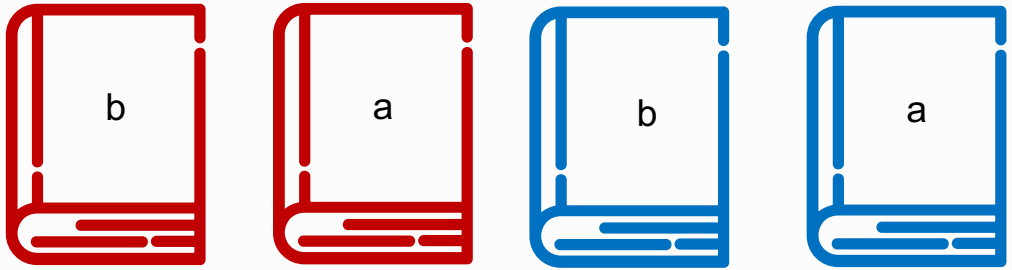
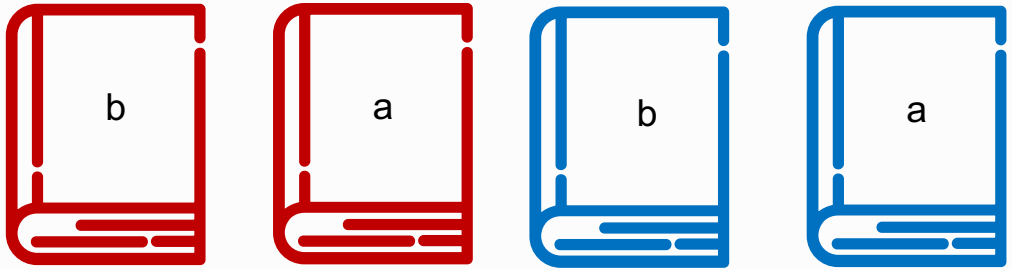
Week 3 Sorting

COMP2521 23T3

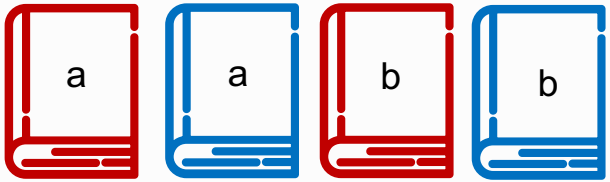
Stability



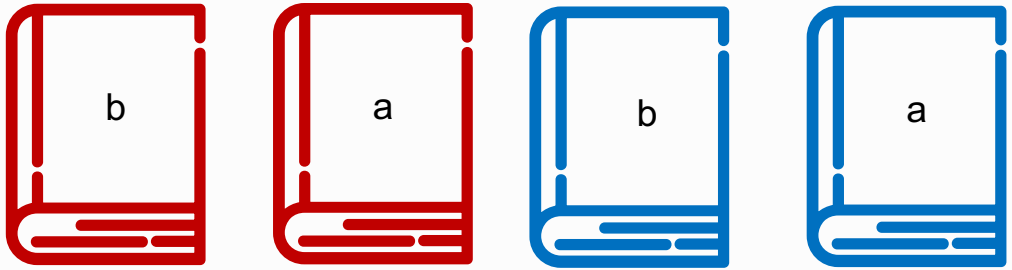
Stability



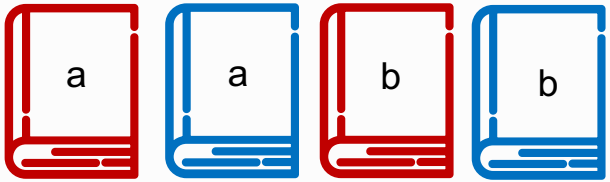
Stable sorting algorithm



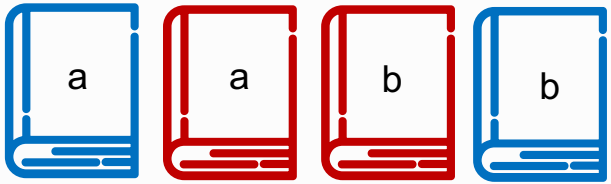
Stability



Stable sorting algorithm



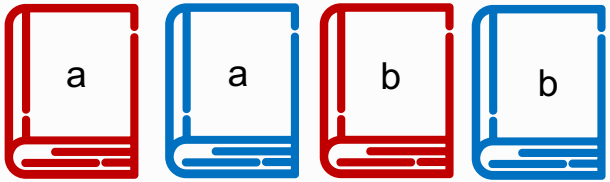
Unstable sorting algorithm



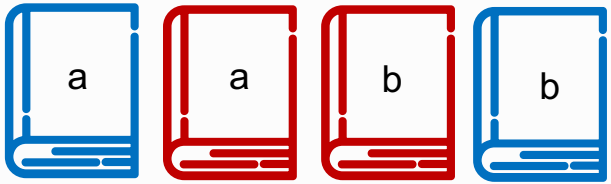
Stability

Has the relative order of equal elements changed?

Stable sorting algorithm



Unstable sorting algorithm



COMP1927	Jane	3970
COMP1927	John	3978
COMP1927	Pete	3978
MATH1231	John	3978
MATH1231	Adam	3970
PSYC1011	Adam	3970
PSYC1011	Jane	3970

Sort the table of student records by student name, so that we can easily see what courses each student is studying.

Show an example of what the final array would look like if

a) we used a stable sorting algorithm
b) we used an unstable sorting algorithm

Stable Sorting

~~COMP1927~~	~~Jane~~	~~3970~~
~~COMP1927~~	~~John~~	~~3978~~
~~COMP1927~~	~~Pete~~	~~3978~~
~~MATH1231~~	~~John~~	~~3978~~
~~MATH1231~~	~~Adam~~	~~3970~~
~~PSYC1011~~	~~Adam~~	~~3970~~
~~PSYC1011~~	~~Jane~~	~~3970~~

MATH1231	Adam	3970
PSYC1011	Adam	3970
COMP1927	Jane	3970
PSYC1011	Jane	3970
COMP1927	John	3978
MATH1231	John	3978
COMP1927	Pete	3978

Unstable Sorting

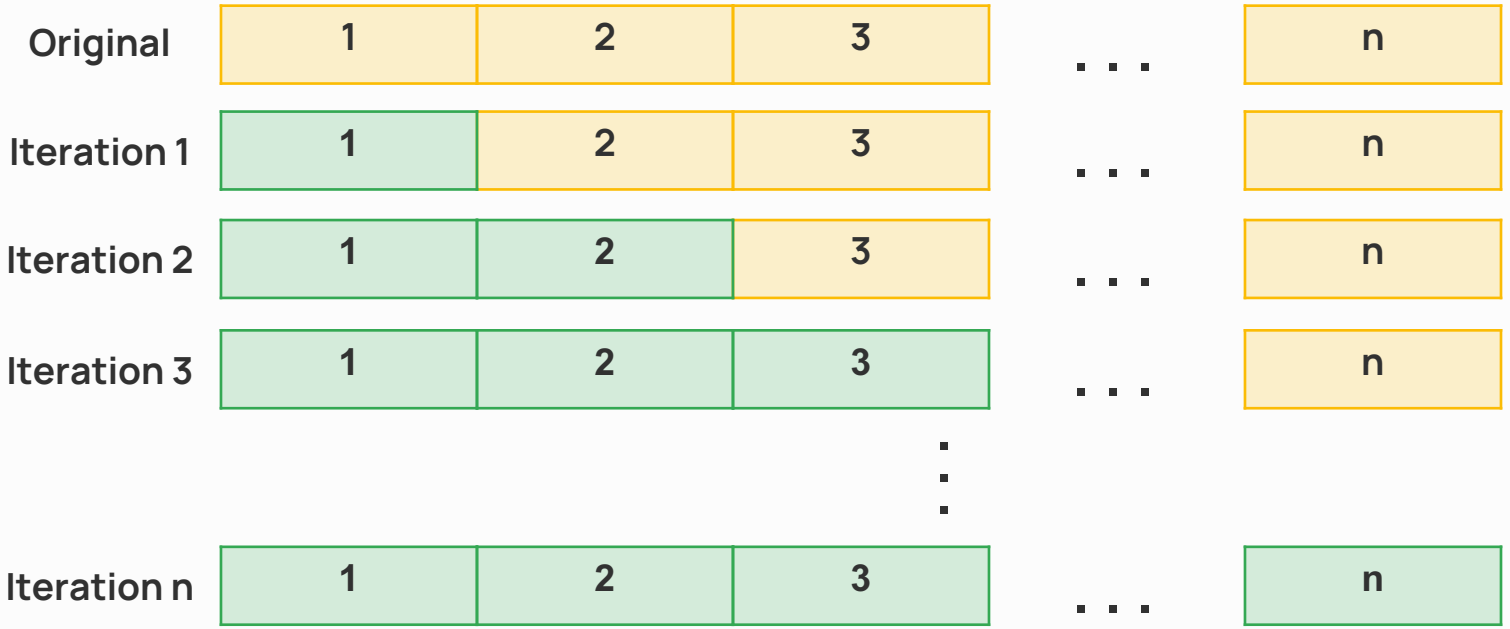
~~COMP1927~~	~~Jane~~	~~3970~~
~~COMP1927~~	~~John~~	~~3978~~
~~COMP1927~~	~~Pete~~	~~3978~~
~~MATH1231~~	~~John~~	~~3978~~
~~MATH1231~~	~~Adam~~	~~3970~~
~~PSYC1011~~	~~Adam~~	~~3970~~
~~PSYC1011~~	~~Jane~~	~~3970~~

