CS240: Programming in C

Lecture 12: Polymorphic Sorting

Sorting

- Given a collection of items and a *total order* over them, sort the collection under this order.
  - Total order: every item is ordered with respect to every other time in the collection
- Given an array of numbers, sort the numbers in ascending order
  - sort in descending order
  - sort odd before even, with each sub-collection respecting an ordering
- Given an array of strings, sort according to lexiographic ordering
- Given an array of structures, sort the structures based on the value of the field named “index”
- Given an array of arrays, sort the lists based on the value of their least element

Key Observation

- The code necessary to perform the sort
  - swapping elements, rearranging the array as necessary
- works independently of the type of the elements being sorted.
- As long as the elements have a comparator that understands ordering, the same sorting function can be used to sort different kinds of elements
- Polymorphism:
  - poly : many
  - morphism : form

The Swap function

```
void swap(void *v[], int i, int j)
  {
     void *temp;

      temp = v[i];
      v[i] = v[j];
      v[j] = temp;
}
```

A void type specifies an empty set of values.

An expression that has a type denotes an expression where the value of that type is not required.

Any pointer to an object can be converted to void * without loss of information, and can be recovered to the original pointer type.

Thus, void is C's mechanism to support a generic or polymorphic type
A function with void * arguments thus accepts pointers of any type

Quicksort

Start with:

06 34 69 33 75 64 04 74 25 95 15 58 78 36 51 73 13 27

End with:

04 06 13 15 25 27 33 34 36 51 58 64 69 73 74 75 78 95

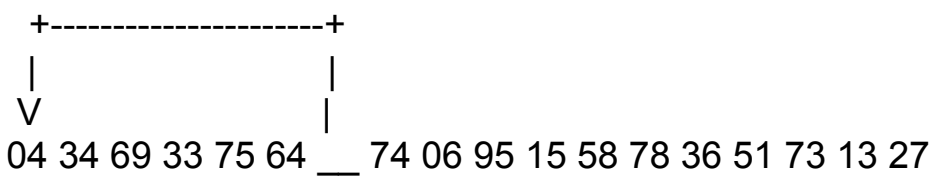
Basic idea:

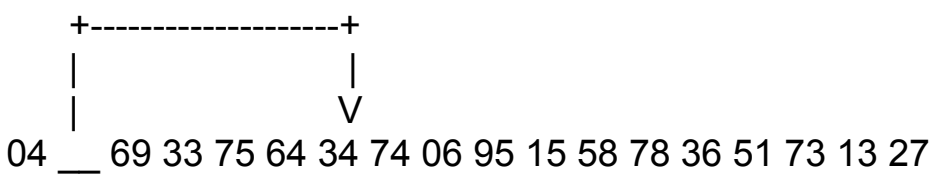
06 34 69 33 75 64 04 74 __ 95 15 58 78 36 51 73 13 27 pivot = 25

Pick an element (hopefully close to the middle) and sort around it

__ 34 69 33 75 64 04 74 06 95 15 58 78 36 51 73 13 27 pivot = 25

Start examining elements to the right of the hole until we find an element smaller than the pivot





Quicksort

After processing 06,

04 06 __ 33 75 64 34 74 69 95 15 58 78 36 51 73 13 27 pivot = 25

At the end of the process, we have:

04 06 15 13 __ 64 34 74 69 95 33 58 78 36 51 73 75 27 pivot = 25

Repeat the process for each of the two subarrays
Recursion ends with a partition has only two elements

In C ....

```
void qsort(void *v[], int left, int right,
                int (*comp)(void *, void *))
{
    int i, last;
    void swap(void *v[], int, int);

    if (left >=right)  return;
    swap(v, left, (left + right)/2);
    last = left;
    for (i = left+1; i <= right; i++)
      if ((*comp)(v[i], v[left])) < 0
            swap(v, ++last,i);
    swap(v, left, last);
    qsort(v, left, last-1, comp);
    qsort(v, last+1, right, comp)
}
```

comp is a pointer to a function that returns an int, and takes two objects of unspecified type

swap is a function that takes an array as its first argument whose elements are pointers of unspecified type

call the function pointed to by comp

Invoking quicksort

```
char *lineptr[MAXLINES]  /* pointers to text lines */

int readlines(char *lineptr[], int nlines);
void writelines(char *lineptr[], int nlines);

void qsort(void *lineptr[], int left, int right,
                int (*comp)(void *, void *));
int numcmp(char *, char *);
int numeric = ...
....
```

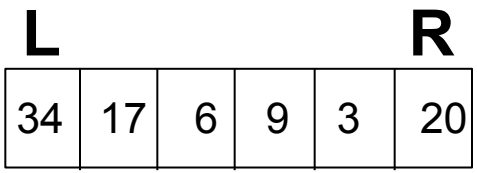
address of function

```
qsort((void **) lineptr, 0, nlines-1,
        (int (*)(void *, void *)) numeric ? numcmp: strcmp))
```

Sorting procedure

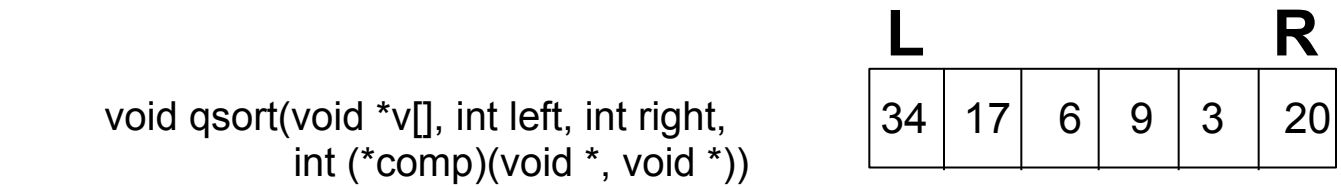
```
void qsort(void *v[], int left, int right,
                int (*comp)(void *, void *))
{
    int i, last;
    void swap(void *v[], int, int);

    if (left >=right)  return;
    swap(v, left, (left + right)/2);
    last = left;
    for (i = left+1; i <= right; i++)
      if ((*comp)(v[i], v[left])) < 0
          swap(v, ++last,i);
    swap(v, left, last);
    qsort(v, left, last-1, comp);
    qsort(v, last+1, right, comp)
}
```

Sorting procedure




```
void qsort(void *v[], int left, int right,
                int (*comp)(void *, void *))
{
    int i, last;
    void swap(void *v[], int, int);

    if (left >=right)  return;
    swap(v, left, (left + right)/2);
    last = left;
    for (i = left+1; i <= right; i++)
      if ((*comp)(v[i], v[left])) < 0
          swap(v, ++last,i);
    swap(v, left, last);
    qsort(v, left, last-1, comp);
    qsort(v, last+1, right, comp)
}
```

