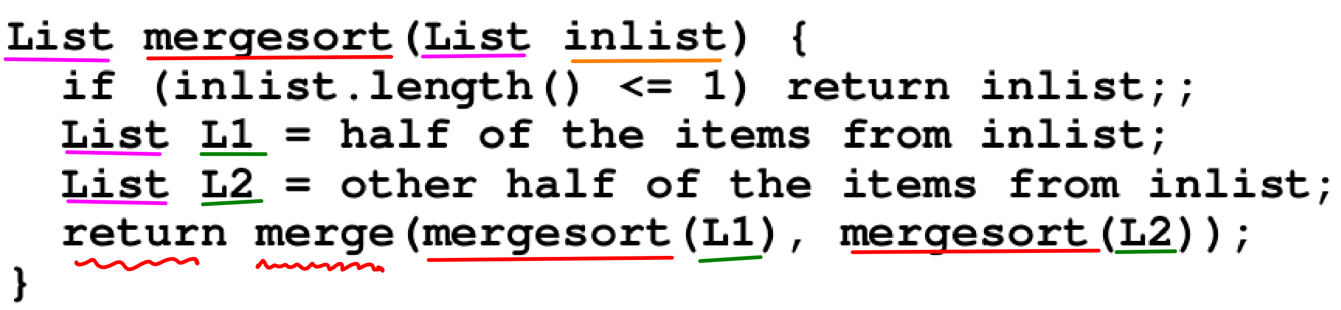
Merge Sort



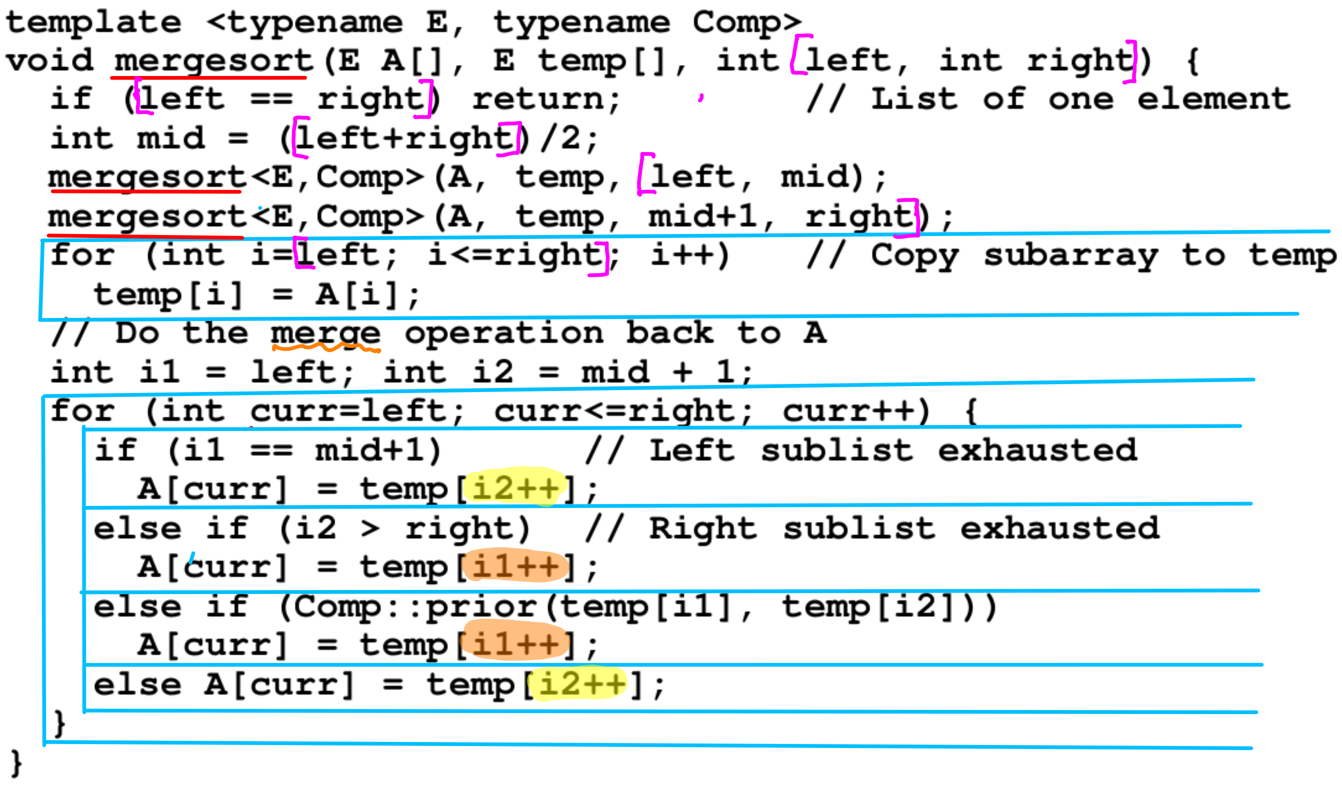
13 17 20 36 . 17 20 36 . 17 20 36
14 15 23 28 14 15 23 28 15 23 28

17 20 36 . 20 36 36 36
23 28 23 28 23 28 28

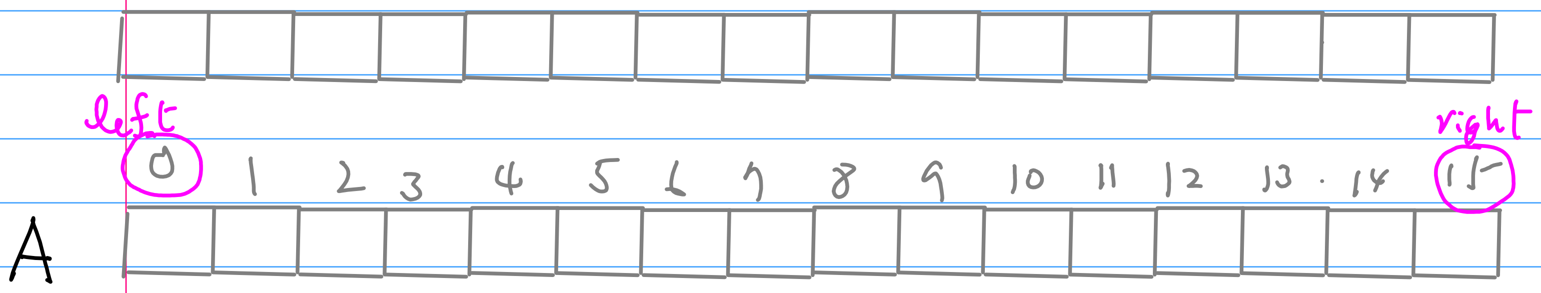
13 14 15 17 20 23 28 36



```
List mergesort(List inlist) {
  if (inlist.length() <= 1) return inlist;;
  List L1 = half of the items from inlist;
  List L2 = other half of the items from inlist;
  return merge(mergesort(L1), mergesort(L2));
}
```

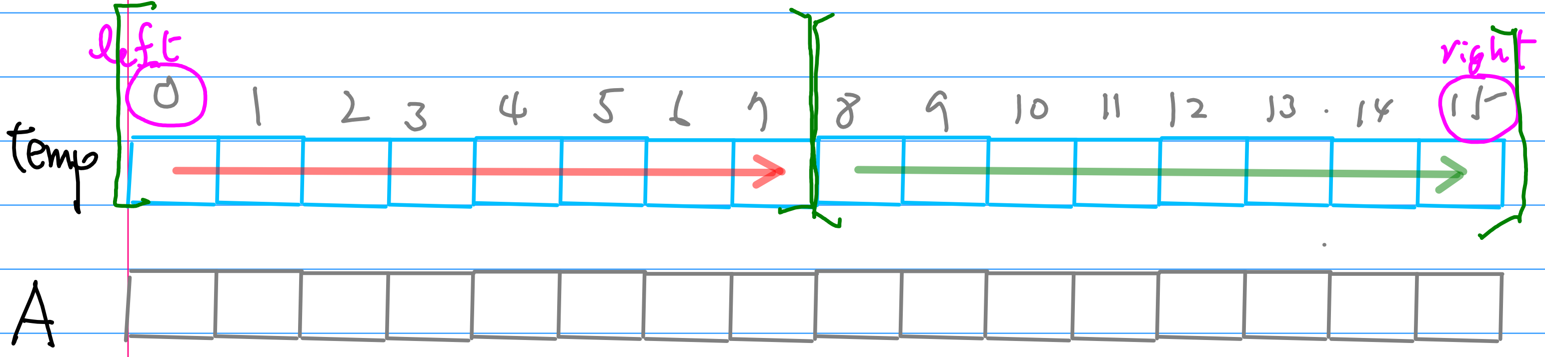


```
template <typename E, typename Comp>
void mergesort(E A[], E temp[], int left, int right) {
  if (left == right) return;          // List of one element
  int mid = (left+right)/2;
  mergesort<E,Comp>(A, temp, left, mid);
  mergesort<E,Comp>(A, temp, mid+1, right);
  for (int i=left; i<=right; i++)    // Copy subarray to temp
    temp[i] = A[i];
  // Do the merge operation back to A
  int i1 = left; int i2 = mid + 1;
  for (int curr=left; curr<=right; curr++) {
    if (i1 == mid+1)        // Left sublist exhausted
      A[curr] = temp[i2++];
    else if (i2 > right)    // Right sublist exhausted
      A[curr] = temp[i1++];
    else if (Comp::prior(temp[i1], temp[i2]))
      A[curr] = temp[i1++];
    else A[curr] = temp[i2++];
  }
}
```