PSYC1011	Adam	3970
MATH1231	Adam	3970
COMP1927	Jane	3970
PSYC1011	Jane	3970
MATH1231	John	3978
COMP1927	John	3978
COMP1927	Pete	3978

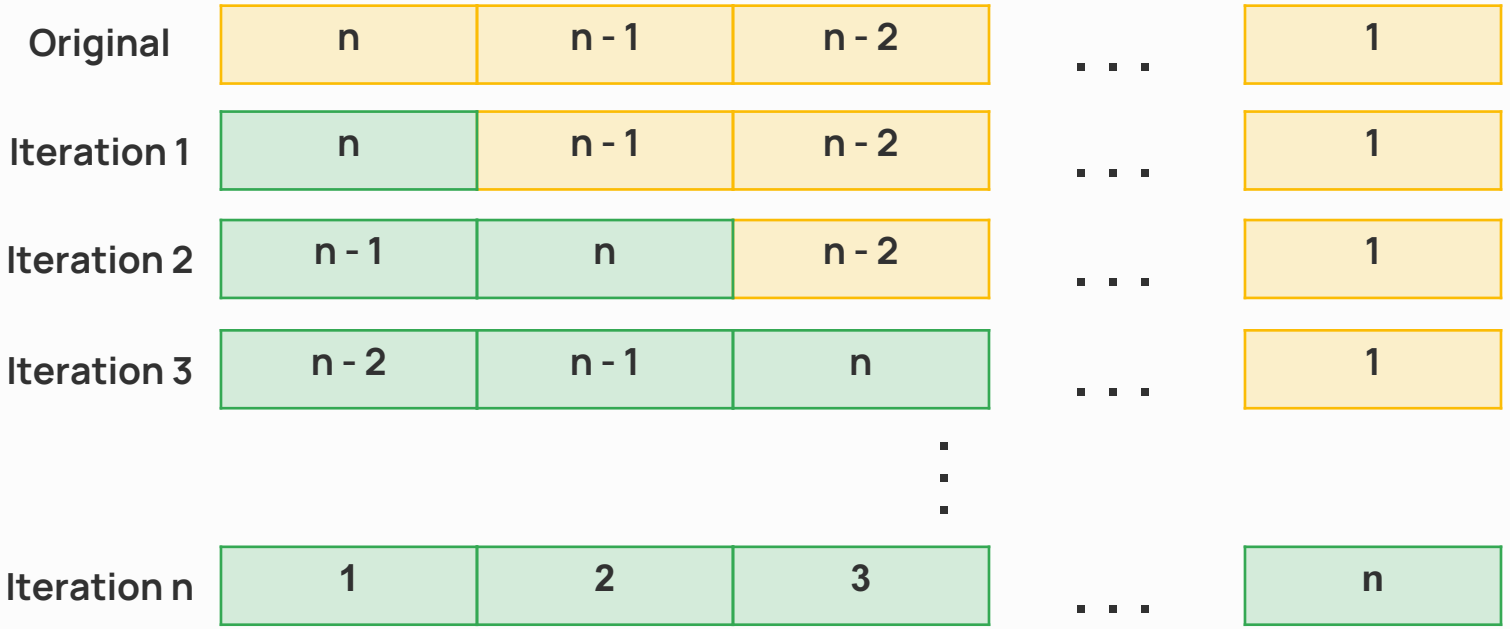
Insertion Sort

```
void insertionSort(int a[], int lo, int hi)
{
   int i, j, val;
   for (i = lo+1; i <= hi; i++) {
      val = a[i];
      for (j = i; j > lo; j--) {
         if (!less(val,a[j-1])) break;
         a[j] = a[j-1];
      }
      a[j] = val;
   }
}
```