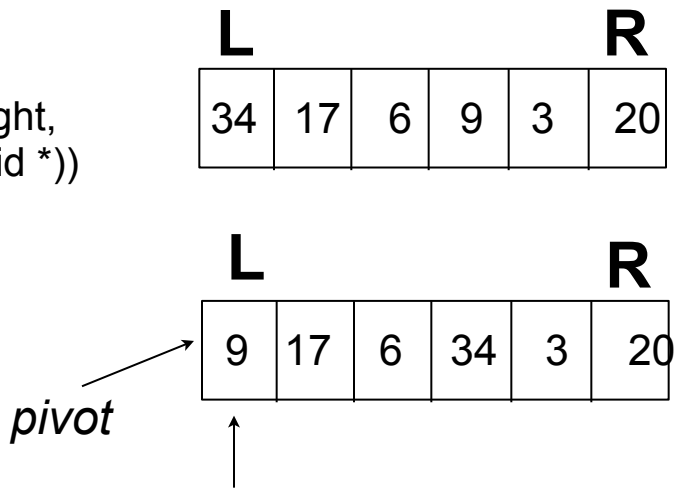
Sorting procedure




```
void qsort(void *v[], int left, int right,
                int (*comp)(void *, void *))
{
    int i, last;
    void swap(void *v[], int, int);

    if (left >=right)  return;
    swap(v, left, (left + right)/2);
    last = left;
    for (i = left+1; i <= right; i++)
      if ((*comp)(v[i], v[left])) < 0
          swap(v, ++last,i);
    swap(v, left, last);
    qsort(v, left, last-1, comp);
    qsort(v, last+1, right, comp)
}
```

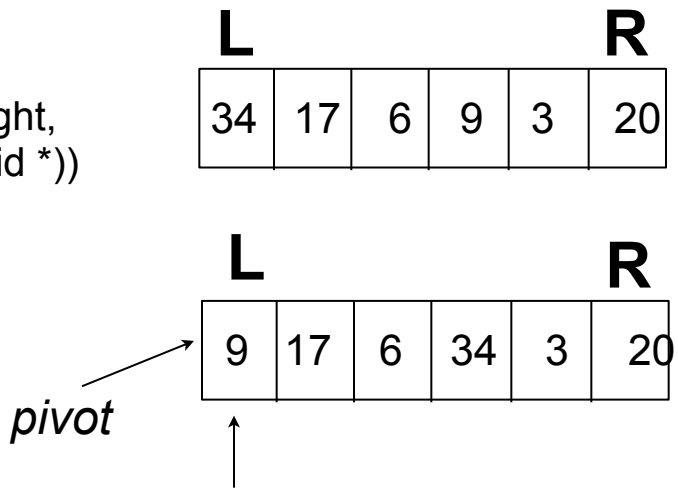
Sorting procedure

```
void qsort(void *v[], int left, int right,
                int (*comp)(void *, void *))
{
    int i, last;
    void swap(void *v[], int, int);

    if (left >=right)  return;
    swap(v, left, (left + right)/2);
    last = left;
    for (i = left+1; i <= right; i++)
      if ((*comp)(v[i], v[left])) < 0
          swap(v, ++last,i);
    swap(v, left, last);
    qsort(v, left, last-1, comp);
    qsort(v, last+1, right, comp)
}
```



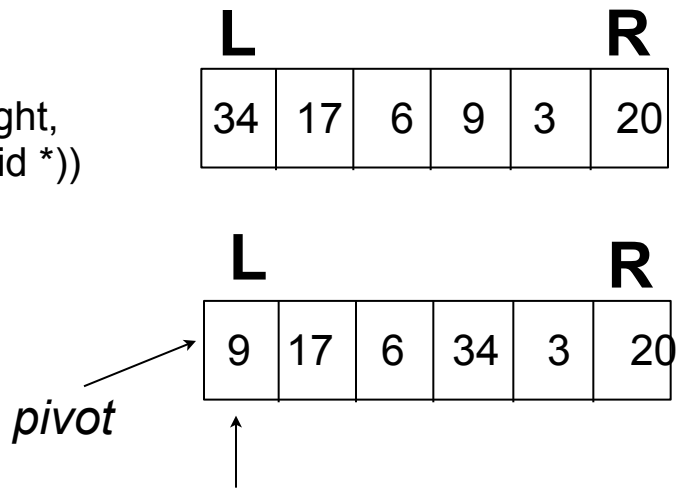
Sorting procedure

```
void qsort(void *v[], int left, int right,
                int (*comp)(void *, void *))
{
    int i, last;
    void swap(void *v[], int, int);

    if (left >=right)  return;
    swap(v, left, (left + right)/2);
    last = left;
    for (i = left+1; i <= right; i++)
      if ((*comp)(v[i], v[left])) < 0
          swap(v, ++last,i);
    swap(v, left, last);
    qsort(v, left, last-1, comp);
    qsort(v, last+1, right, comp)
}
```



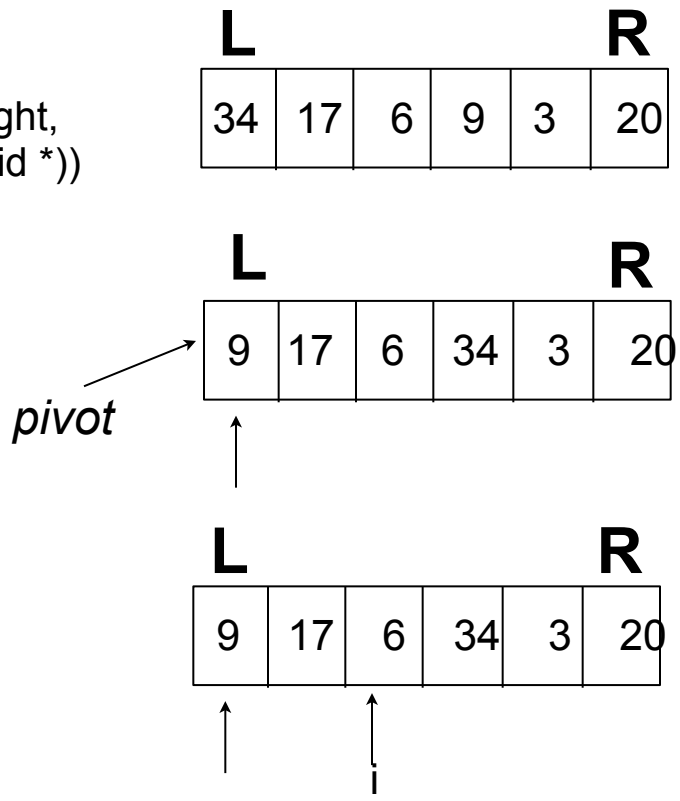
Sorting procedure

```
void qsort(void *v[], int left, int right,
           int (*comp)(void *, void *))
{
    int i, last;
    void swap(void *v[], int, int);

    if (left >=right)  return;
    swap(v, left, (left + right)/2);
    last = left;
    for (i = left+1; i <= right; i++)
      if ((*comp)(v[i], v[left])) < 0
          swap(v, ++last,i);
    swap(v, left, last);
    qsort(v, left, last-1, comp);
    qsort(v, last+1, right, comp)
}
```