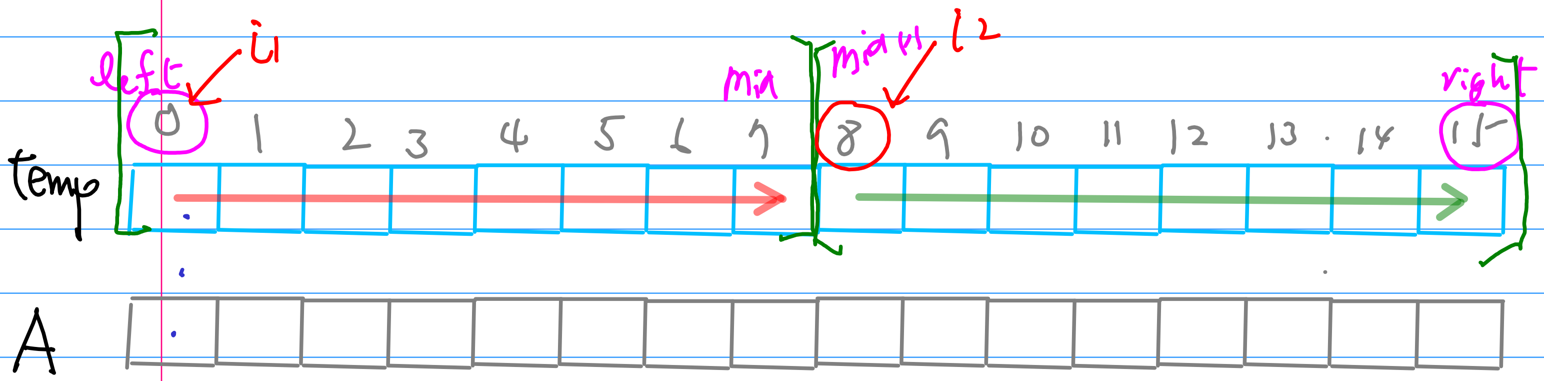


mid= (0+15)/2 = 7   m+1=8

mergesort (A, temp, [0, 7])
mergesort (A, temp, [8, 15])



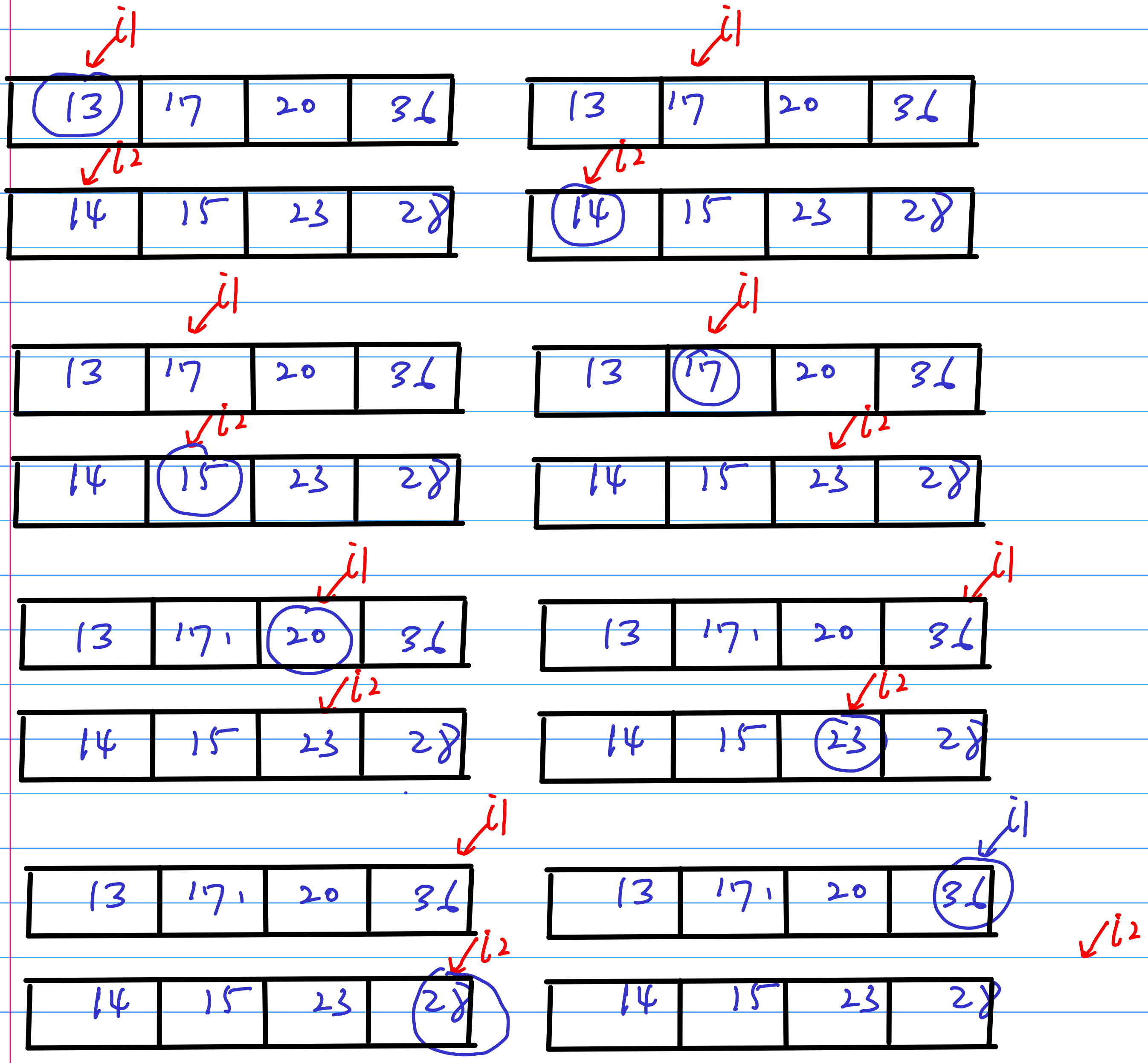
```
  mergesort<E,Comp>(A, temp, mid+1, right);
  for (int i=left; i<=right; i++)    // Copy subarray to temp
    temp[i] = A[i];
  // Do the merge operation back to A
  int i1 = left; int i2 = mid + 1;
  for (int curr=left; curr<=right; curr++) {
    if (i1 == mid+1)       // Left sublist exhausted
      A[curr] = temp[i2++];
    else if (i2 > right)   // Right sublist exhausted
      A[curr] = temp[i1++];
    else if (Comp::prior(temp[i1], temp[i2]))
      A[curr] = temp[i1++];
    else A[curr] = temp[i2++];
  }
}
```