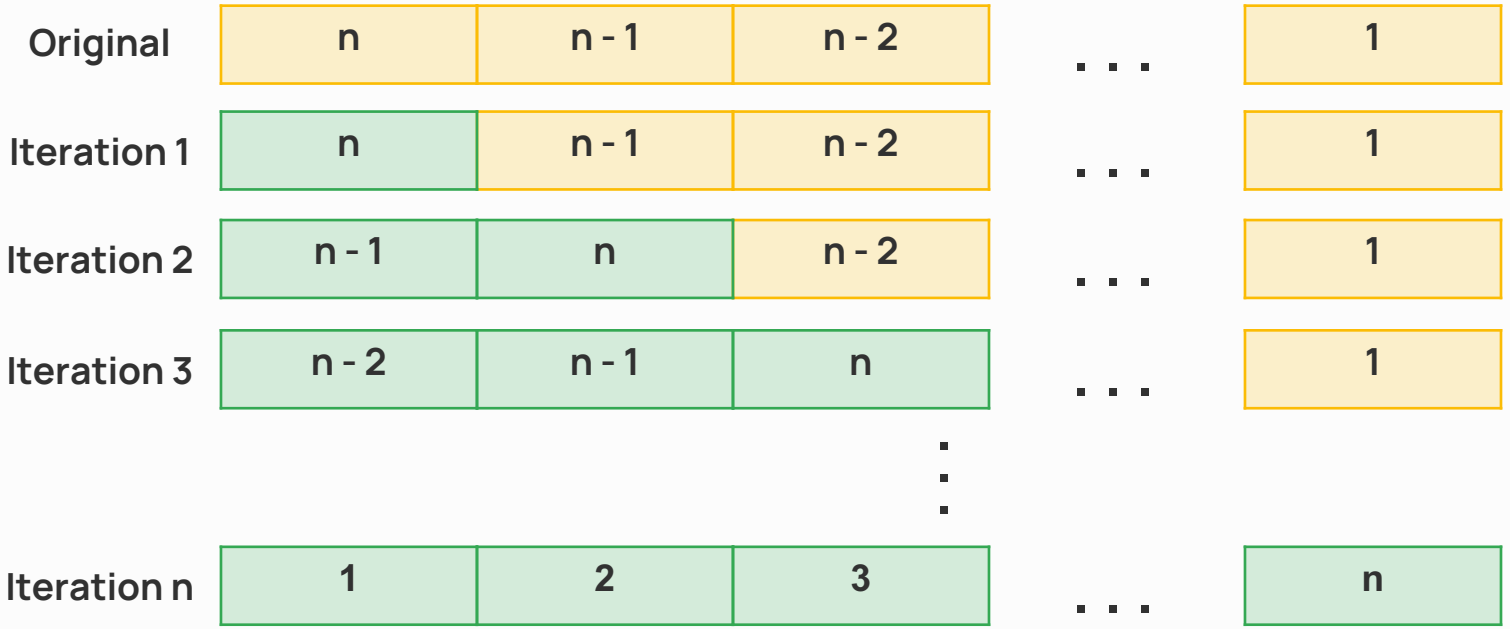
Insertion Sort



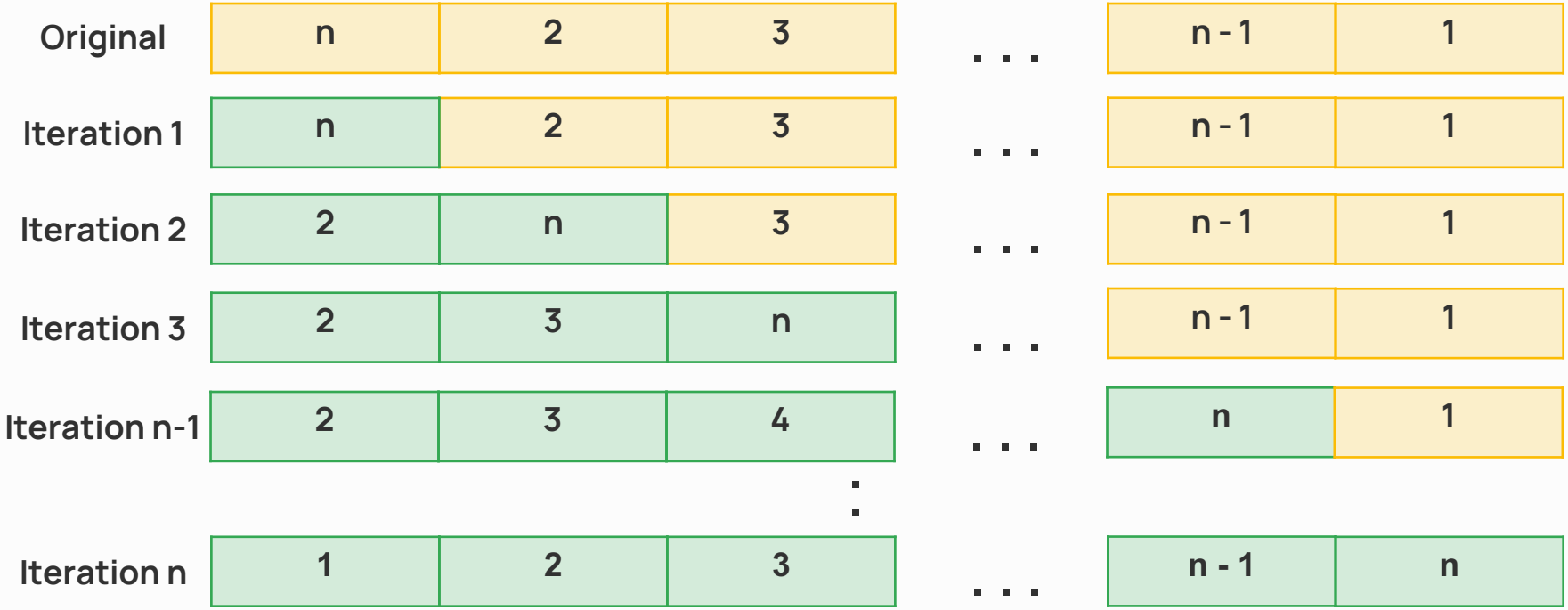
Insertion Sort



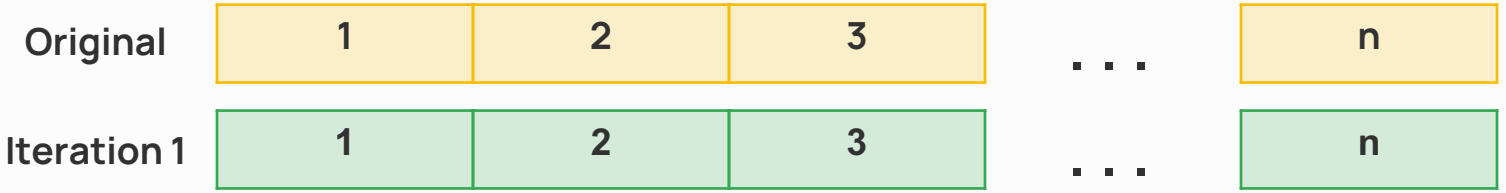
Insertion Sort



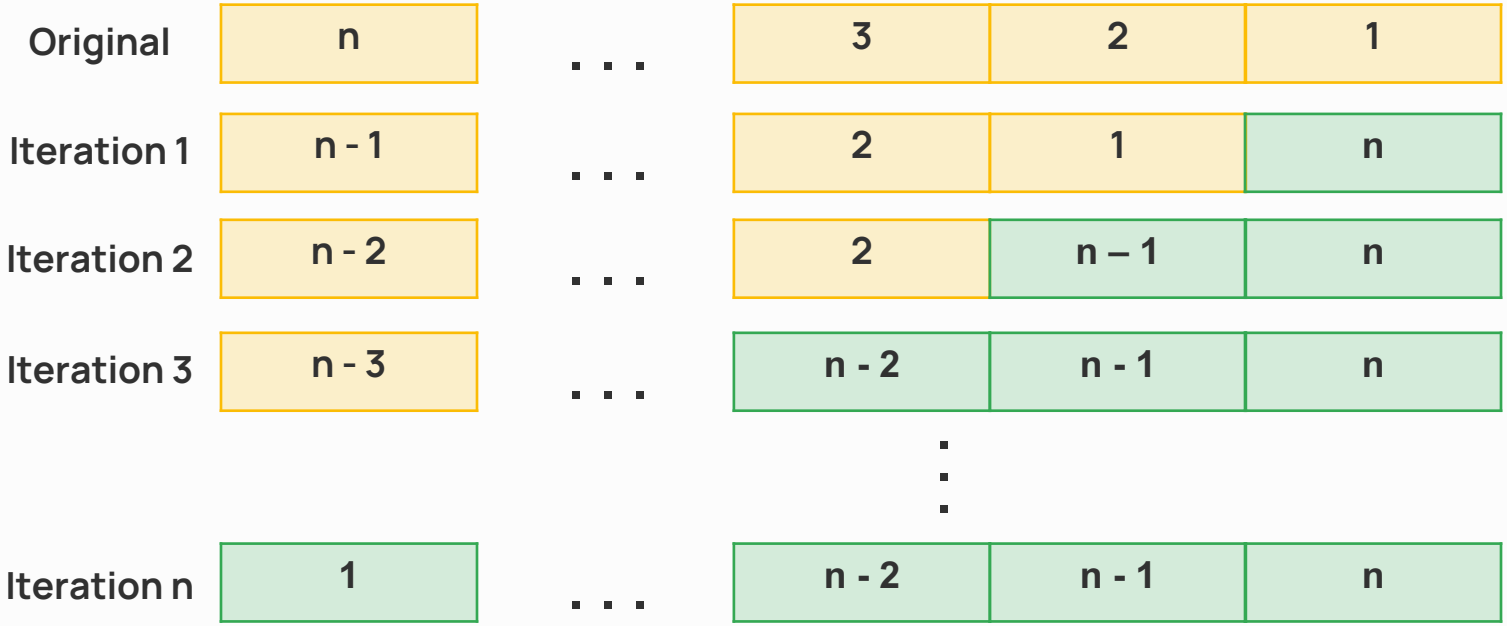
Insertion Sort



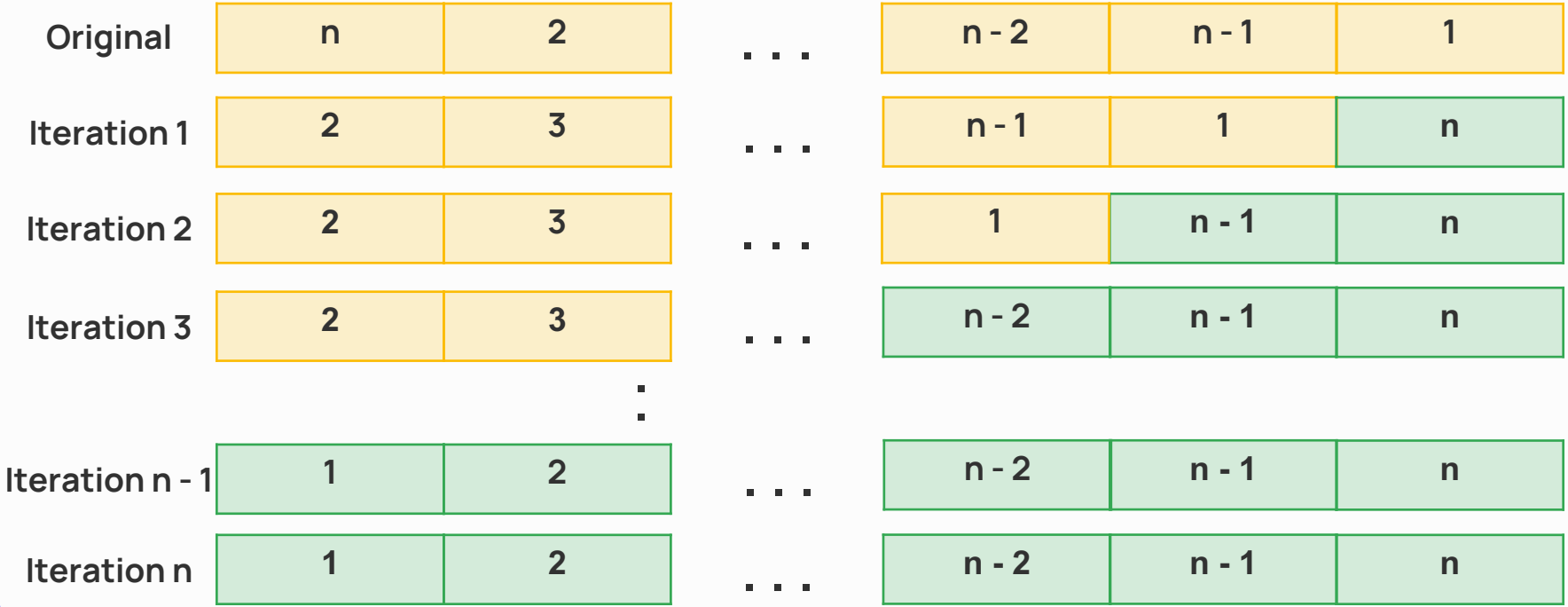
Bubble Sort



Bubble Sort


Original
n
3
2
1
Iteration 1
n - 1
2
1
n
Iteration 2
n - 2
2
n – 1
n
Iteration 3
n - 3
n - 2
n - 1
n
Iteration n
1
n - 2
n - 1
n

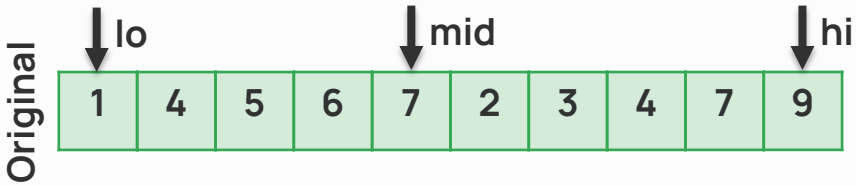
Bubble Sort


Original
n
2
. . .
n - 2
n - 1
1
Iteration 1
2
3
. . .
n - 1
1
n
Iteration 2
2
3
. . .
1
n - 1
n
Iteration 3
2
3
. . .
n - 2
n - 1
n
Iteration n - 1
1
2
. . .
n - 2
n - 1
n
Iteration n
1
2
. . .
n - 2
n - 1
n