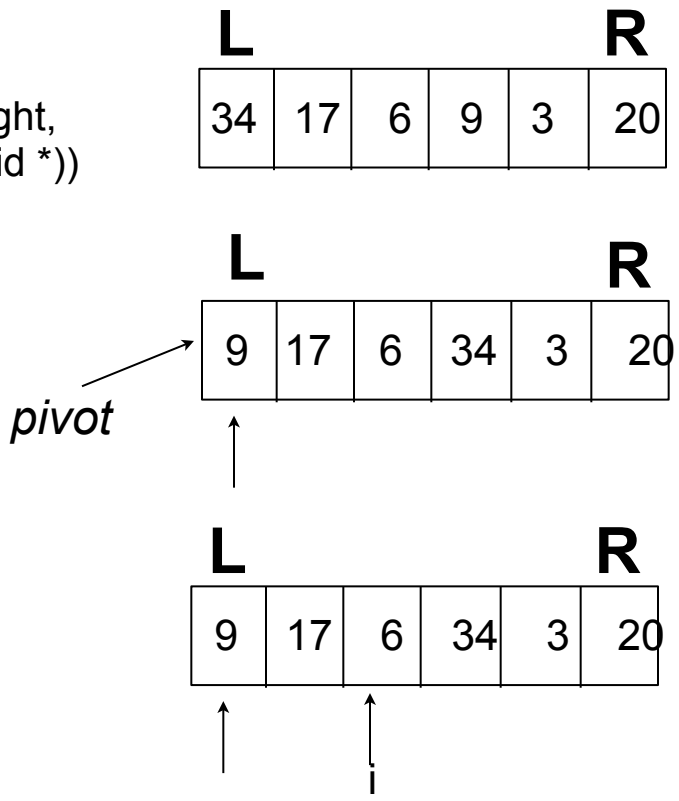
Sorting procedure

```
void qsort(void *v[], int left, int right,
           int (*comp)(void *, void *))
{
    int i, last;
    void swap(void *v[], int, int);

    if (left >=right)  return;
    swap(v, left, (left + right)/2);
    last = left;
    for (i = left+1; i <= right; i++)
      if ((*comp)(v[i], v[left])) < 0
          swap(v, ++last,i);
    swap(v, left, last);
    qsort(v, left, last-1, comp);
    qsort(v, last+1, right, comp)
}
```



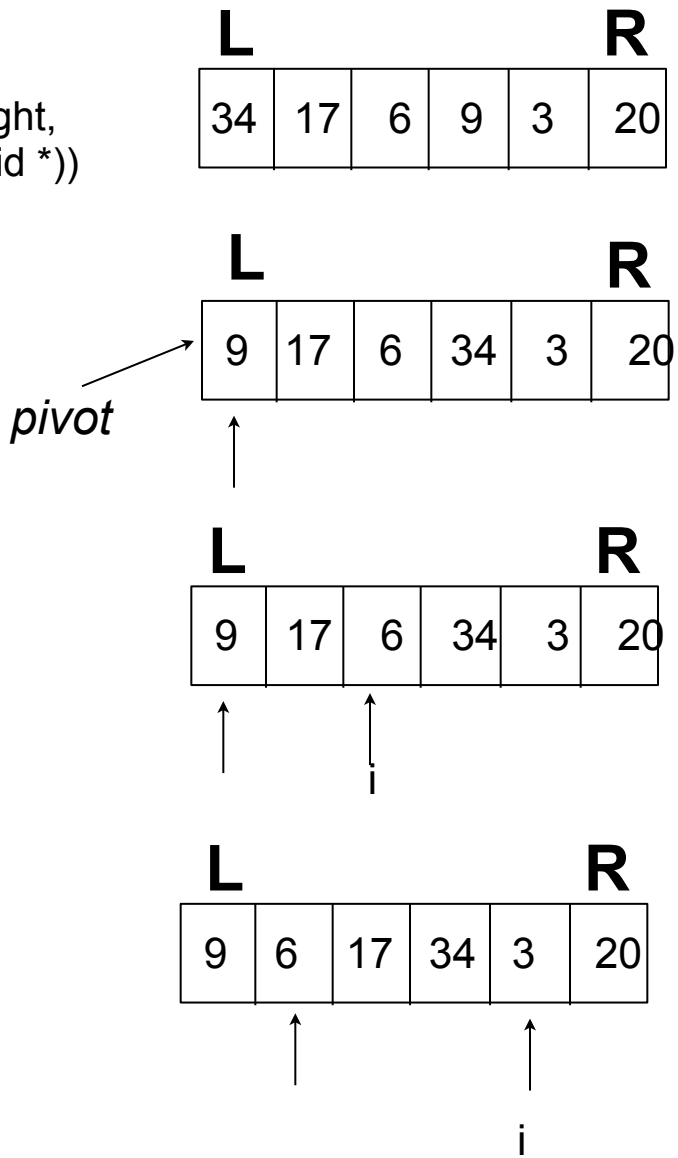
Sorting procedure

```
void qsort(void *v[], int left, int right,
                int (*comp)(void *, void *))
{
    int i, last;
    void swap(void *v[], int, int);

    if (left >=right)  return;
    swap(v, left, (left + right)/2);
    last = left;
    for (i = left+1; i <= right; i++)
      if ((*comp)(v[i], v[left])) < 0
          swap(v, ++last,i);
    swap(v, left, last);
    qsort(v, left, last-1, comp);
    qsort(v, last+1, right, comp)
}
```



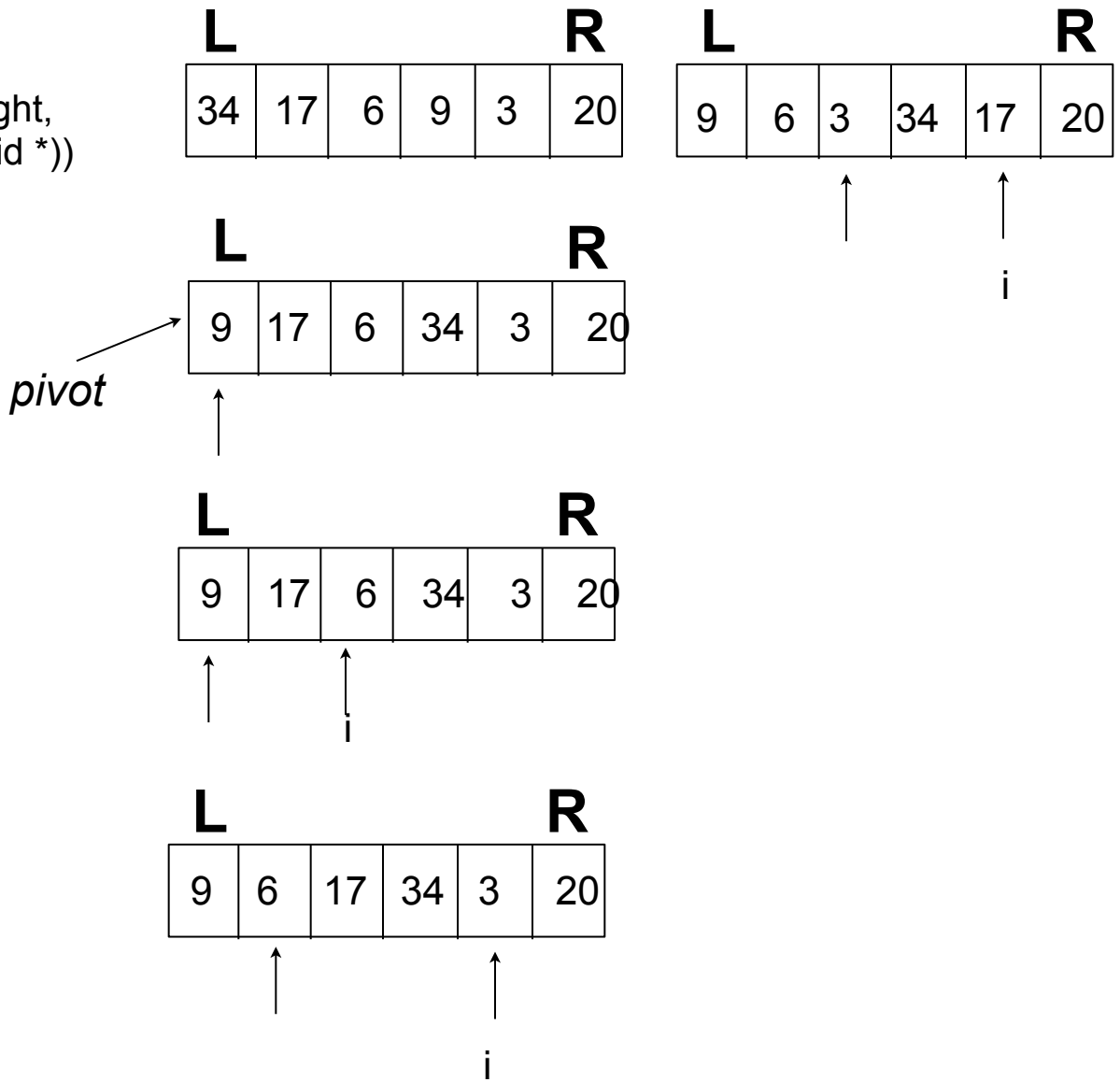
Sorting procedure

```
void qsort(void *v[], int left, int right,
                int (*comp)(void *, void *))
{
    int i, last;
    void swap(void *v[], int, int);

    if (left >=right)  return;
    swap(v, left, (left + right)/2);
    last = left;
    for (i = left+1; i <= right; i++)
      if ((*comp)(v[i], v[left])) < 0
          swap(v, ++last,i);
    swap(v, left, last);
    qsort(v, left, last-1, comp);
    qsort(v, last+1, right, comp)
}
```