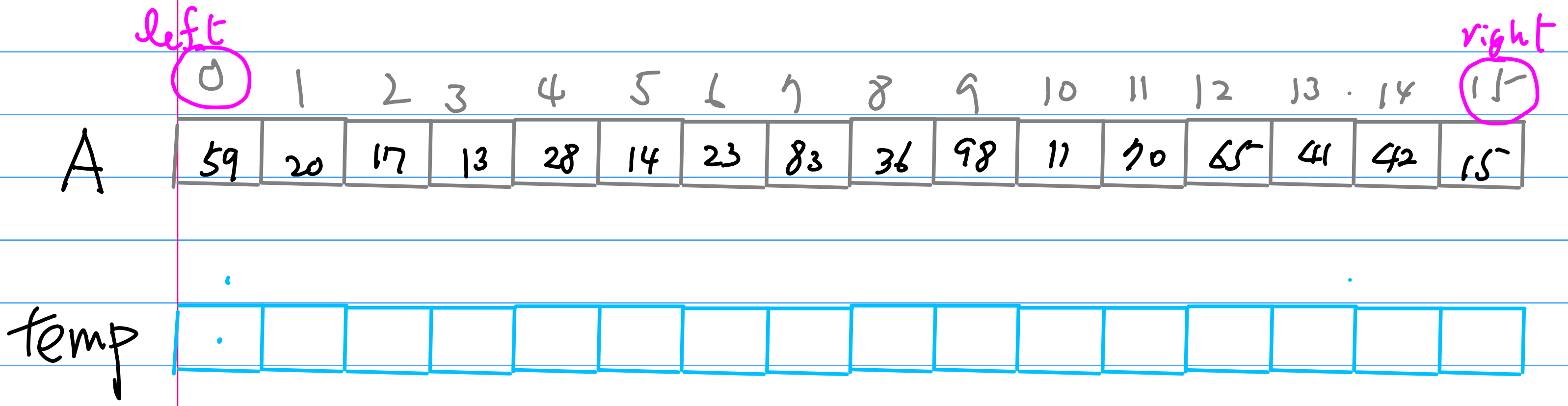


13 17 20 36 . 17 20 36 . 17 20 36
14 15 23 28 . 14 15 23 28 . 15 23 28

17 20 36 . 20 36 . 36 . 36
23 28 . 23 28 . 23 28 . 28

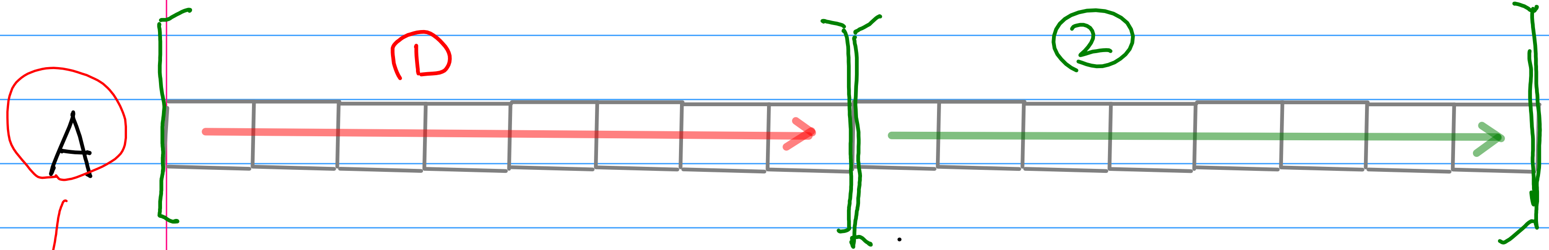
13 14 15 17 20 23 28 36

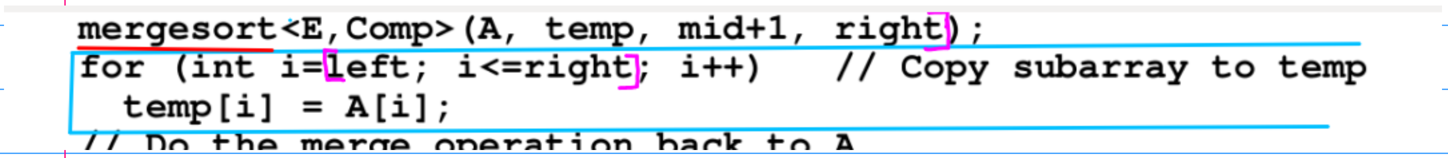




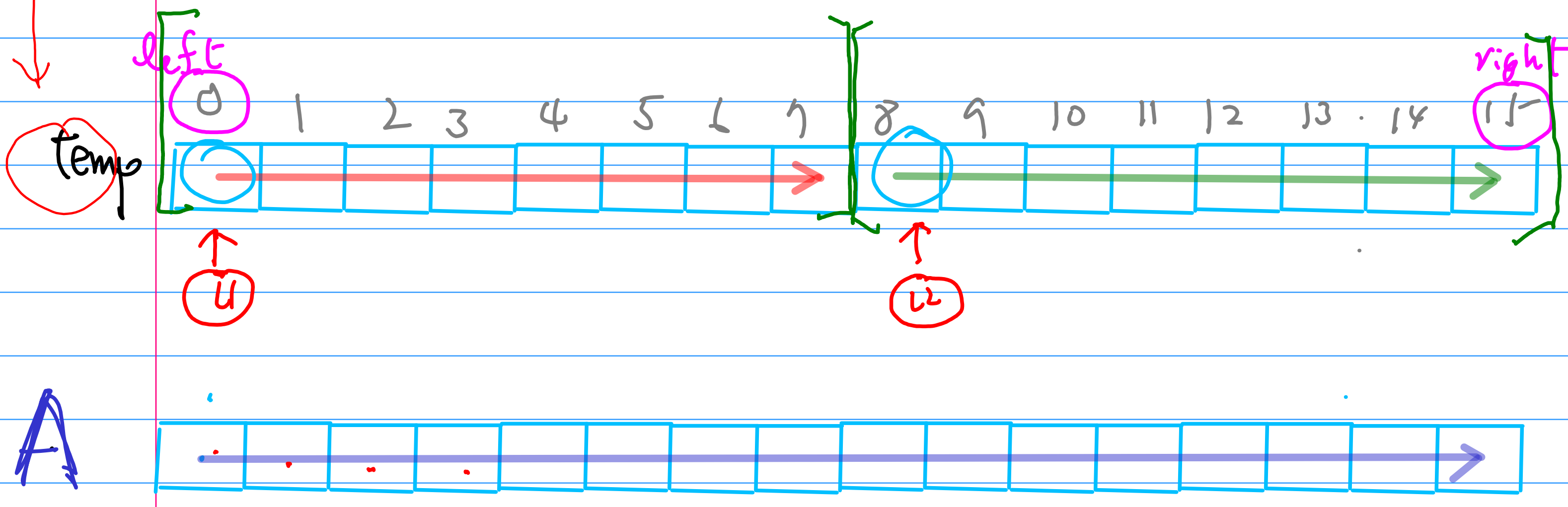
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	59	20	17	13	28	14	23	83	36	98	11	70	65	41	42	15
temp																