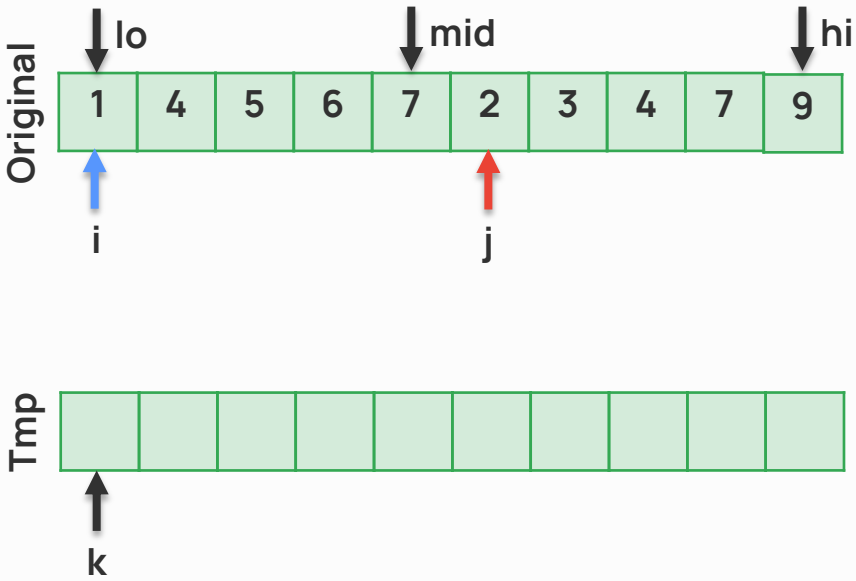
Merge Sort

```
void mergesort(Item a[], int lo, int hi)
{
    int mid = (lo+hi)/2; // mid point
    if (hi <= lo) return;
    mergesort(a, lo, mid);
    mergesort(a, mid+1, hi);
    merge(a, lo, mid, hi);
}
```

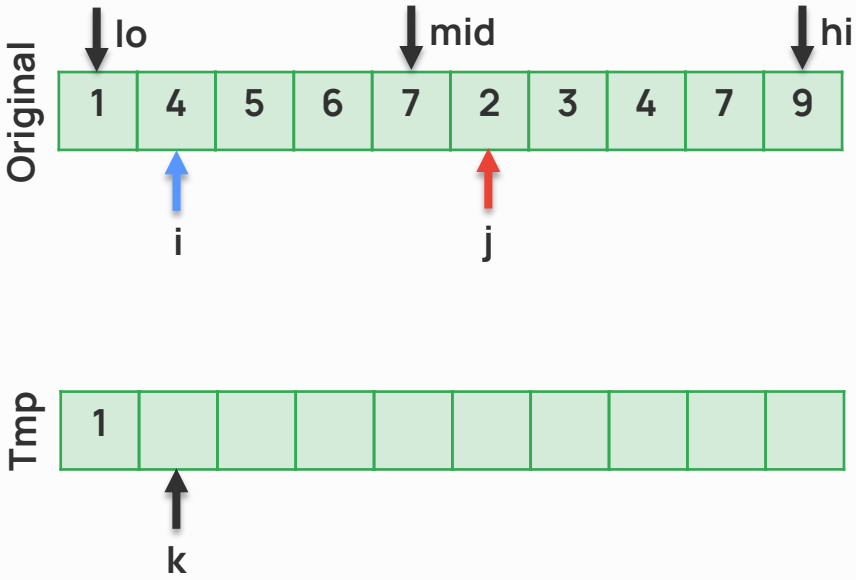
```
void merge(Item a[], int lo, int mid, int hi)
{
   int  i, j, k, nitems = hi-lo+1;
   Item *tmp = malloc(nitems*sizeof(Item));

   i = lo; j = mid+1; k = 0;
   // scan both segments, copying to tmp
   while (i <= mid && j <= hi) {
     if (less(a[i],a[j]))
        tmp[k++] = a[i++];
     else
        tmp[k++] = a[j++];
   }
   // copy items from unfinished segment
   while (i <= mid) tmp[k++] = a[i++];
   while (j <= hi) tmp[k++] = a[j++];

   //copy tmp back to main array
   for (i = lo, k = 0; i <= hi; i++, k++)
      a[i] = tmp[k];
   free(tmp);
}
```



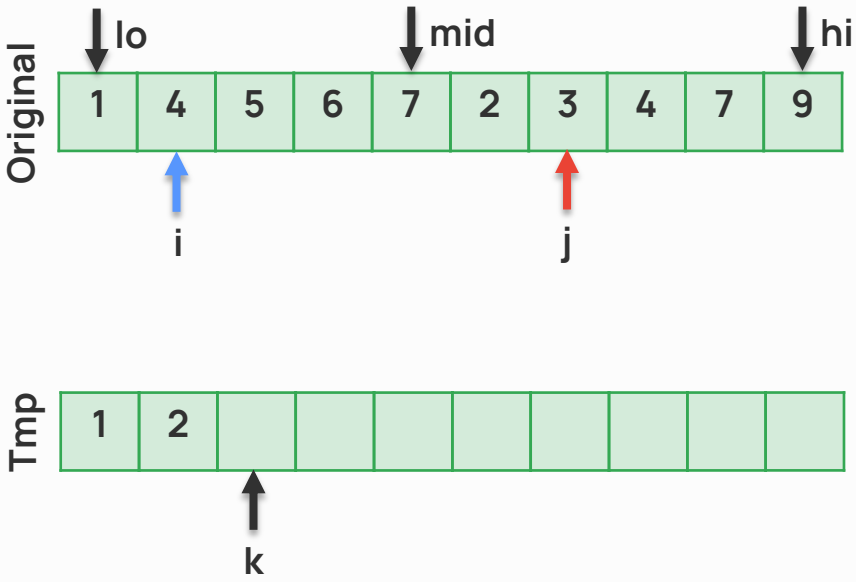
```
void merge(Item a[], int lo, int mid, int hi)
{
   int  i, j, k, nitems = hi-lo+1;
   Item *tmp = malloc(nitems*sizeof(Item));

   i = lo; j = mid+1; k = 0;
   // scan both segments, copying to tmp
   while (i <= mid && j <= hi) {
     if (less(a[i],a[j]))
        tmp[k++] = a[i++];
     else
        tmp[k++] = a[j++];
   }
   // copy items from unfinished segment
   while (i <= mid) tmp[k++] = a[i++];
   while (j <= hi) tmp[k++] = a[j++];

   //copy tmp back to main array
   for (i = lo, k = 0; i <= hi; i++, k++)
      a[i] = tmp[k];
   free(tmp);
}
```



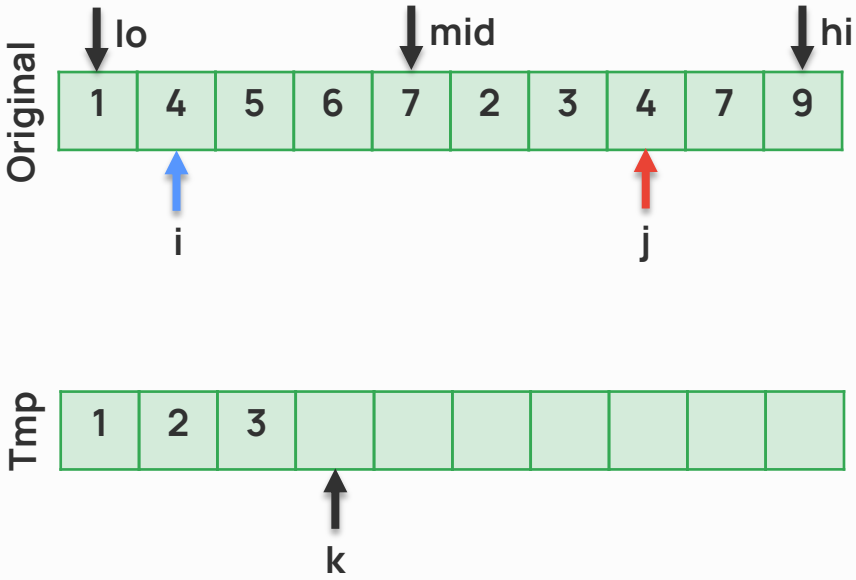
```
void merge(Item a[], int lo, int mid, int hi)
{
   int  i, j, k, nitems = hi-lo+1;
   Item *tmp = malloc(nitems*sizeof(Item));

   i = lo; j = mid+1; k = 0;
   // scan both segments, copying to tmp
   while (i <= mid && j <= hi) {
      if (less(a[i],a[j]))
         tmp[k++] = a[i++];
      else
         tmp[k++] = a[j++];
   }
   // copy items from unfinished segment
   while (i <= mid) tmp[k++] = a[i++];
   while (j <= hi) tmp[k++] = a[j++];

   //copy tmp back to main array
   for (i = lo, k = 0; i <= hi; i++, k++)
      a[i] = tmp[k];
   free(tmp);
}
```



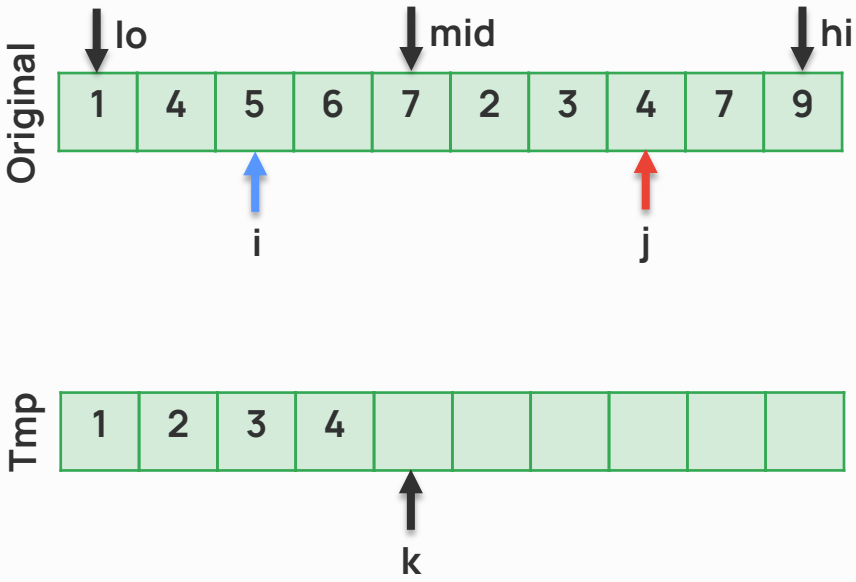
```
void merge(Item a[], int lo, int mid, int hi)
{
   int  i, j, k, nitems = hi-lo+1;
   Item *tmp = malloc(nitems*sizeof(Item));

   i = lo; j = mid+1; k = 0;
   // scan both segments, copying to tmp
   while (i <= mid && j <= hi) {
     if (less(a[i],a[j]))
        tmp[k++] = a[i++];
     else
        tmp[k++] = a[j++];
   }
   // copy items from unfinished segment
   while (i <= mid) tmp[k++] = a[i++];
   while (j <= hi) tmp[k++] = a[j++];

   //copy tmp back to main array
   for (i = lo, k = 0; i <= hi; i++, k++)
      a[i] = tmp[k];
   free(tmp);
}
```