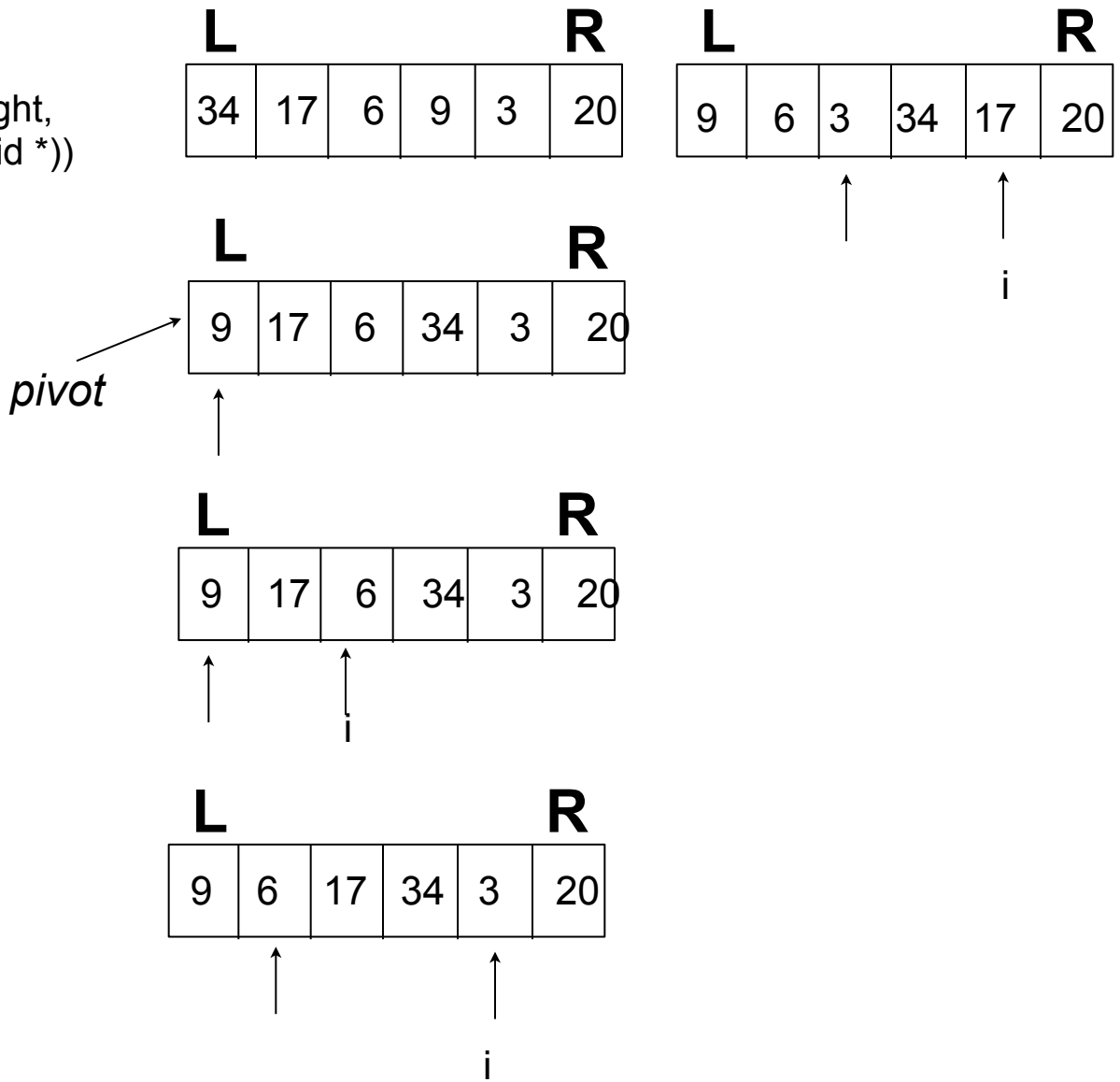
Sorting procedure

```
void qsort(void *v[], int left, int right,
                int (*comp)(void *, void *))
{
    int i, last;
    void swap(void *v[], int, int);

    if (left >=right)  return;
    swap(v, left, (left + right)/2);
    last = left;
    for (i = left+1; i <= right; i++)
      if ((*comp)(v[i], v[left])) < 0
          swap(v, ++last,i);
    swap(v, left, last);
    qsort(v, left, last-1, comp);
    qsort(v, last+1, right, comp)
}
```



