Unit-IV

Sorting

Sorting by Insertion

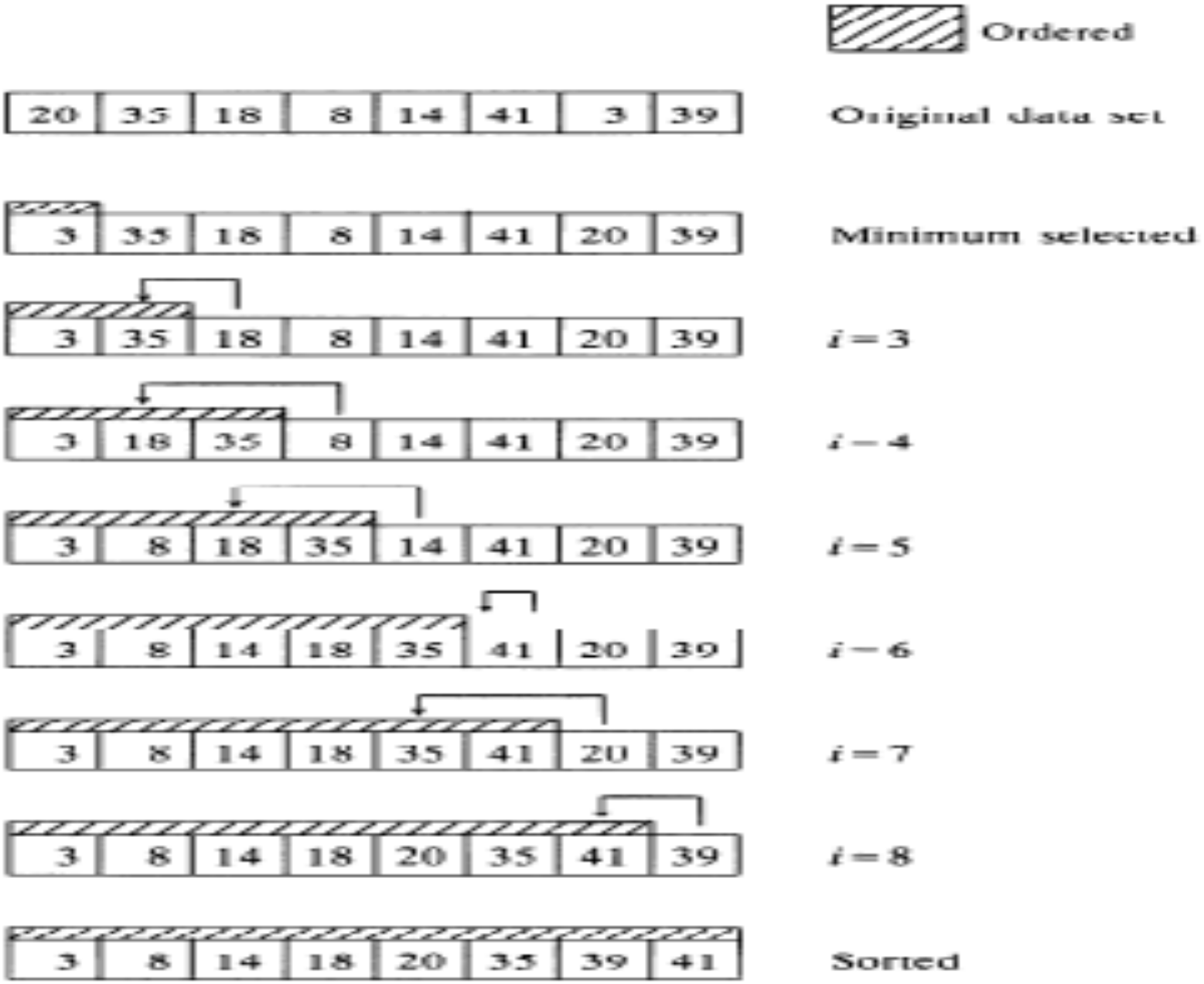
- It initially sorts the first two members of the array. Next, the algorithm inserts the third member into its sorted position in relation to the first two members. Then it inserts the fourth element into the list of three elements.
- The process continues until all elements have been sorted.

Algorithm description

1. Establish the array $a[1..n]$ of n elements.
2. Find the minimum and put it in place to act as sentinel.

3. While there are still elements to be inserted in the ordered part do
   (a) select next element x to be inserted;
   (b) while x is less than preceding element do
      (b.1) move preceding element up one position,
      (b.2) extend search back one element further;
   (c) insert x at current position.

```
/* The Insertion Sort. */
void insert (int *a, int count)
{
 int i, b;
int t;
for(i=1; i < count; ++i) {
t = a[i];
for(b=i-1; (b >= 0) && (t < a[b]); b--)
a[b+1] = a[b];
a[b+1] = t;
}
}
```


Ordered
20 35 18 8 14 41 3 39
Original data set
3 35 18 8 14 41 20 39
Minimum selected
3 35 18 8 14 41 20 39
i = 3
3 18 35 8 14 41 20 39
i = 4
3 8 18 35 14 41 20 39
i = 5
3 8 14 18 35 41 20 39
i = 6
3 8 14 18 35 41 20 39
i = 7
3 8 14 18 20 35 41 39
i = 8
3 8 14 18 20 35 39 41
Sorted

The Quicksort

- The quicksort, invented and named by C. A. R. Hoare, is superior to all others.
- Generally considered the best general purpose sorting algorithm currently available.
- It is based on the exchange sort

- The quicksort is built on the idea of partitions.
- The general procedure is to select a value, called the *comparand, and then to partition the array into two sections.*
- *All elements greater than or equal to* the partition value are put on one side, and those less than the value are put on the other.
- This process is then repeated for each remaining section until the array is sorted.

Algorithm description

1. Establish array $a[1..n]$ to be sorted.
2. Place upper and lower limits for array on the stack and initialize pointer to top of stack.
3. While stack is not empty do
   (a) remove upper and lower limits of array segment from top of stack;
   (b) while current segment not reduced to size 1 do
      (b.1) select middle element of array segment from stack;
      (b.2) partition the current segment into two with respect to the current middle value;*
      (b.3) save the limits of the larger partition on the stack of later processing and do setup to process the smaller partition next if it contains more than one element.

```
/* Quicksort setup function. */
void quick(int *a, int count)
{
qs(a, 0, count-1);
}
/* The Quicksort. */
void qs(int*a, int left, int right)
{
 int i, j;
int  x, y;
i = left; j = right;
x = a[(left+right)/2];
do {
while((a[i] < x) && (i < right)) i++;
while((x < a[j]) && (j > left)) j--;
if(i <= j) {
y = a[i];
a[i] =a[j];
a[j] = y;
i++; j--;
}
} while(i <= j);
if(left < j) qs(a, left, j);
if(i < right) qs(a, i, right);
}
```