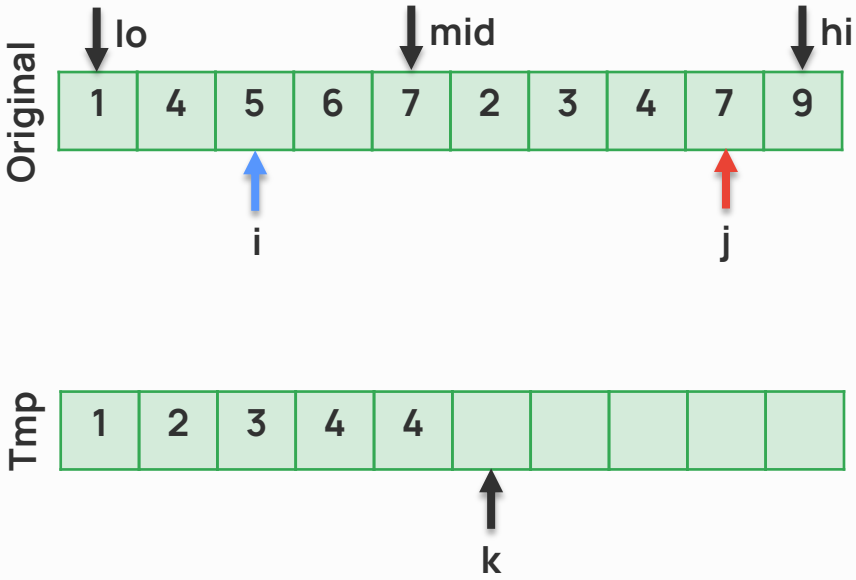
```
void merge(Item a[], int lo, int mid, int hi)
{
   int  i, j, k, nitems = hi-lo+1;
   Item *tmp = malloc(nitems*sizeof(Item));

   i = lo; j = mid+1; k = 0;
   // scan both segments, copying to tmp
   while (i <= mid && j <= hi) {
     if (less(a[i],a[j]))
        tmp[k++] = a[i++];
     else
        tmp[k++] = a[j++];
   }
   // copy items from unfinished segment
   while (i <= mid) tmp[k++] = a[i++];
   while (j <= hi) tmp[k++] = a[j++];

   //copy tmp back to main array
   for (i = lo, k = 0; i <= hi; i++, k++)
      a[i] = tmp[k];
   free(tmp);
}
```



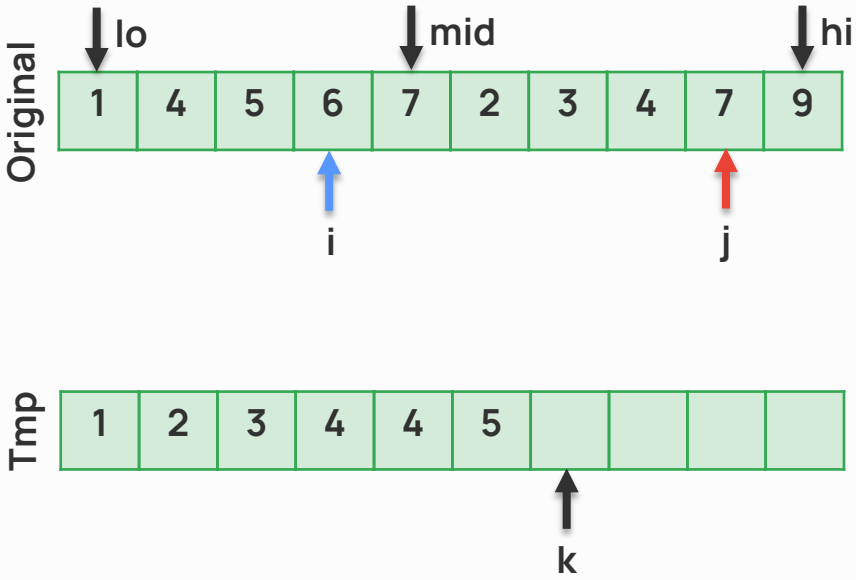
```
void merge(Item a[], int lo, int mid, int hi)
{
   int  i, j, k, nitems = hi-lo+1;
   Item *tmp = malloc(nitems*sizeof(Item));

   i = lo; j = mid+1; k = 0;
   // scan both segments, copying to tmp
   while (i <= mid && j <= hi) {
      if (less(a[i],a[j]))
         tmp[k++] = a[i++];
      else
         tmp[k++] = a[j++];
   }
   // copy items from unfinished segment
   while (i <= mid) tmp[k++] = a[i++];
   while (j <= hi) tmp[k++] = a[j++];

   //copy tmp back to main array
   for (i = lo, k = 0; i <= hi; i++, k++)
      a[i] = tmp[k];
   free(tmp);
}
```



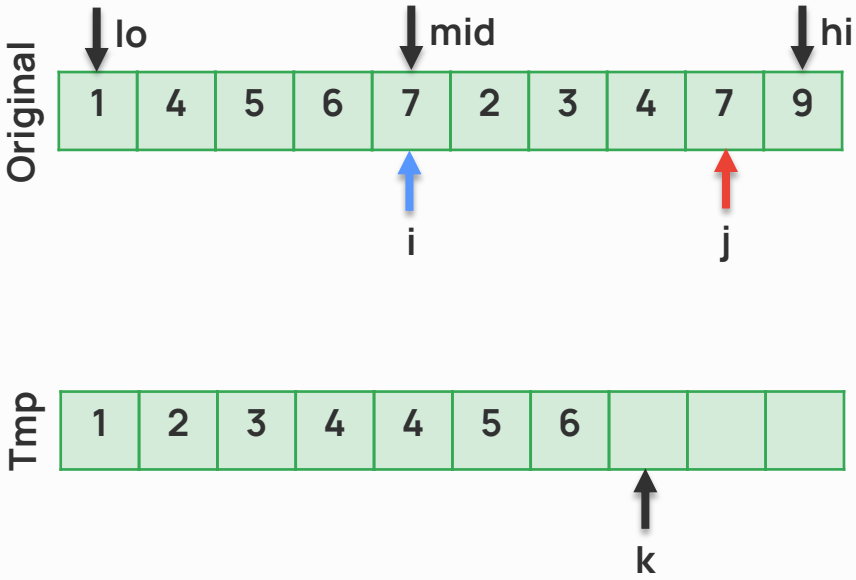
```
void merge(Item a[], int lo, int mid, int hi)
{
   int  i, j, k, nitems = hi-lo+1;
   Item *tmp = malloc(nitems*sizeof(Item));

   i = lo; j = mid+1; k = 0;
   // scan both segments, copying to tmp
   while (i <= mid && j <= hi) {
     if (less(a[i],a[j]))
        tmp[k++] = a[i++];
     else
        tmp[k++] = a[j++];
   }
   // copy items from unfinished segment
   while (i <= mid) tmp[k++] = a[i++];
   while (j <= hi) tmp[k++] = a[j++];

   //copy tmp back to main array
   for (i = lo, k = 0; i <= hi; i++, k++)
      a[i] = tmp[k];
   free(tmp);
}
```



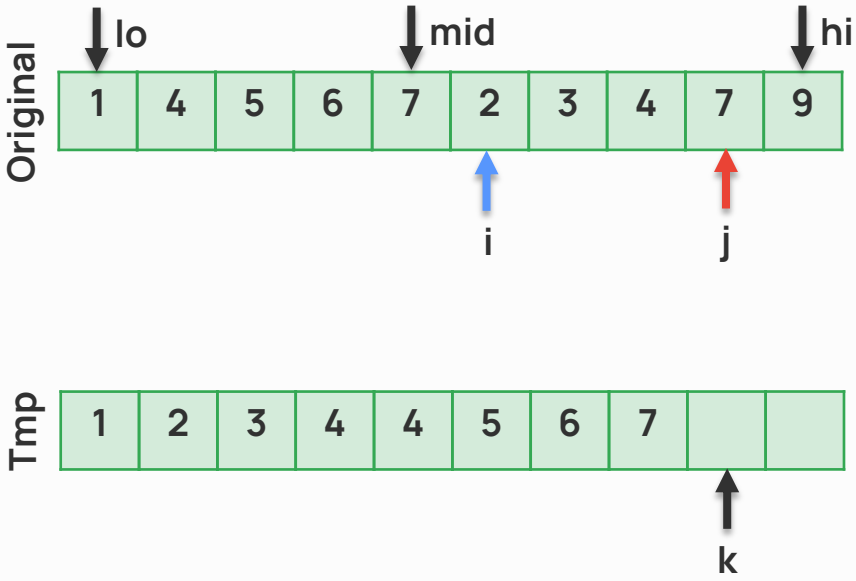
```
void merge(Item a[], int lo, int mid, int hi)
{
   int  i, j, k, nitems = hi-lo+1;
   Item *tmp = malloc(nitems*sizeof(Item));

   i = lo; j = mid+1; k = 0;
   // scan both segments, copying to tmp
   while (i <= mid && j <= hi) {
      if (less(a[i],a[j]))
         tmp[k++] = a[i++];
      else
         tmp[k++] = a[j++];
   }
   // copy items from unfinished segment
   while (i <= mid) tmp[k++] = a[i++];
   while (j <= hi) tmp[k++] = a[j++];

   //copy tmp back to main array
   for (i = lo, k = 0; i <= hi; i++, k++)
      a[i] = tmp[k];
   free(tmp);
}
```