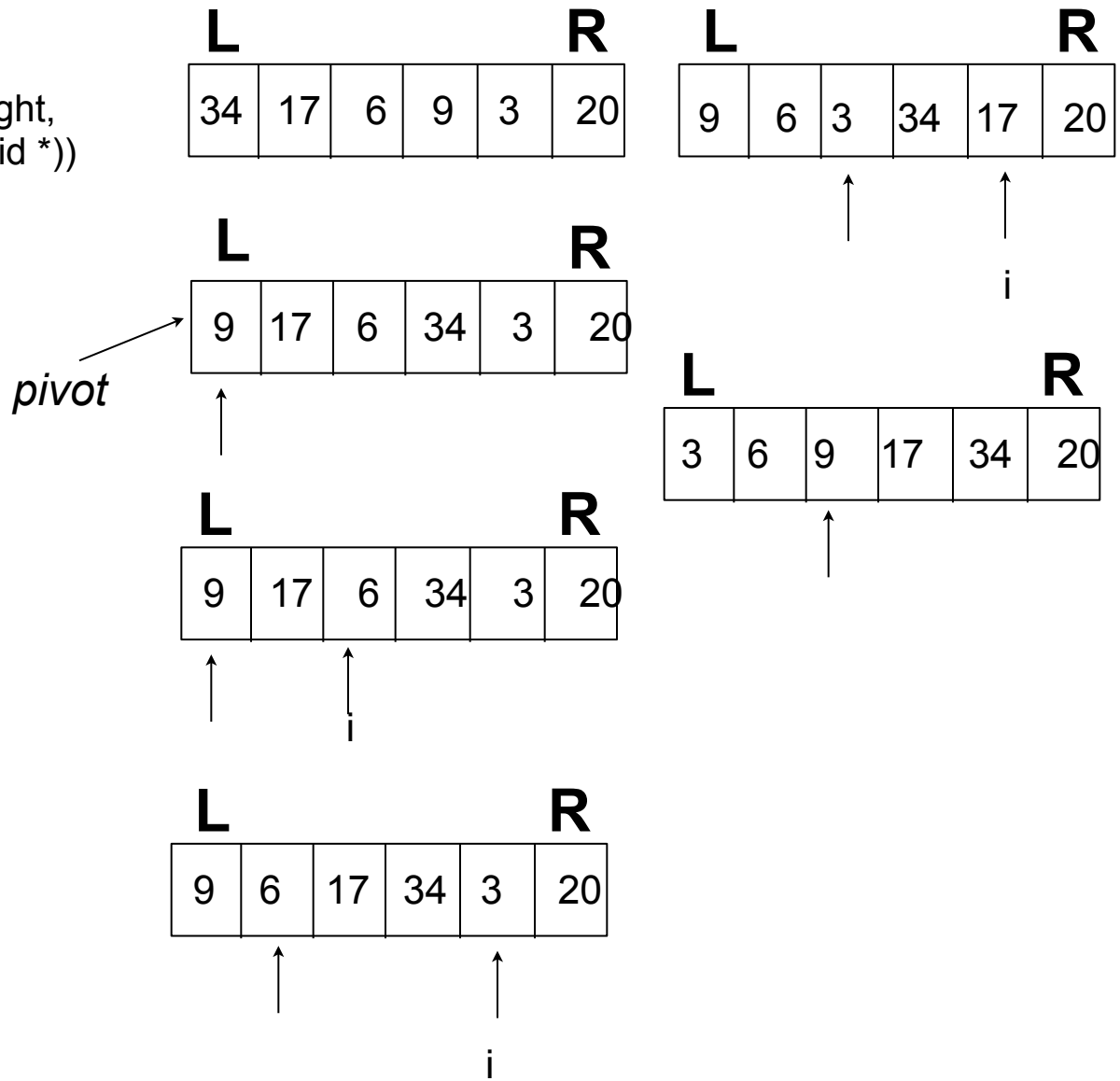
Sorting procedure

```
void qsort(void *v[], int left, int right,
           int (*comp)(void *, void *))
{
    int i, last;
    void swap(void *v[], int, int);

    if (left >=right)  return;
    swap(v, left, (left + right)/2);
    last = left;
    for (i = left+1; i <= right; i++)
      if ((*comp)(v[i], v[left])) < 0
          swap(v, ++last,i);
    swap(v, left, last);
    qsort(v, left, last-1, comp);
    qsort(v, last+1, right, comp)
}
```



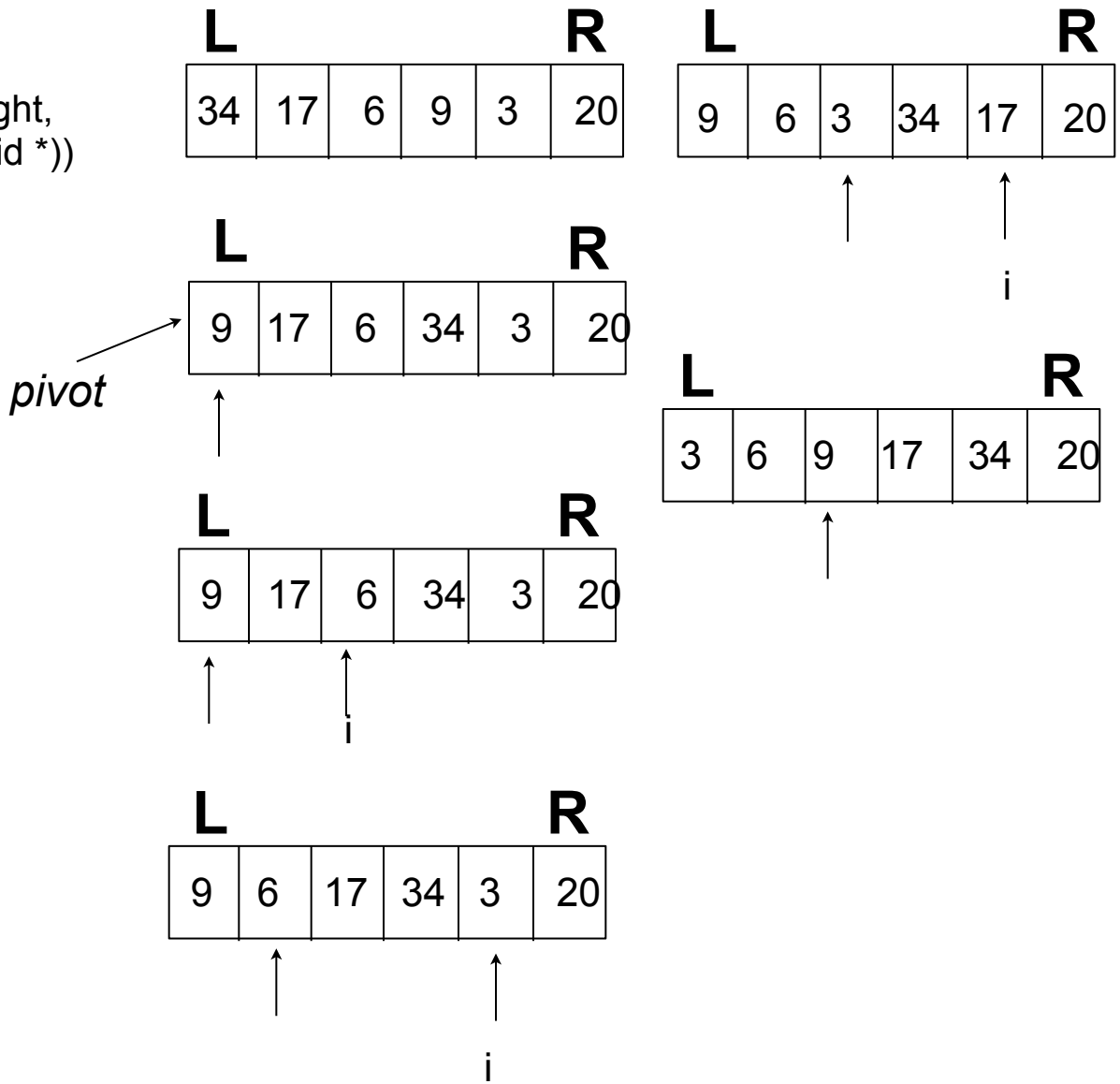
Sorting procedure

```
void qsort(void *v[], int left, int right,
           int (*comp)(void *, void *))
{
    int i, last;
    void swap(void *v[], int, int);

    if (left >=right)  return;
    swap(v, left, (left + right)/2);
    last = left;
    for (i = left+1; i <= right; i++)
      if ((*comp)(v[i], v[left])) < 0
          swap(v, ++last,i);
    swap(v, left, last);
    qsort(v, left, last-1, comp);
    qsort(v, last+1, right, comp)
}
```