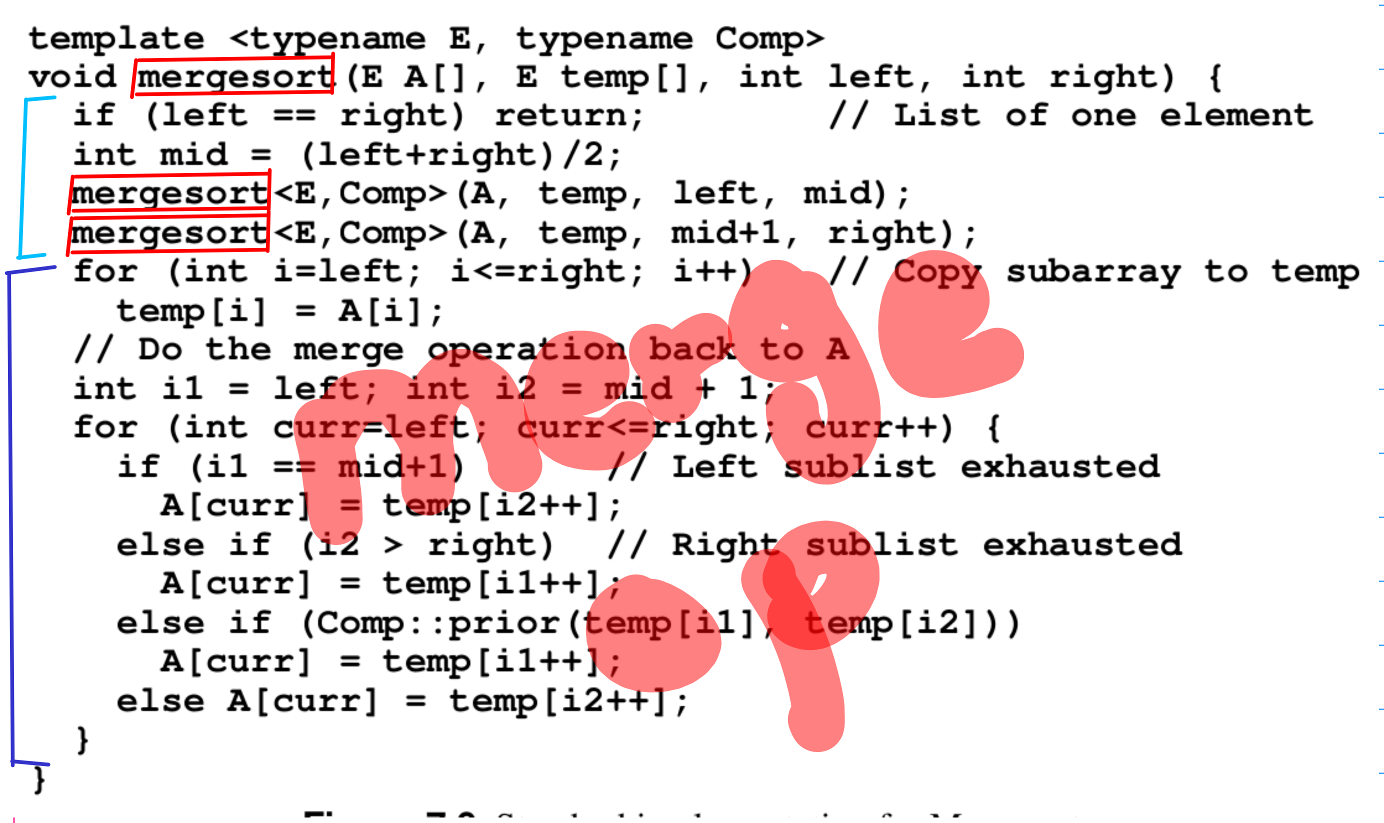
① mergesort (A, temp, [0, 7])
② mergesort (A, temp, [8, 15])





```
mergesort<E,Comp>(A, temp, mid+1, right);
for (int i=left; i<=right; i++)     // Copy subarray to temp
  temp[i] = A[i];
// Do the merge operation back to A
```



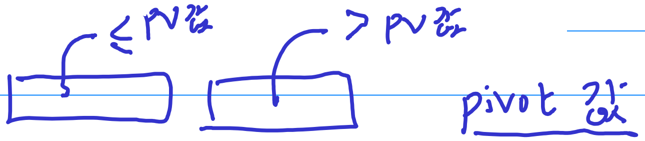


```
template <typename E, typename Comp>
void mergesort(E A[], E temp[], int left, int right) {
  if (left == right) return;          // List of one element
  int mid = (left+right)/2;
  mergesort<E,Comp>(A, temp, left, mid);
  mergesort<E,Comp>(A, temp, mid+1, right);
  for (int i=left; i<=right; i++)   // Copy subarray to temp
    temp[i] = A[i];
  // Do the merge operation back to A
  int i1 = left; int i2 = mid + 1;
  for (int curr=left; curr<=right; curr++) {
    if (i1 == mid+1)      // Left sublist exhausted
      A[curr] = temp[i2++];
    else if (i2 > right)  // Right sublist exhausted
      A[curr] = temp[i1++];
    else if (Comp::prior(temp[i1], temp[i2]))
      A[curr] = temp[i1++];
    else A[curr] = temp[i2++];
  }
}
```

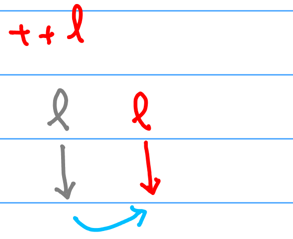
merge
op

```
template <typename E, typename Comp>
void qsort(E A[], int i, int j) { // Quicksort
  if (j <= i) return; // Don't sort 0 or 1 element
  int pivotindex = findpivot(A, i, j);
  swap(A, pivotindex, j);     // Put pivot at end
  // k will be the first position in the right subarray
  int k = partition<E,Comp>(A, i-1, j, A[j]);
  swap(A, k, j);              // Put pivot in place
  qsort<E,Comp>(A, i, k-1);
  qsort<E,Comp>(A, k+1, j);
}
```

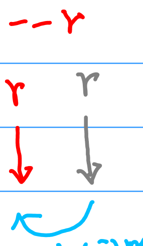
```
template <typename E>
inline int findpivot(E A[], int i, int j)
  { return (i+j)/2; }
```



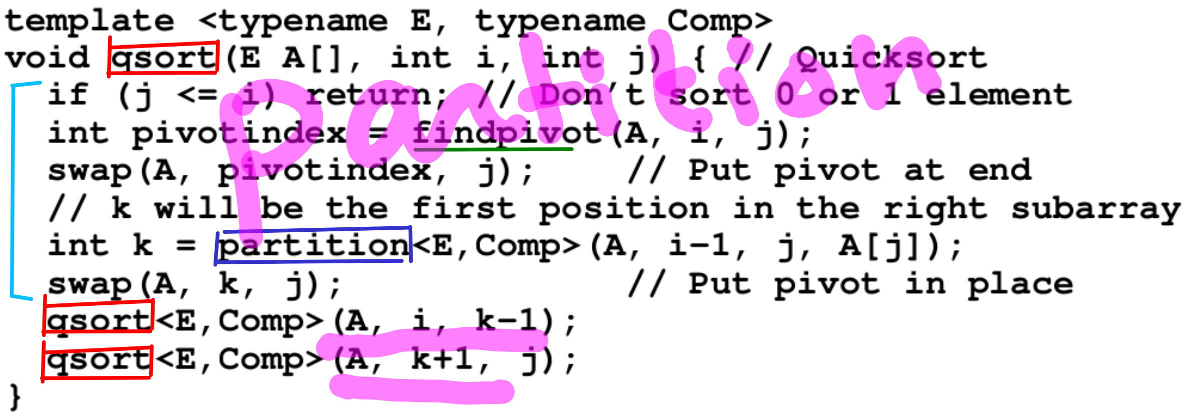
```
template <typename E, typename Comp>
inline int partition(E A[], int l, int r, E& pivot) {
  do {                    // Move the bounds inward until they meet
    while (Comp::prior(A[++l], pivot));    // Move l right and
    while ((l < r) && Comp::prior(pivot, A[--r])); // r left
    swap(A, l, r);                  // Swap out-of-place values
  } while (l < r);                  // Stop when they cross
  return l;        // Return first position in right partition
}
```