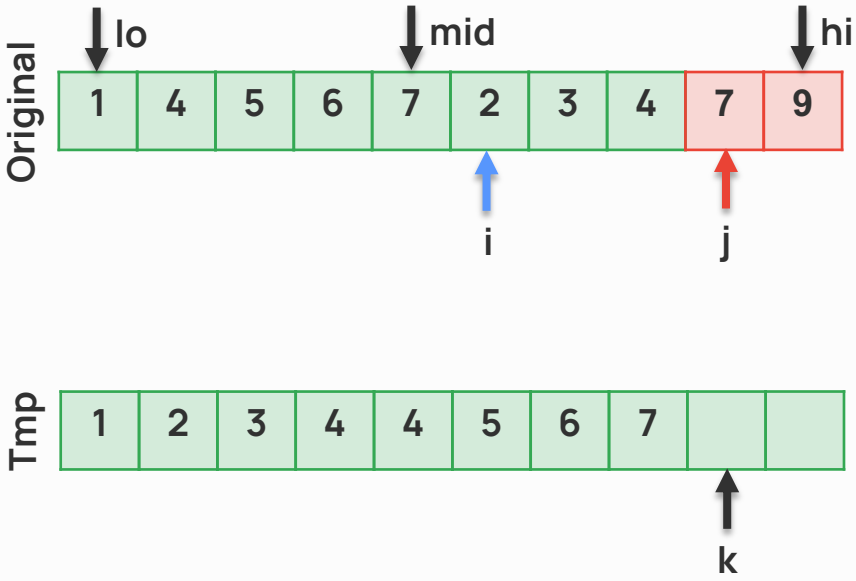
```
void merge(Item a[], int lo, int mid, int hi)
{
   int  i, j, k, nitems = hi-lo+1;
   Item *tmp = malloc(nitems*sizeof(Item));

   i = lo; j = mid+1; k = 0;
   // scan both segments, copying to tmp
   while (i <= mid && j <= hi) {
      if (less(a[i],a[j]))
         tmp[k++] = a[i++];
      else
         tmp[k++] = a[j++];
   }
   // copy items from unfinished segment
   while (i <= mid) tmp[k++] = a[i++];
   while (j <= hi) tmp[k++] = a[j++];

   //copy tmp back to main array
   for (i = lo, k = 0; i <= hi; i++, k++)
      a[i] = tmp[k];
   free(tmp);
}
```



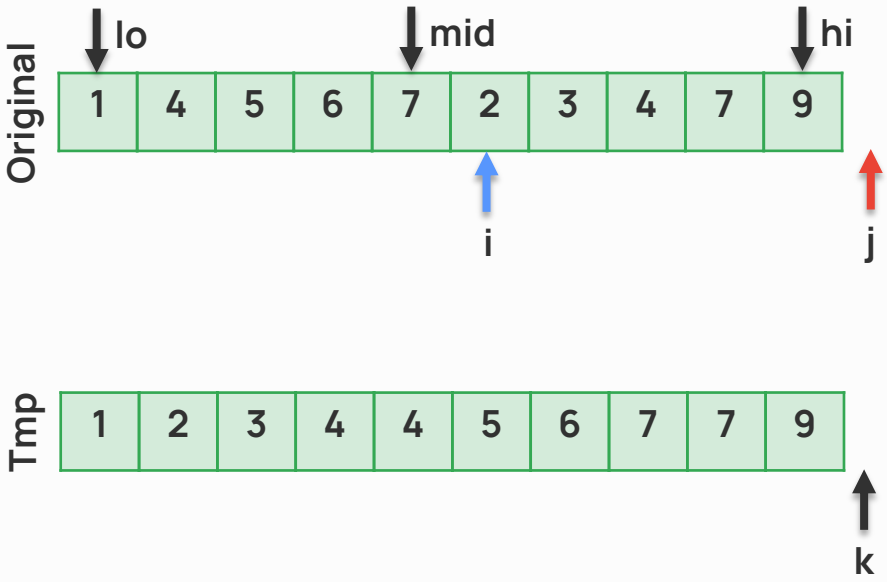
```
void merge(Item a[], int lo, int mid, int hi)
{
   int  i, j, k, nitems = hi-lo+1;
   Item *tmp = malloc(nitems*sizeof(Item));

   i = lo; j = mid+1; k = 0;
   // scan both segments, copying to tmp
   while (i <= mid && j <= hi) {
      if (less(a[i],a[j]))
         tmp[k++] = a[i++];
      else
         tmp[k++] = a[j++];
   }
   // copy items from unfinished segment
   while (i <= mid) tmp[k++] = a[i++];
   while (j <= hi) tmp[k++] = a[j++];

   //copy tmp back to main array
   for (i = lo, k = 0; i <= hi; i++, k++)
      a[i] = tmp[k];
   free(tmp);
}
```



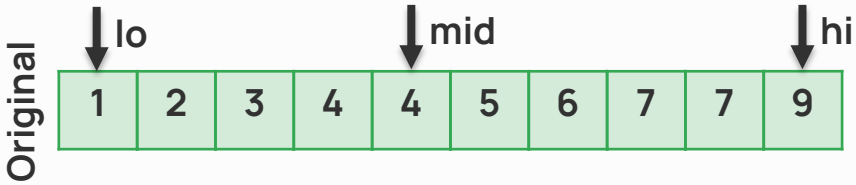
```
void merge(Item a[], int lo, int mid, int hi)
{
   int  i, j, k, nitems = hi-lo+1;
   Item *tmp = malloc(nitems*sizeof(Item));

   i = lo; j = mid+1; k = 0;
   // scan both segments, copying to tmp
   while (i <= mid && j <= hi) {
      if (less(a[i],a[j]))
         tmp[k++] = a[i++];
      else
         tmp[k++] = a[j++];
   }
   // copy items from unfinished segment
   while (i <= mid) tmp[k++] = a[i++];
   while (j <= hi) tmp[k++] = a[j++];

   //copy tmp back to main array
   for (i = lo, k = 0; i <= hi; i++, k++)
      a[i] = tmp[k];
   free(tmp);
}
```



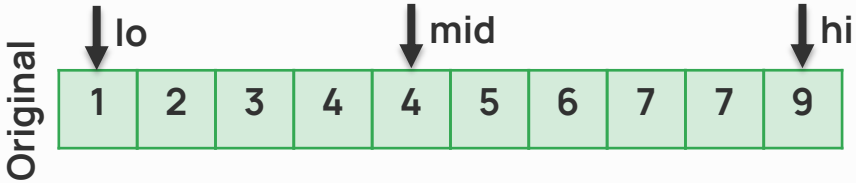
```
void merge(Item a[], int lo, int mid, int hi)
{
   int  i, j, k, nitems = hi-lo+1;
   Item *tmp = malloc(nitems*sizeof(Item));

   i = lo; j = mid+1; k = 0;
   // scan both segments, copying to tmp
   while (i <= mid && j <= hi) {
     if (less(a[i],a[j]))
        tmp[k++] = a[i++];
     else
        tmp[k++] = a[j++];
   }
   // copy items from unfinished segment
   while (i <= mid) tmp[k++] = a[i++];
   while (j <= hi) tmp[k++] = a[j++];

   //copy tmp back to main array
   for (i = lo, k = 0; i <= hi; i++, k++)
      a[i] = tmp[k];
   free(tmp);
}
```



```
void merge(Item a[], int lo, int mid, int hi)
{
   int  i, j, k, nitems = hi-lo+1;
   Item *tmp = malloc(nitems*sizeof(Item));

   i = lo; j = mid+1; k = 0;
   // scan both segments, copying to tmp
   while (i <= mid && j <= hi) {
     if (less(a[i],a[j]))
        tmp[k++] = a[i++];
     else
        tmp[k++] = a[j++];
   }
   // copy items from unfinished segment
   while (i <= mid) tmp[k++] = a[i++];
   while (j <= hi) tmp[k++] = a[j++];

   //copy tmp back to main array
   for (i = lo, k = 0; i <= hi; i++, k++)
      a[i] = tmp[k];
   free(tmp);
}
```



```
void merge(Item a[], int lo, int mid, int hi)
{
   int  i, j, k, nitems = hi-lo+1;
   Item *tmp = malloc(nitems*sizeof(Item));

   i = lo; j = mid+1; k = 0;
   // scan both segments, copying to tmp
   while (i <= mid && j <= hi) {
     if (less(a[i],a[j]))
        tmp[k++] = a[i++];
     else
        tmp[k++] = a[j++];
   }
   // copy items from unfinished segment
   while (i <= mid) tmp[k++] = a[i++];
   while (j <= hi) tmp[k++] = a[j++];

   //copy tmp back to main array
   for (i = lo, k = 0; i <= hi; i++, k++)
      a[i] = tmp[k];
   free(tmp);
}
```