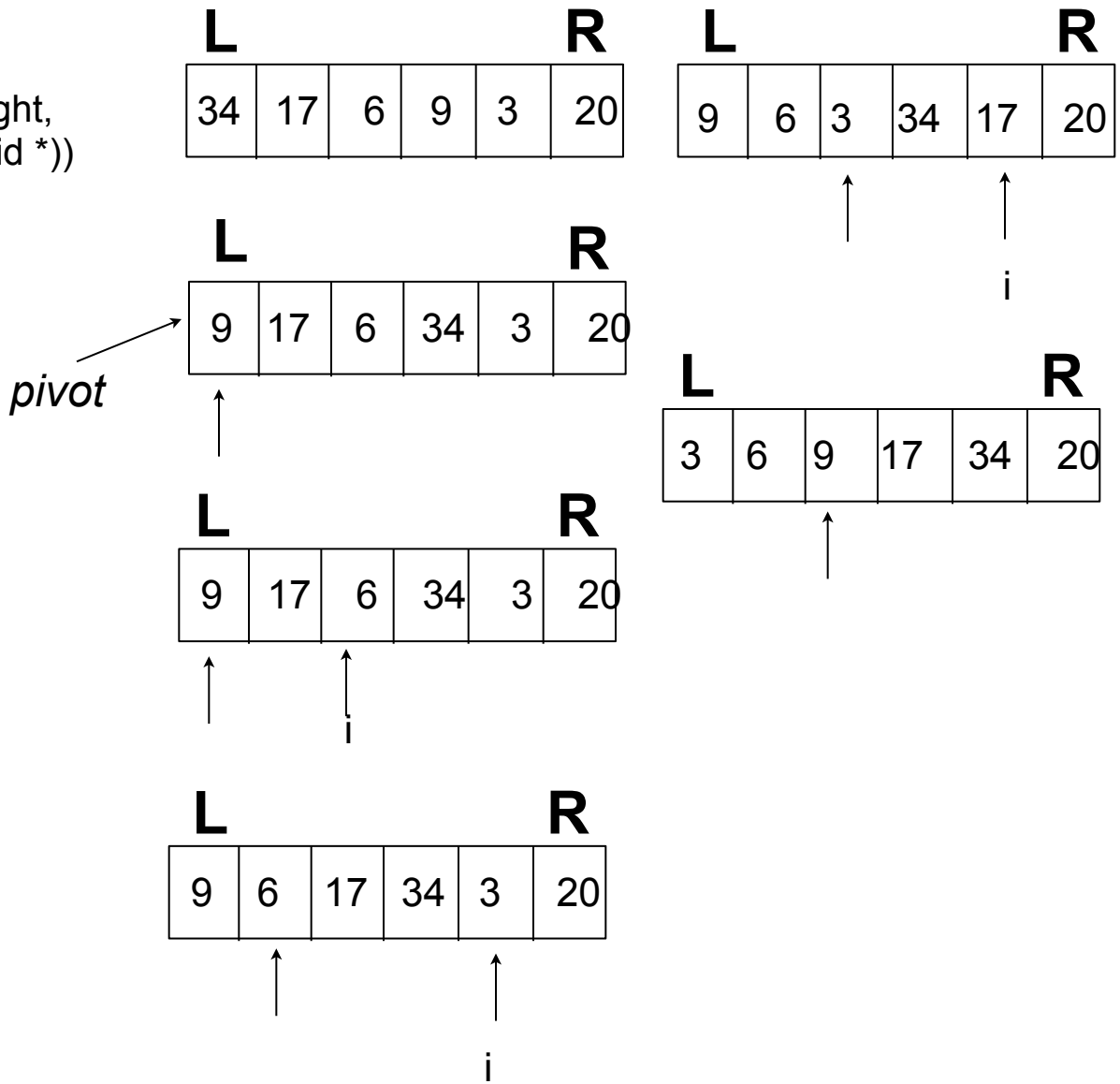
Sorting procedure

```
void qsort(void *v[], int left, int right,
                int (*comp)(void *, void *))
{
    int i, last;
    void swap(void *v[], int, int);

    if (left >=right)  return;
    swap(v, left, (left + right)/2);
    last = left;
    for (i = left+1; i <= right; i++)
      if ((*comp)(v[i], v[left])) < 0
          swap(v, ++last,i);
    swap(v, left, last);
    qsort(v, left, last-1, comp);
    qsort(v, last+1, right, comp)
}
```



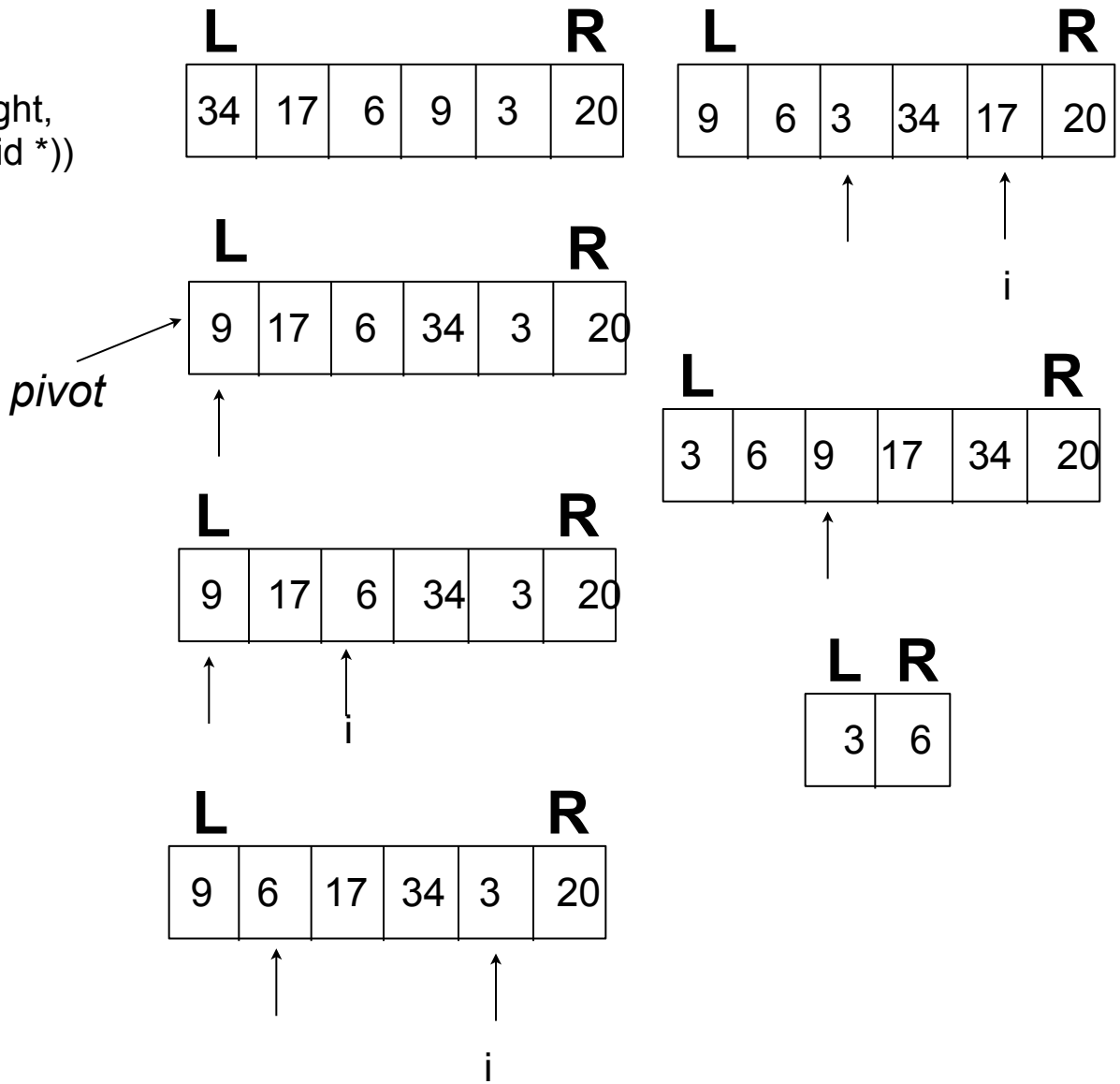
Sorting procedure

```
void qsort(void *v[], int left, int right,
           int (*comp)(void *, void *))
{
    int i, last;
    void swap(void *v[], int, int);

    if (left >=right)  return;
    swap(v, left, (left + right)/2);
    last = left;
    for (i = left+1; i <= right; i++)
      if ((*comp)(v[i], v[left])) < 0
          swap(v, ++last,i);
    swap(v, left, last);
    qsort(v, left, last-1, comp);
    qsort(v, last+1, right, comp)
}
```