1 증가 하면 오른쪽으로 한칸

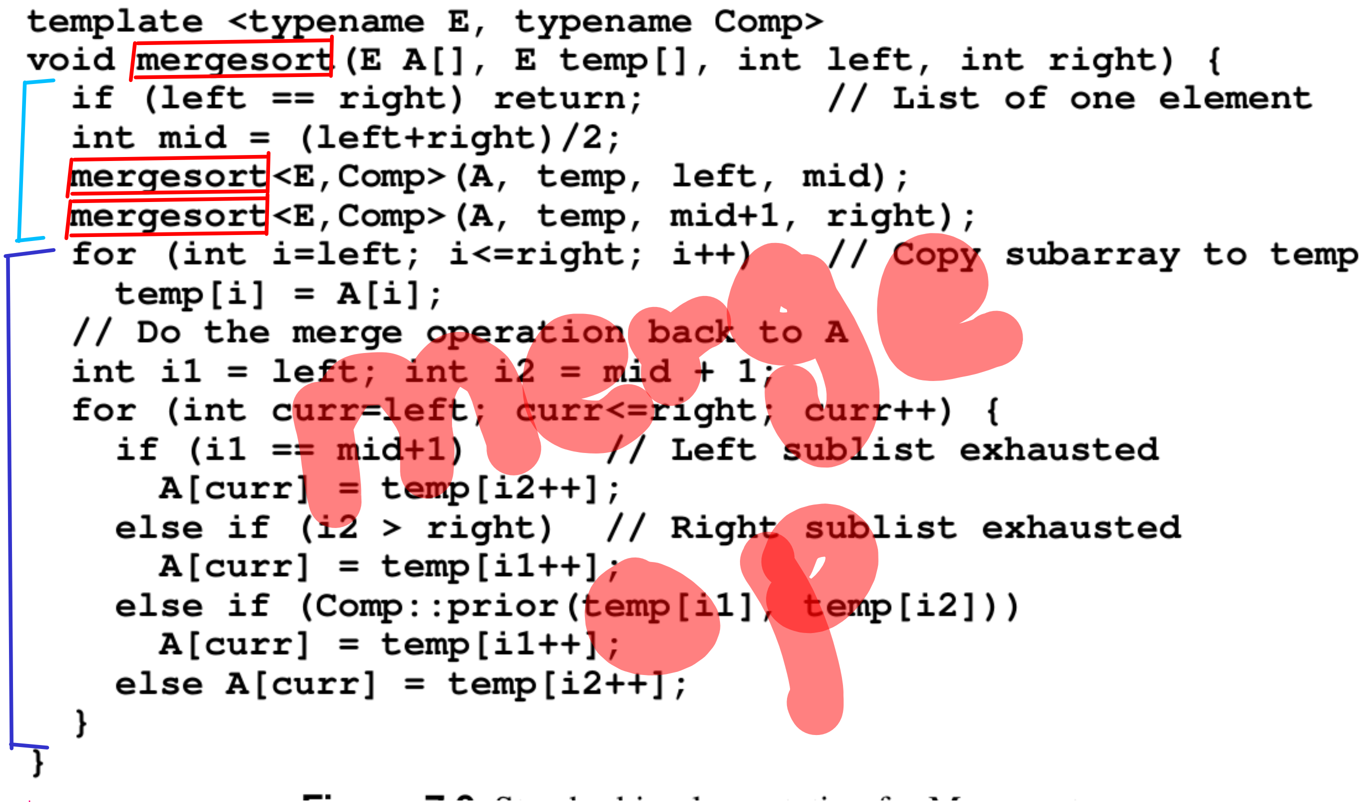


1 감소하면 왼쪽으로 한칸 이동



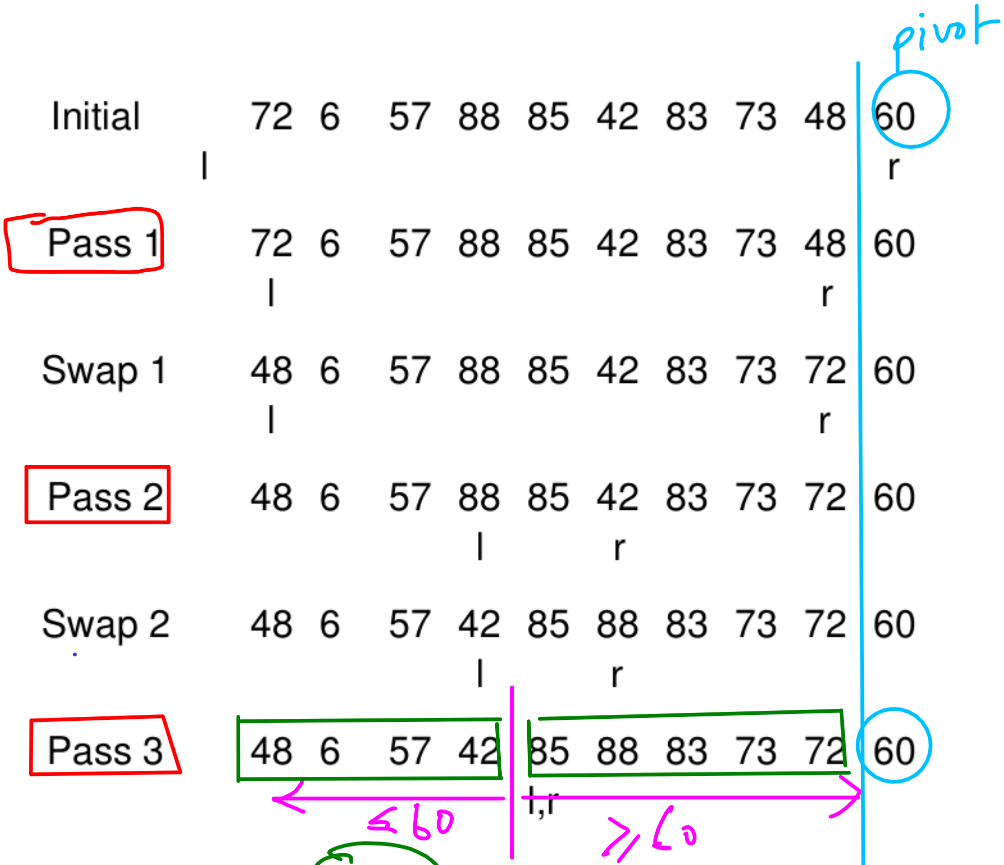
```
template <typename E, typename Comp>
void qsort(E A[], int i, int j) { // Quicksort
  if (j <= i) return; // Don't sort 0 or 1 element
  int pivotindex = findpivot(A, i, j);
  swap(A, pivotindex, j);     // Put pivot at end
  // k will be the first position in the right subarray
  int k = partition<E,Comp>(A, i-1, j, A[j]);
  swap(A, k, j);              // Put pivot in place
  qsort<E,Comp>(A, i, k-1);
  qsort<E,Comp>(A, k+1, j);
}
```

partition



```
template <typename E, typename Comp>
void mergesort(E A[], E temp[], int left, int right) {
  if (left == right) return;        // List of one element
  int mid = (left+right)/2;
  mergesort<E,Comp>(A, temp, left, mid);
  mergesort<E,Comp>(A, temp, mid+1, right);
  for (int i=left; i<=right; i++)   // Copy subarray to temp
    temp[i] = A[i];
  // Do the merge operation back to A
  int i1 = left; int i2 = mid + 1;
  for (int curr=left; curr<=right; curr++) {
    if (i1 == mid+1)     // Left sublist exhausted
      A[curr] = temp[i2++];
    else if (i2 > right)  // Right sublist exhausted
      A[curr] = temp[i1++];
    else if (Comp::prior(temp[i1], temp[i2]))
      A[curr] = temp[i1++];
    else A[curr] = temp[i2++];
  }
}
```

merge
op



```
Initial   72  6   57  88  85  42  83  73  48  60
        l                                     r
Pass 1    72  6   57  88  85  42  83  73  48  60
          l                               r
Swap 1    48  6   57  88  85  42  83  73  72  60
          l                               r
Pass 2    48  6   57  88  85  42  83  73  72  60
                      l       r
Swap 2    48  6   57  42  85  88  83  73  72  60
                      l       r
Pass 3    48  6   57  42  85  88  83  73  72  60
                          l,r
```

Figure 7.13 The Quicksort partition step. The first row shows the initial positions for a collection of ten key values. The pivot value is 60, which has been swapped to the end of the array. The **do** loop makes three iterations, each time moving counters **l** and **r** inwards until they meet in the third pass. In the end, the left partition contains four values and the right partition contains six values. Function **qsort** will place the pivot value into position 4.