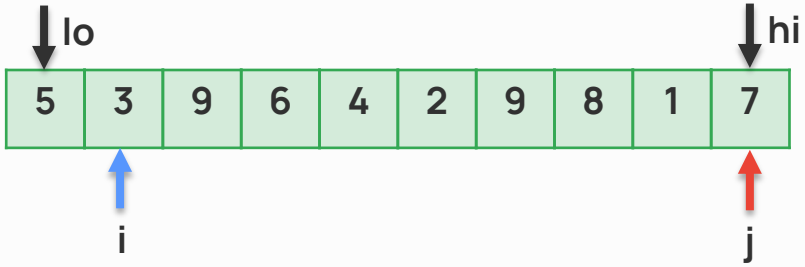


Quicksort

```
void quicksort(Item a[], int lo, int hi)
{
    int i; // index of pivot
    if (hi <= lo) return;
    i = partition(a, lo, hi);
    quicksort(a, lo, i-1);
    quicksort(a, i+1, hi);
}
```

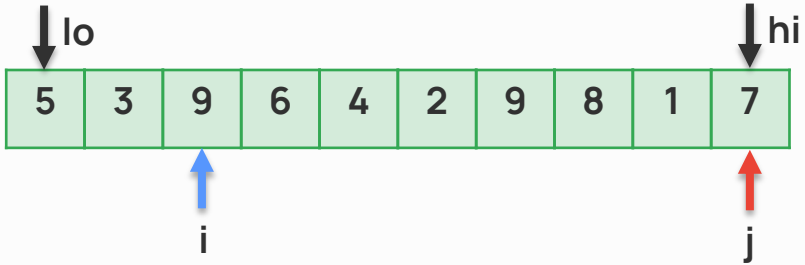
```
int partition(Item a[], int lo, int hi)
{
  Item v = a[lo];  // pivot
  int  i = lo+1, j = hi;
  for (;;) {
     while (less(a[i],v) && i < j) i++;
     while (less(v,a[j]) && j > i) j--;
     if (i == j) break;
     swap(a,i,j);
  }
  j = less(a[i],v) ? i : i-1;
  swap(a,lo,j);
  return j;
}
```

pivot = 5



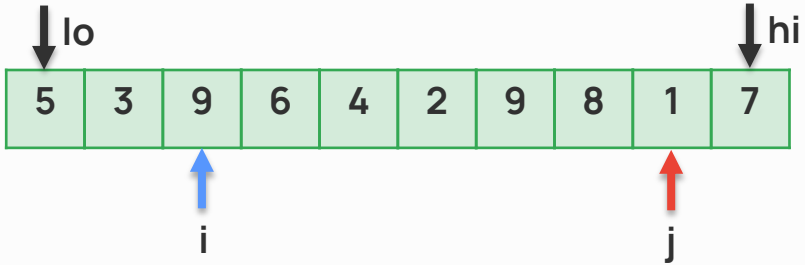
```
int partition(Item a[], int lo, int hi)
{
   Item v = a[lo];  // pivot
   int  i = lo+1, j = hi;
   for (;;) {
      while (less(a[i],v) && i < j) i++;
      while (less(v,a[j]) && j > i) j--;
      if (i == j) break;
      swap(a,i,j);
   }
   j = less(a[i],v) ? i : i-1;
   swap(a,lo,j);
   return j;
}
```

pivot = 5



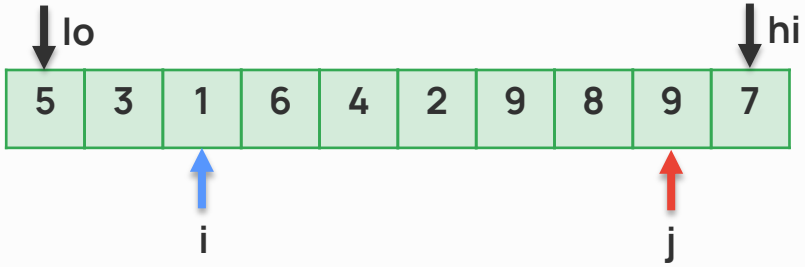
```
int partition(Item a[], int lo, int hi)
{
   Item v = a[lo];  // pivot
   int  i = lo+1, j = hi;
   for (;;) {
      while (less(a[i],v) && i < j) i++;
      while (less(v,a[j]) && j > i) j--;
      if (i == j) break;
      swap(a,i,j);
   }
   j = less(a[i],v) ? i : i-1;
   swap(a,lo,j);
   return j;
}
```

pivot = 5



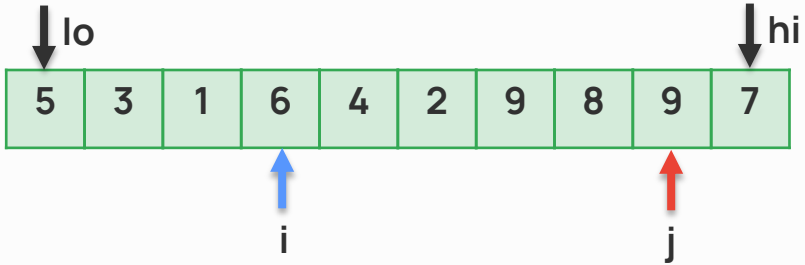
```
int partition(Item a[], int lo, int hi)
{
   Item v = a[lo];  // pivot
   int  i = lo+1, j = hi;
   for (;;) {
      while (less(a[i],v) && i < j) i++;
      while (less(v,a[j]) && j > i) j--;
      if (i == j) break;
      swap(a,i,j);
   }
   j = less(a[i],v) ? i : i-1;
   swap(a,lo,j);
   return j;
}
```

pivot = 5



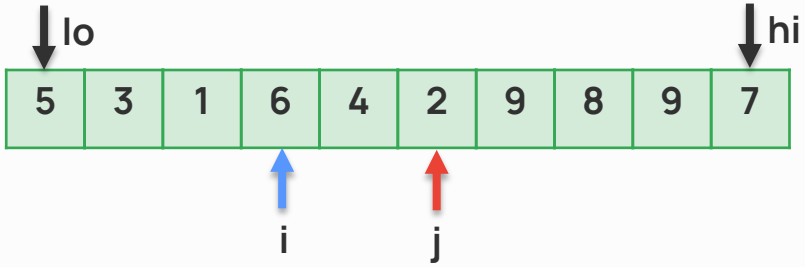
```
int partition(Item a[], int lo, int hi)
{
   Item v = a[lo];  // pivot
   int  i = lo+1, j = hi;
   for (;;) {
      while (less(a[i],v) && i < j) i++;
      while (less(v,a[j]) && j > i) j--;
      if (i == j) break;
      swap(a,i,j);
   }
   j = less(a[i],v) ? i : i-1;
   swap(a,lo,j);
   return j;
}
```

pivot = 5



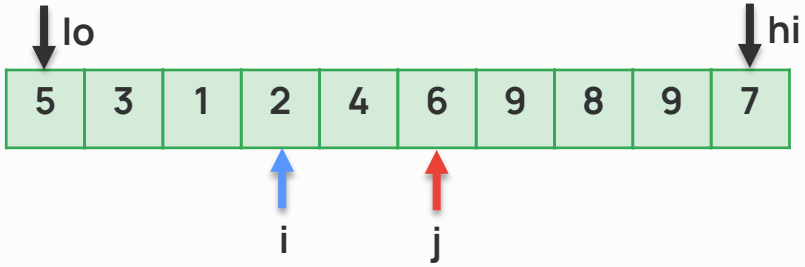
```
int partition(Item a[], int lo, int hi)
{
   Item v = a[lo];  // pivot
   int  i = lo+1, j = hi;
   for (;;) {
      while (less(a[i],v) && i < j) i++;
      while (less(v,a[j]) && j > i) j--;
      if (i == j) break;
      swap(a,i,j);
   }
   j = less(a[i],v) ? i : i-1;
   swap(a,lo,j);
   return j;
}
```

pivot = 5



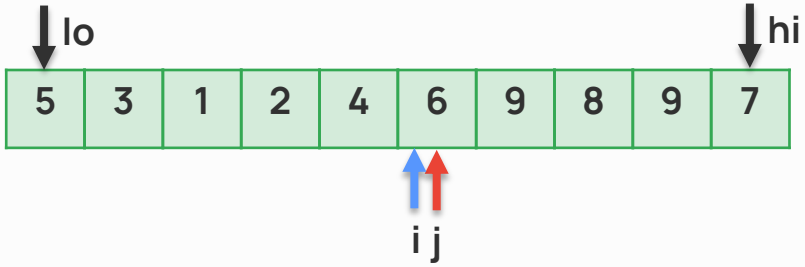
```
int partition(Item a[], int lo, int hi)
{
   Item v = a[lo];  // pivot
   int  i = lo+1, j = hi;
   for (;;) {
      while (less(a[i],v) && i < j) i++;
      while (less(v,a[j]) && j > i) j--;
      if (i == j) break;
      swap(a,i,j);
   }
   j = less(a[i],v) ? i : i-1;
   swap(a,lo,j);
   return j;
}
```