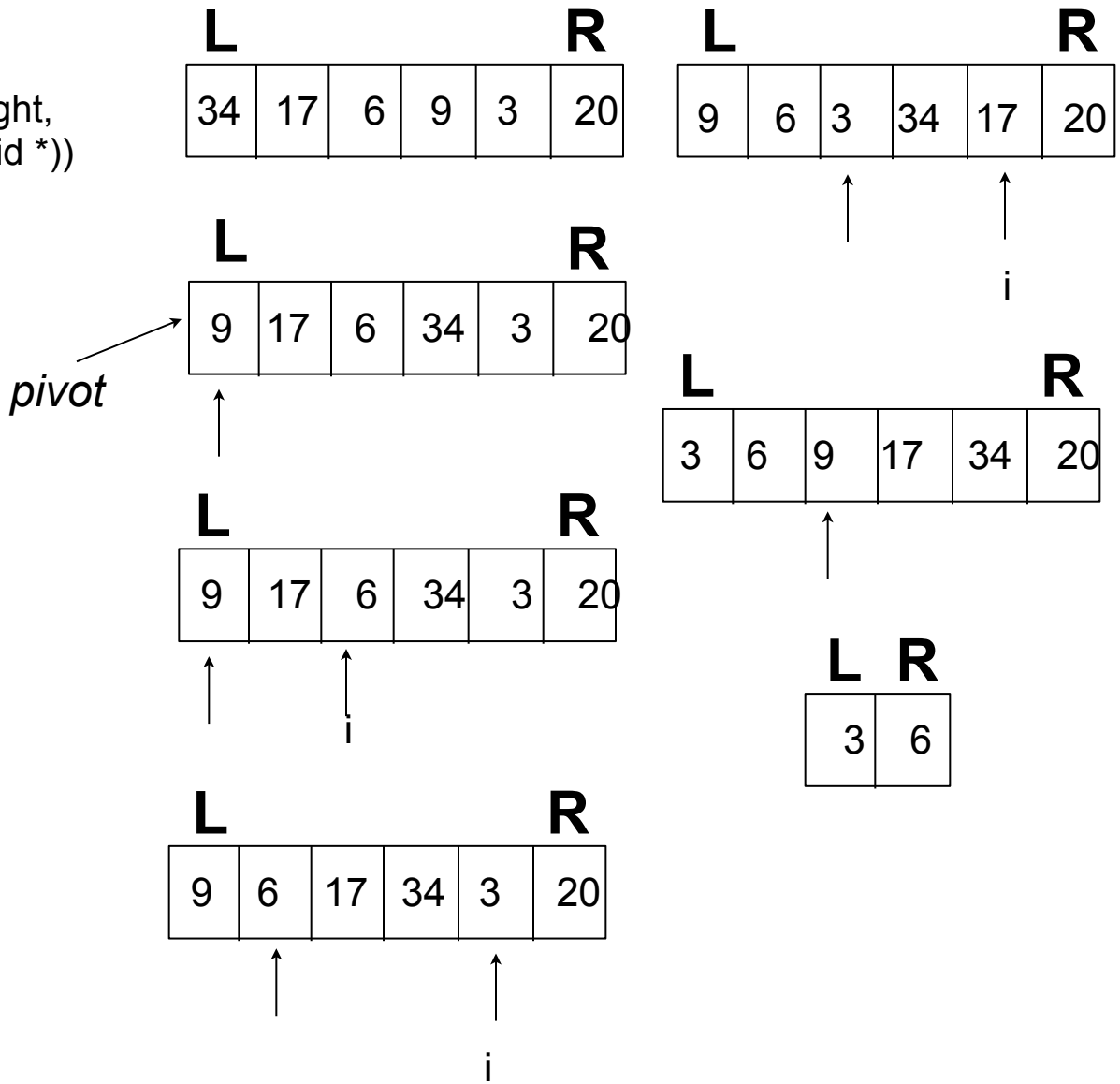
Sorting procedure

```
void qsort(void *v[], int left, int right,
           int (*comp)(void *, void *))
{
    int i, last;
    void swap(void *v[], int, int);

    if (left >=right)  return;
    swap(v, left, (left + right)/2);
    last = left;
    for (i = left+1; i <= right; i++)
      if ((*comp)(v[i], v[left])) < 0
          swap(v, ++last,i);
    swap(v, left, last);
    qsort(v, left, last-1, comp);
    qsort(v, last+1, right, comp)
}
```



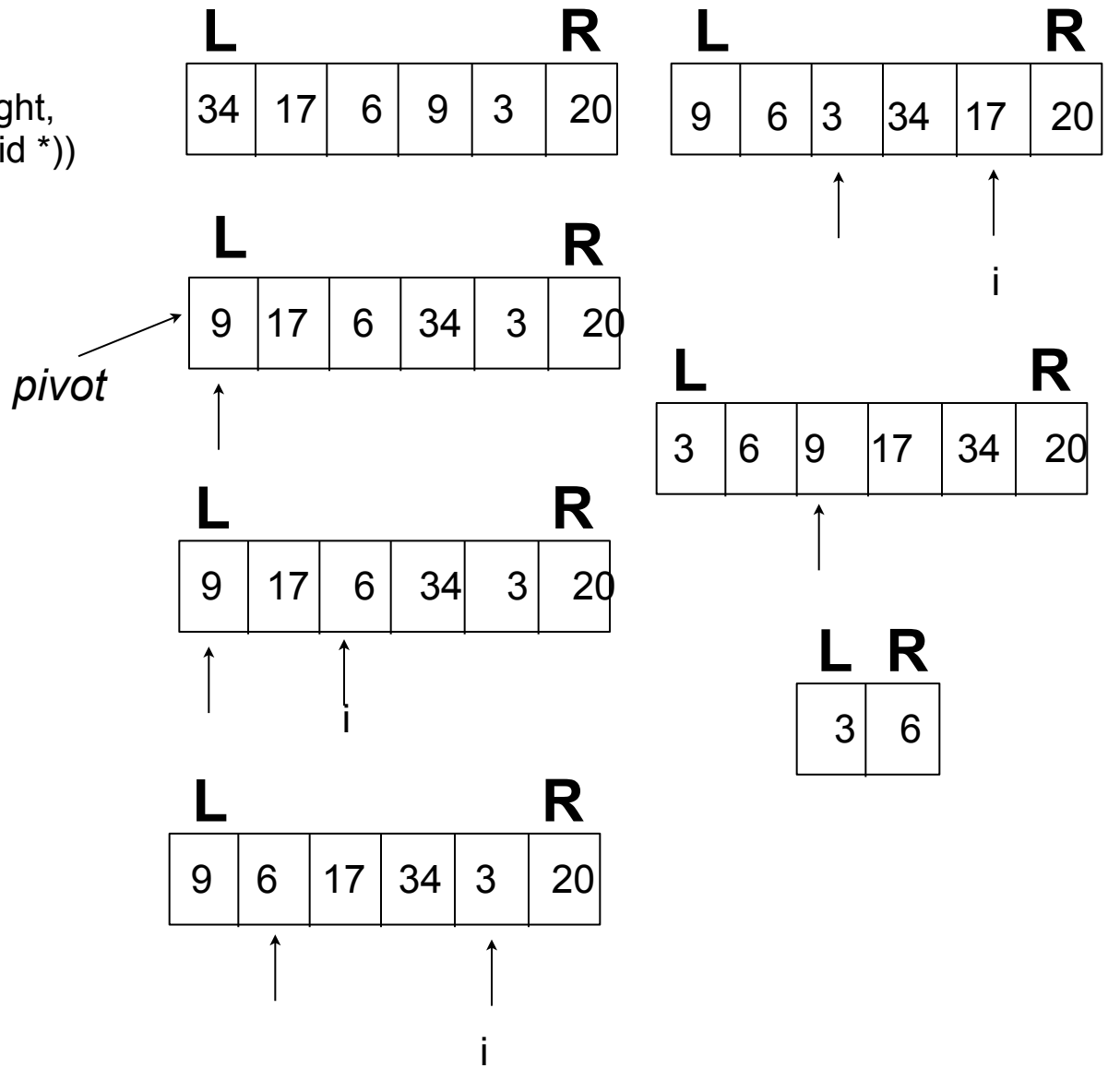
Sorting procedure

```
void qsort(void *v[], int left, int right,
                int (*comp)(void *, void *))
{
    int i, last;
    void swap(void *v[], int, int);

    if (left >=right)  return;
    swap(v, left, (left + right)/2);
    last = left;
    for (i = left+1; i <= right; i++)
      if ((*comp)(v[i], v[left])) < 0
          swap(v, ++last,i);
    swap(v, left, last);
    qsort(v, left, last-1, comp);
    qsort(v, last+1, right, comp)
}
```