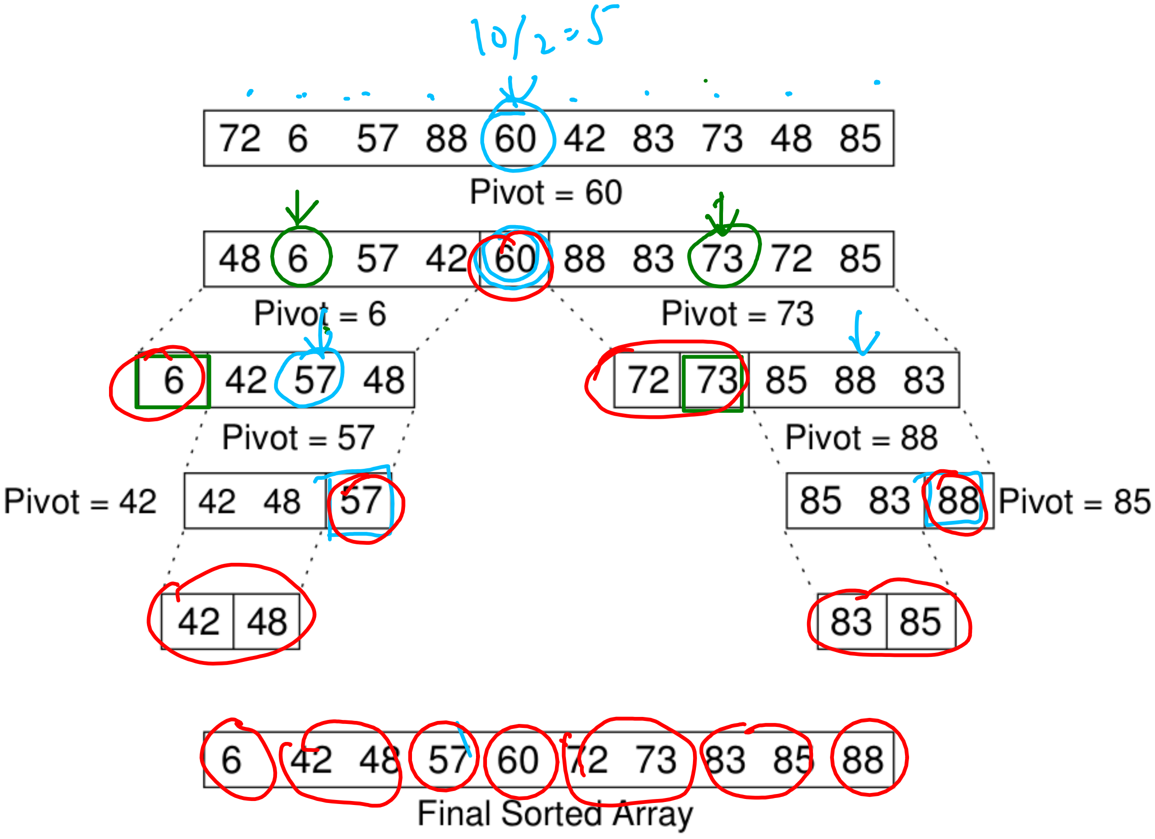


Figure 7.14 An illustration of Quicksort.

$$\frac{0+9}{2} = 4$$

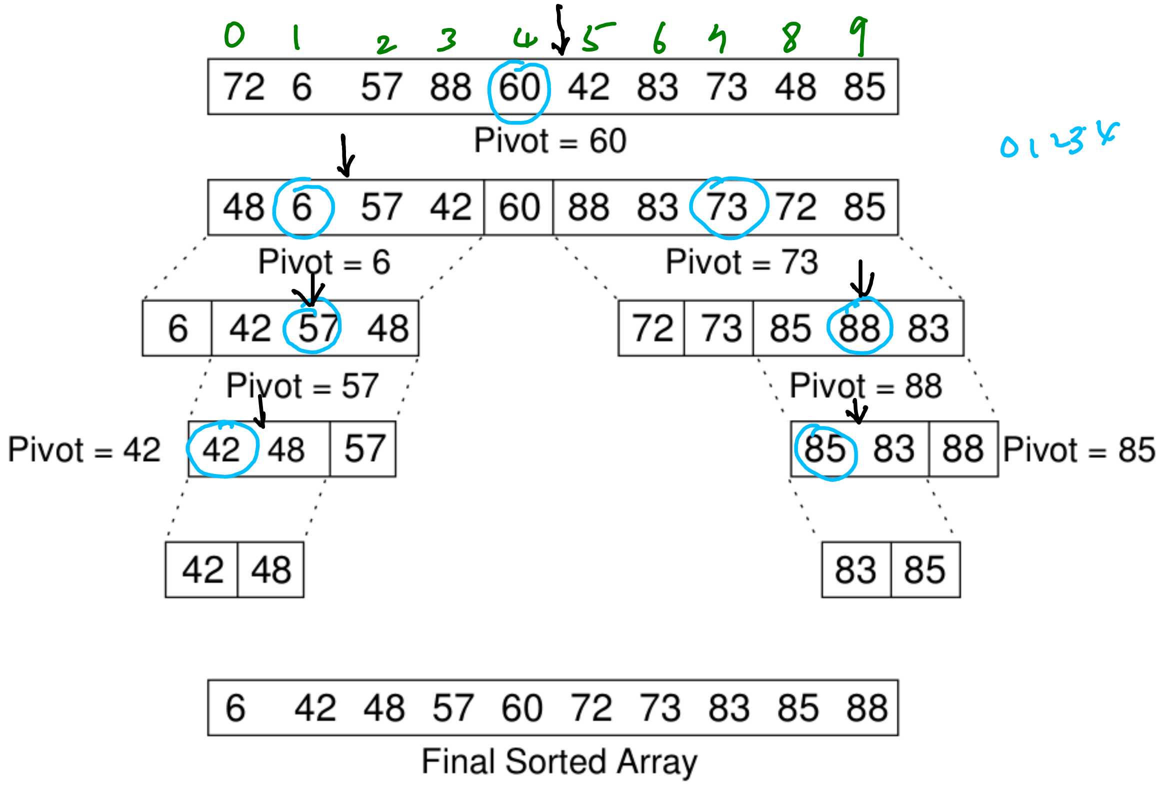


Figure 7.14 An illustration of Quicksort.

```
template <typename E>
inline int findpivot(E A[], int i, int j)
  { return (i+j)/2; }
```

$\boxed{0}$ 1 $(0+1)/2 = 0$

0 $\boxed{1}$ 2 $(0+2)/2 = 1$

0 $\boxed{1}$ 2 3 $(0+3)/2 = 1$

0 1 $\boxed{2}$ 3 4 $(0+4)/2 = 2$

0 1 $\boxed{2}$ 3 4 5 $(0+5)/2 = 2$

0 1 2 $\boxed{3}$ 4 5 6 $(0+6)/2 = 3$

$\textcircled{0}$ 1 — $(0+1)/2 = 0$

0 $\textcircled{1}$ 2 — $(0+2)/2 = 1$

0 $\textcircled{1}$ 2 3 — $(0+3)/2 = 1$

0 1 $\textcircled{2}$ 3 4 — $(0+4)/2 = 2$

0 1 $\textcircled{2}$ 3 4 5 — $(0+5)/2 = 2$

0 1 2 $\textcircled{3}$ 4 5 6 — $(0+6)/2 = 3$

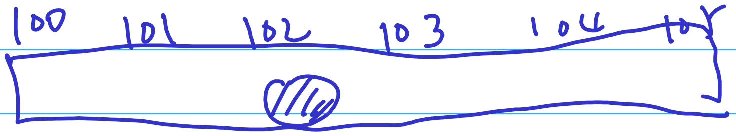
$\textcircled{3}$ 4 — $(3+4)/2 = 3$

3 $\textcircled{4}$ 5 — $(3+5)/2 = 4$

3 $\textcircled{4}$ 5 6 — $(3+6)/2 = 4$

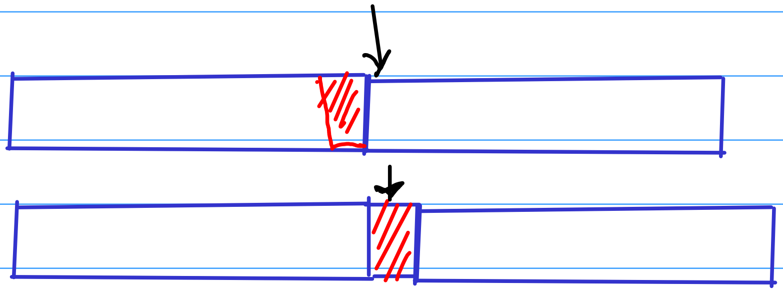
3 4 $\textcircled{5}$ 6 7 — $(3+7)/2 = 5$

3 4 $\textcircled{5}$ 6 7 8 — $(3+8)/2 = 5$



$$\frac{100+105}{2}$$

$$\begin{array}{r}102.\\ 2\overline{)205}\end{array}$$

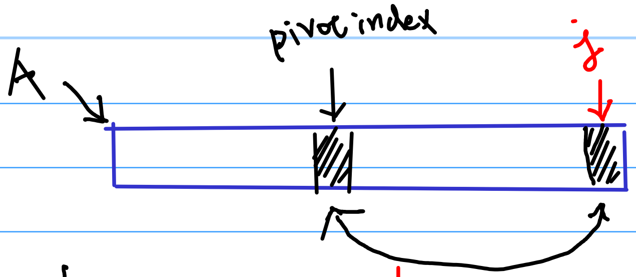


Even # of elements

①

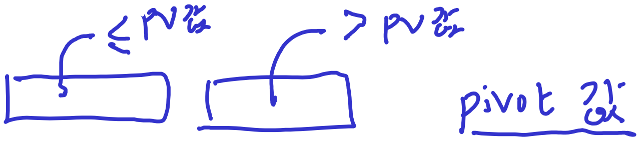
```
template <typename E, typename Comp>
void qsort(E A[], int i, int j) { // Quicksort
  if (j <= i) return; // Don’t sort 0 or 1 element
  int pivotindex = findpivot(A, i, j);
  swap(A, pivotindex, j);      // Put pivot at end
  // k will be the first position in the right subarray
  int k = partition<E,Comp>(A, i-1, j, A[j]);
  swap(A, k, j);               // Put pivot in place
  qsort<E,Comp>(A, i, k-1);
  qsort<E,Comp>(A, k+1, j);
}
```