pivot = 5



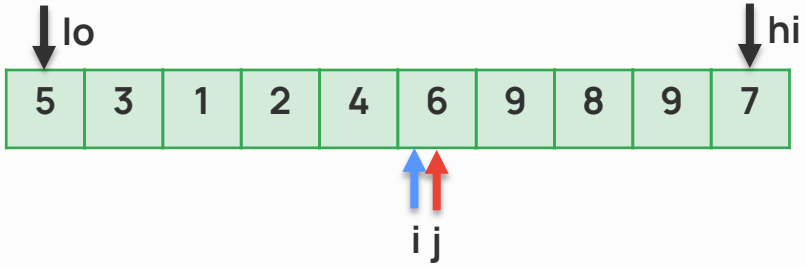
```
int partition(Item a[], int lo, int hi)
{
   Item v = a[lo];  // pivot
   int  i = lo+1, j = hi;
   for (;;) {
      while (less(a[i],v) && i < j) i++;
      while (less(v,a[j]) && j > i) j--;
      if (i == j) break;
      swap(a,i,j);
   }
   j = less(a[i],v) ? i : i-1;
   swap(a,lo,j);
   return j;
}
```

pivot = 5



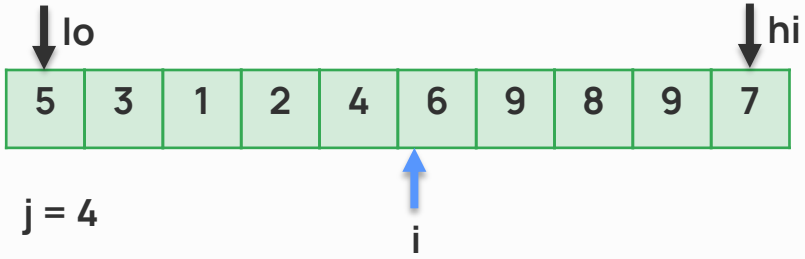
```
int partition(Item a[], int lo, int hi)
{
   Item v = a[lo];  // pivot
   int  i = lo+1, j = hi;
   for (;;) {
      while (less(a[i],v) && i < j) i++;
      while (less(v,a[j]) && j > i) j--;
      if (i == j) break;
      swap(a,i,j);
   }
   j = less(a[i],v) ? i : i-1;
   swap(a,lo,j);
   return j;
}
```

pivot = 5



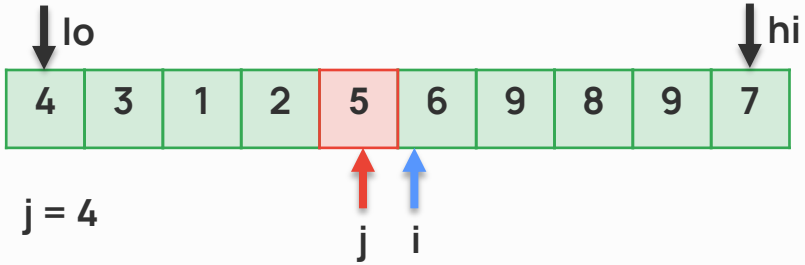
```
int partition(Item a[], int lo, int hi)
{
   Item v = a[lo];  // pivot
   int  i = lo+1, j = hi;
   for (;;) {
      while (less(a[i],v) && i < j) i++;
      while (less(v,a[j]) && j > i) j--;
      if (i == j) break;
      swap(a,i,j);
   }
   j = less(a[i],v) ? i : i-1;
   swap(a,lo,j);
   return j;
}
```

pivot = 5

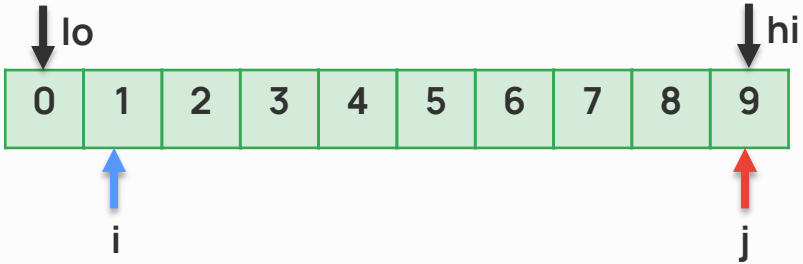


```
int partition(Item a[], int lo, int hi)
{
   Item v = a[lo];  // pivot
   int  i = lo+1, j = hi;
   for (;;) {
      while (less(a[i],v) && i < j) i++;
      while (less(v,a[j]) && j > i) j--;
      if (i == j) break;
      swap(a,i,j);
   }
   j = less(a[i],v) ? i : i-1;
   swap(a,lo,j);
   return j;
}
```

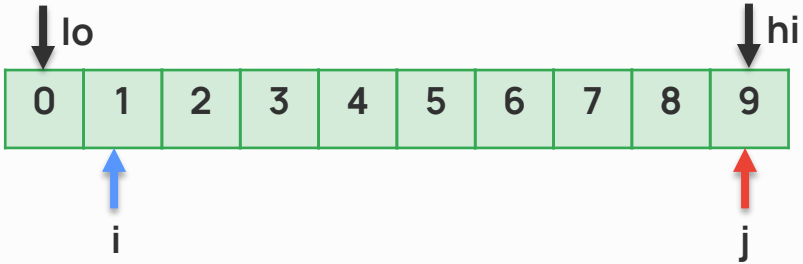
pivot = 5

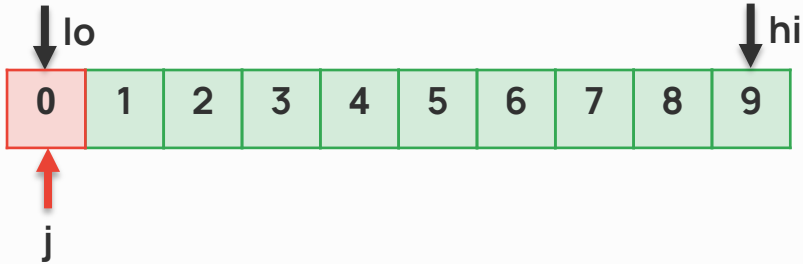


pivot = 0

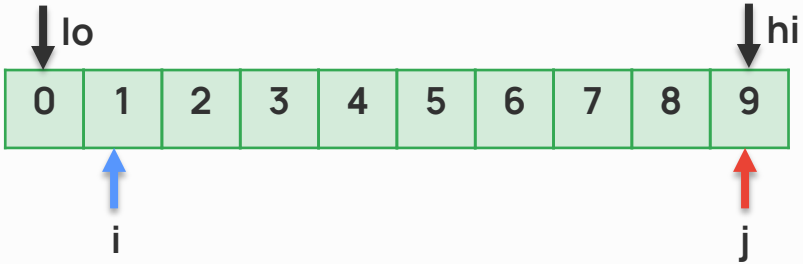


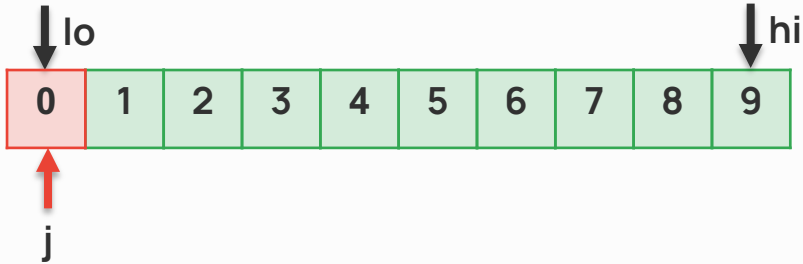
pivot = 0



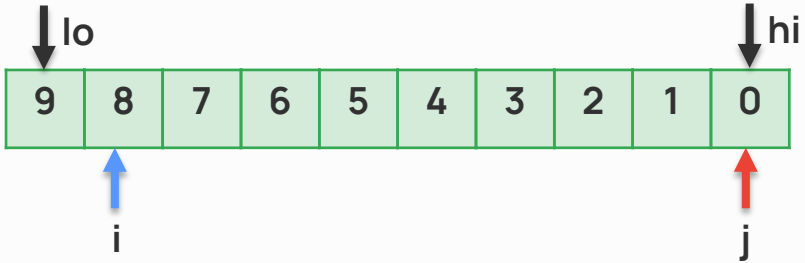


pivot = 0

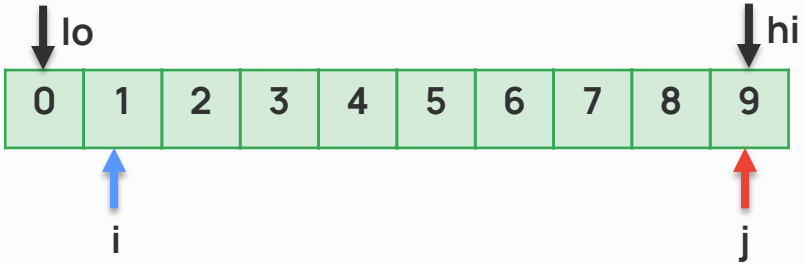


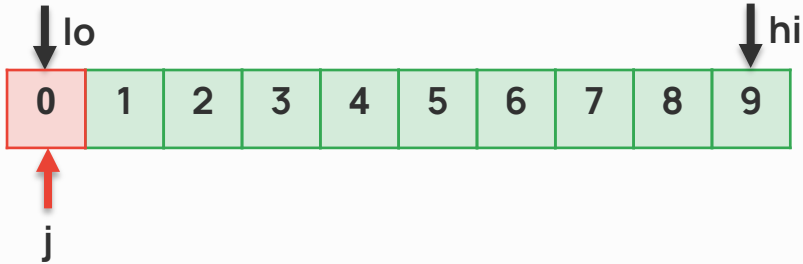


pivot = 9



pivot = 0





pivot = 9