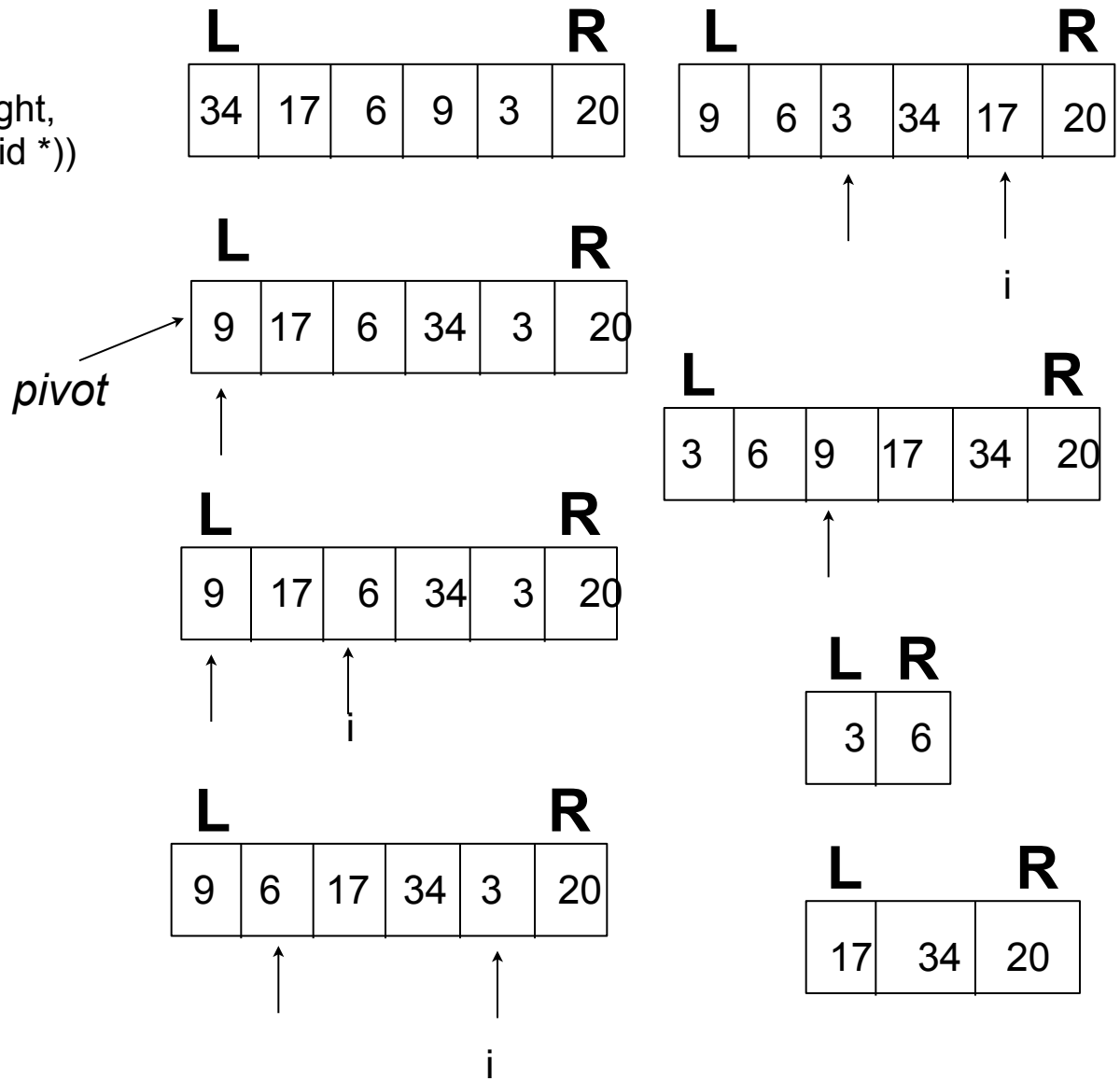
Sorting procedure

```
void qsort(void *v[], int left, int right,
           int (*comp)(void *, void *))
{
    int i, last;
    void swap(void *v[], int, int);

    if (left >=right)  return;
    swap(v, left, (left + right)/2);
    last = left;
    for (i = left+1; i <= right; i++)
      if ((*comp)(v[i], v[left])) < 0
          swap(v, ++last,i);
    swap(v, left, last);
    qsort(v, left, last-1, comp);
    qsort(v, last+1, right, comp)
}
```



Sorting procedure

```
void qsort(void *v[], int left, int right,
           int (*comp)(void *, void *))
{
    int i, last;
    void swap(void *v[], int, int);

    if (left >=right)  return;
    swap(v, left, (left + right)/2);
    last = left;
    for (i = left+1; i <= right; i++)
      if ((*comp)(v[i], v[left])) < 0
          swap(v, ++last,i);
    swap(v, left, last);
    qsort(v, left, last-1, comp);
    qsort(v, last+1, right, comp)
}
```



Sorting procedure

```
void qsort(void *v[], int left, int right,
                int (*comp)(void *, void *))
{
    int i, last;
    void swap(void *v[], int, int);

    if (left >=right)  return;
    swap(v, left, (left + right)/2);
    last = left;
    for (i = left+1; i <= right; i++)
      if ((*comp)(v[i], v[left])) < 0
          swap(v, ++last,i);
    swap(v, left, last);
    qsort(v, left, last-1, comp);
    qsort(v, last+1, right, comp)
}
```