j는 rightmost

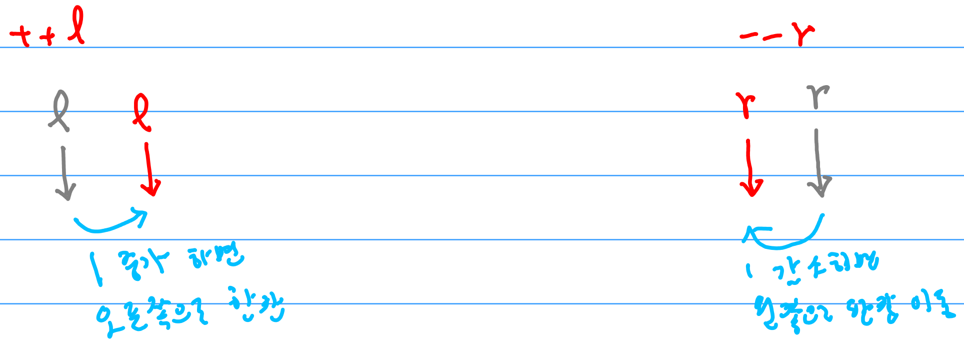


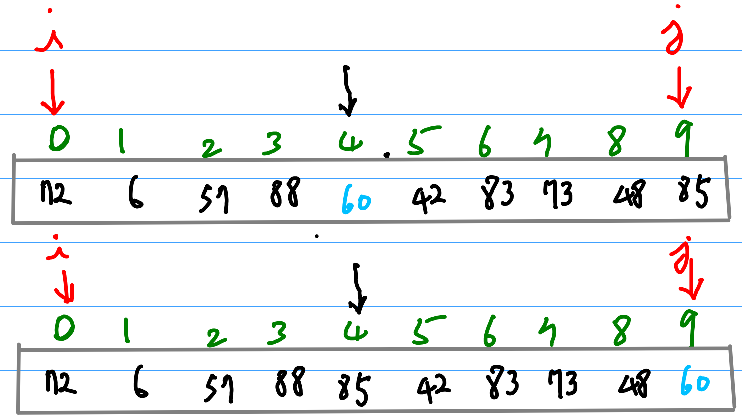
0	1	2	3	4	5	6	7	8	9
72	6	57	88	60	42	83	73	48	85

0	1	2	3	4	5	6	7	8	9
72	6	57	88	85	42	83	73	48	60

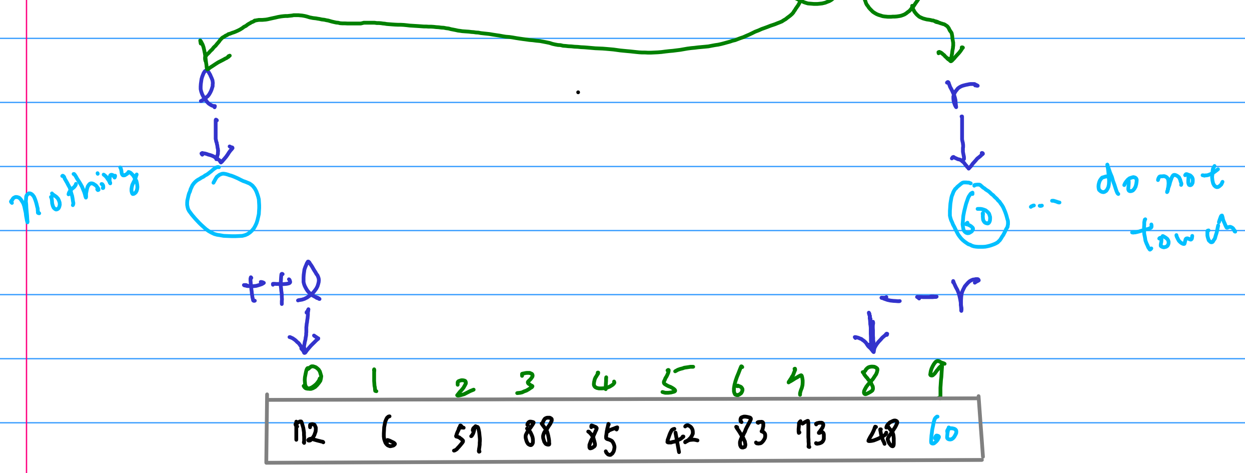


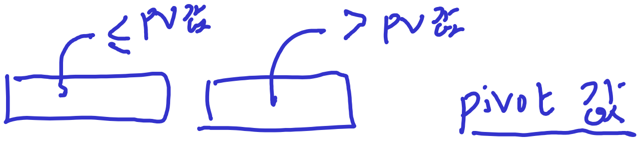
```
template <typename E, typename Comp>
inline int partition(E A[], int l, int r, E& pivot) {
  do {                // Move the bounds inward until they meet
    while (Comp::prior(A[++l], pivot));   // Move l right and
    while ((l < r) && Comp::prior(pivot, A[--r])); // r left
    swap(A, l, r);                      // Swap out-of-place values
  } while (l < r);                      // Stop when they cross
  return l;          // Return first position in right partition
}
```





```
int k = partition<E,Comp>(A, i-1, j, A[j]);
```





```
template <typename E, typename Comp>
inline int partition(E A[], int l, int r, E& pivot) {
  do {                    // Move the bounds inward until they meet
    while (Comp::prior(A[++l], pivot));  // Move l right and
    while ((l < r) && Comp::prior(pivot, A[--r]));  // r left
    swap(A, l, r);                // Swap out-of-place values
  } while (l < r);                // Stop when they cross
  return l;          // Return first position in right partition
}
```

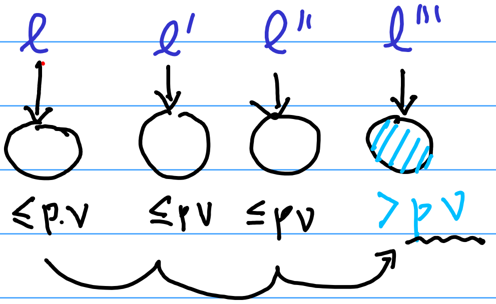
```
while (A[++l] <= pivot) ;

while ((l < r) && (pivot <= A[--r])) ;
```

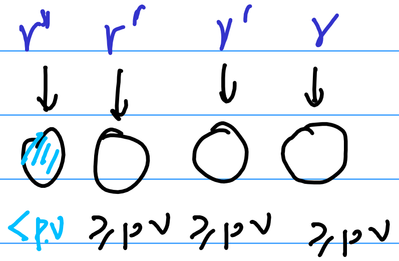
```
++l;
while (A[l] <= pivot) { ++l; };
```

→ while을 빠져나올 때는 A[l] > pivot

```
--r;
while ((l < r) && (pivot <= A[r])) { --r ; }
```

→ while 빠져나올 때는 A[r] < pivot





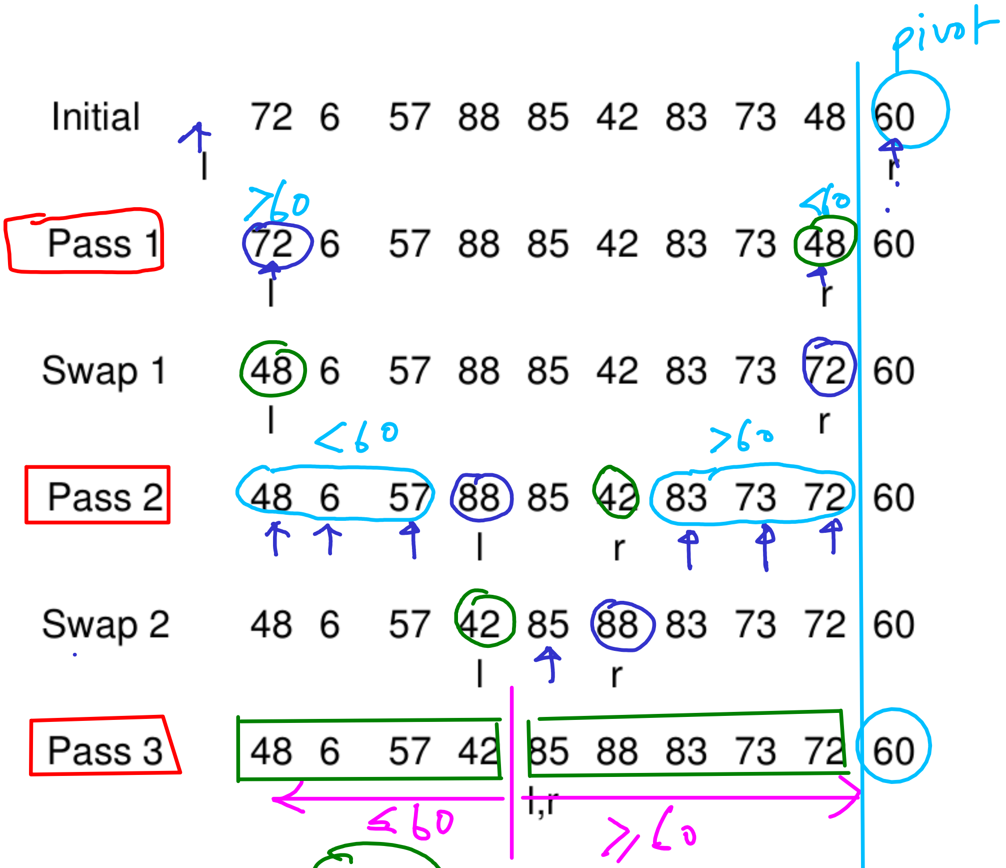
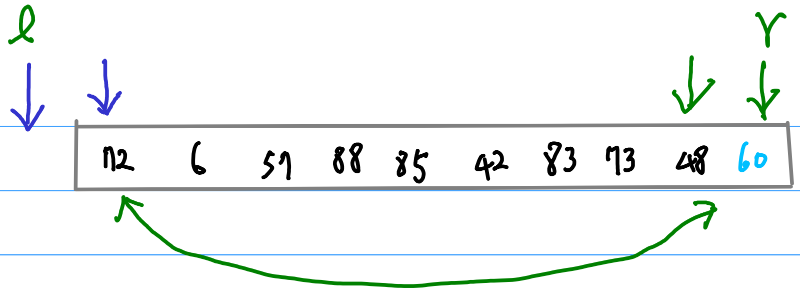
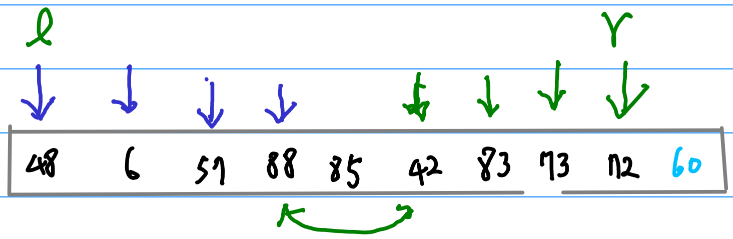

pivot
Initial 72 6 57 88 85 42 83 73 48 60
l r
>60 <60
Pass 1 72 6 57 88 85 42 83 73 48 60
l r
Swap 1 48 6 57 88 85 42 83 73 72 60
l r
<60 >60
Pass 2 48 6 57 88 85 42 83 73 72 60
l r
Swap 2 48 6 57 42 85 88 83 73 72 60
l r
Pass 3 48 6 57 42 85 88 83 73 72 60
l,r
≤60 ≥60

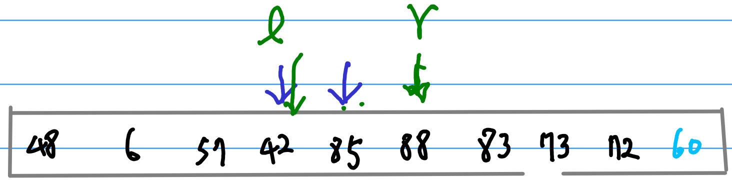
Figure 7.13 The Quicksort partition step. The first row shows the initial positions for a collection of ten key values. The pivot value is 60, which has been swapped to the end of the array. The **do** loop makes three iterations, each time moving counters **l** and **r** inwards until they meet in the third pass. In the end, the left partition contains four values and the right partition contains six values. Function **qsort** will place the pivot value into position 4.



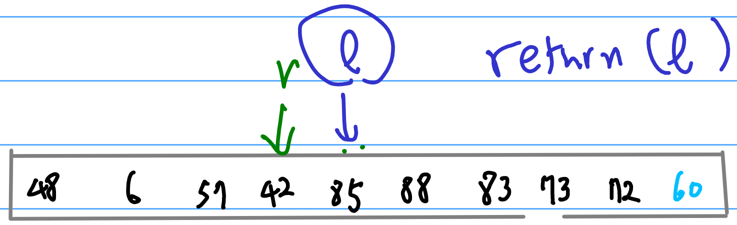
```
while (A[++l] <= pivot) ;

while ((l < r) && (pivot <= A[--r])) ;
```



```
while (A[++l] <= pivot) ;

while ((l < r) && (pivot <= A[--r])) ;
```

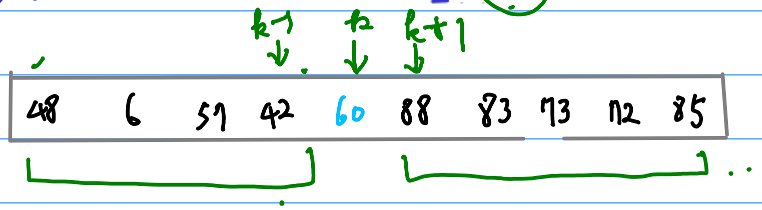


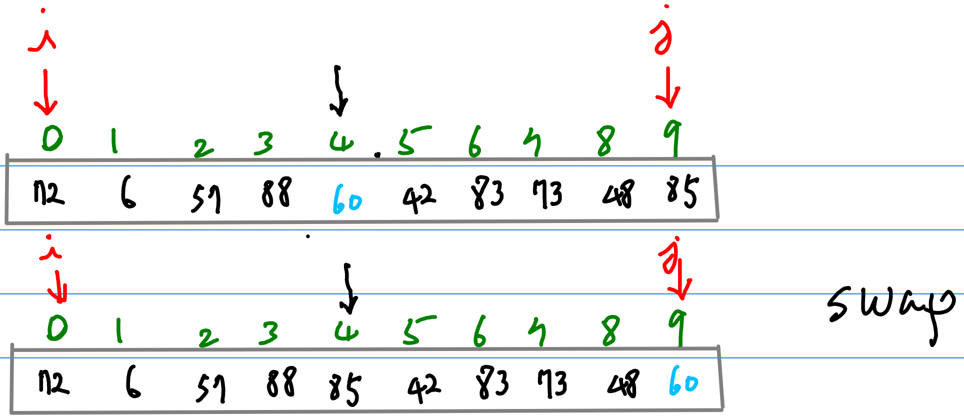
```
while (A[++l] <= pivot) ;

while ((l < r) && (pivot <= A[--r])) ;
```



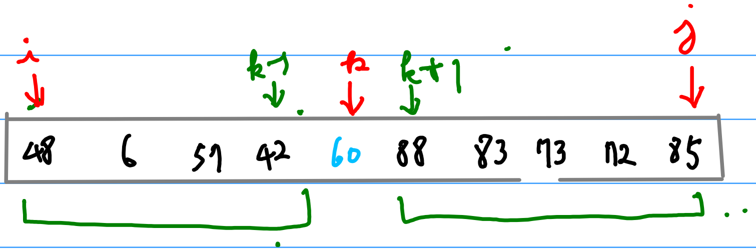
```
int k = partition<E,Comp>(A, i-1, j, A[j]);
```

l

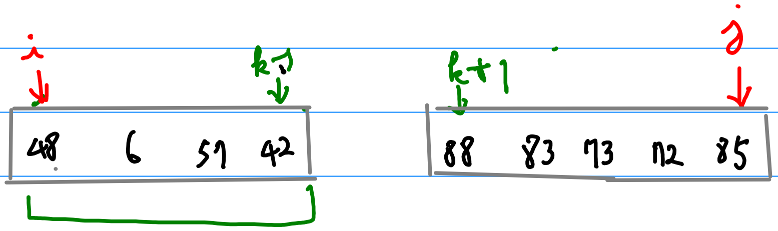




```
int k = partition<E,Comp>(A, i-1, j, A[j]);
```



```
qsort<E,Comp>(A, i, k-1);
qsort<E,Comp>(A, k+1, j);
```



i
k-1
48 6 57 42
k+1
j
88 83 73 72 85