Simple sorting algorithms and their complexity

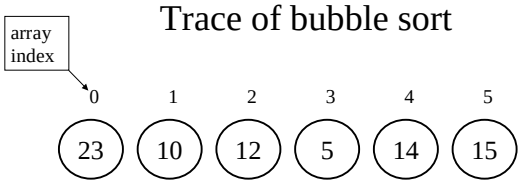
- Bubble sort
- Selection sort
- Insertion sort

Bubble sort

```
void  bubbleSort(int arr[]){
   int i;
   int j;
   int temp;
   for(i = arr.length-1; i > 0; i--){
      for(j = 0; j < i; j++){
         if(arr[j] > arr[j+1]){
            temp = arr[j];
            arr[j] = arr[j+1];
            arr[j+1] = temp;
         }//
      }// end inner loop
   }//end outer loop}// end bubble sort
```

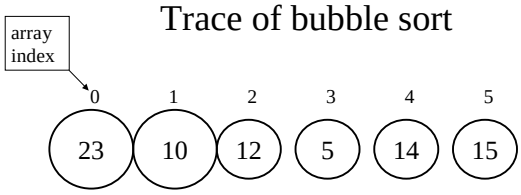
swap adjacent elements, if in the wrong order

Trace of bubble sort



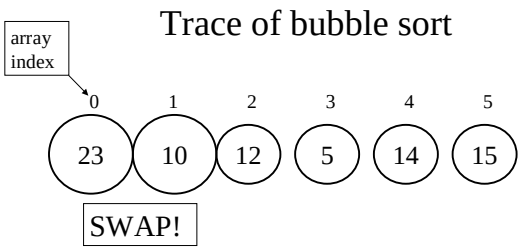
i = 5, first iteration of the outer loop

Trace of bubble sort



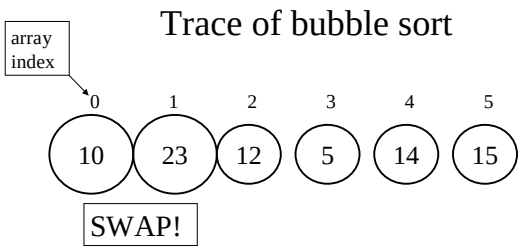
i = 5, first iteration of the outer loop
j = 0, comparing arr[0] and arr[1]

Trace of bubble sort



i = 5, first iteration of the outer loop
j = 0, comparing arr[0] and arr[1]

Trace of bubble sort



i = 5, first iteration of the outer loop
j = 0, comparing arr[0] and arr[1]

Trace of bubble sort

array index

0	1	2	3	4	5
10	23	12	5	14	15

i = 5, first iteration of the outer loop
j = 1, comparing arr[1] and arr[2]

Trace of bubble sort

array index

0	1	2	3	4	5
10	23	12	5	14	15

SWAP!

i = 5, first iteration of the outer loop
j = 1, comparing arr[1] and arr[2]

Trace of bubble sort

array index

0	1	2	3	4	5
10	12	23	5	14	15

SWAP!

i = 5, first iteration of the outer loop
j = 1, comparing arr[1] and arr[2]

Trace of bubble sort

array index

0	1	2	3	4	5
10	12	23	5	14	15

i = 5, first iteration of the outer loop
j = 2, comparing arr[2] and arr[3]

Trace of bubble sort

array index

0	1	2	3	4	5
10	12	23	5	14	15

SWAP!

i = 5, first iteration of the outer loop
j = 2, comparing arr[2] and arr[3]

Trace of bubble sort

array index

0	1	2	3	4	5
10	12	5	23	14	15

SWAP!

i = 5, first iteration of the outer loop
j = 2, comparing arr[2] and arr[3]

Trace of bubble sort

array index

0	1	2	3	4	5
10	12	5	23	14	15

i = 5, first iteration of the outer loop
j = 3, comparing arr[3] and arr[4]

Trace of bubble sort

array index

0	1	2	3	4	5
10	12	5	23	14	15

SWAP!

i = 5, first iteration of the outer loop
j = 3, comparing arr[3] and arr[4]

Trace of bubble sort

array index

0	1	2	3	4	5
10	12	5	14	23	15

SWAP!

i = 5, first iteration of the outer loop
j = 3, comparing arr[3] and arr[4]

Trace of bubble sort

array index

0	1	2	3	4	5
10	12	5	14	23	15

i = 5, first iteration of the outer loop
j = 4, comparing arr[4] and arr[5]

Trace of bubble sort

array index

0	1	2	3	4	5
10	12	5	14	23	15

SWAP!

i = 5, first iteration of the outer loop
j = 4, comparing arr[4] and arr[5]

Trace of bubble sort

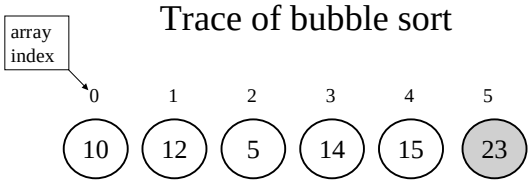
array index

0	1	2	3	4	5
10	12	5	14	15	23

SWAP!

i = 5, first iteration of the outer loop
j = 4, comparing arr[4] and arr[5]

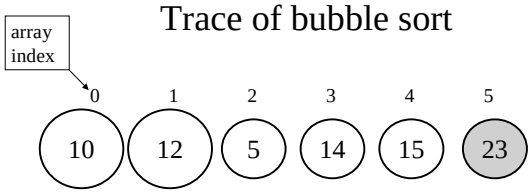
Trace of bubble sort



i = 5, first iteration of the outer loop

inner loop finished; largest element in position 5, positions 0-4 unsorted

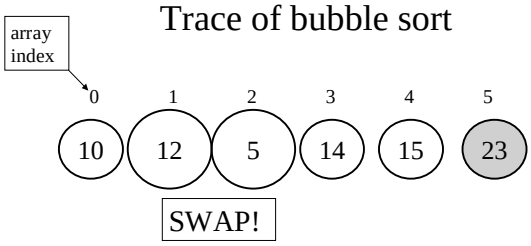
Trace of bubble sort



i = 4, second iteration of the outer loop

j = 0, comparing arr[0] with arr[1]

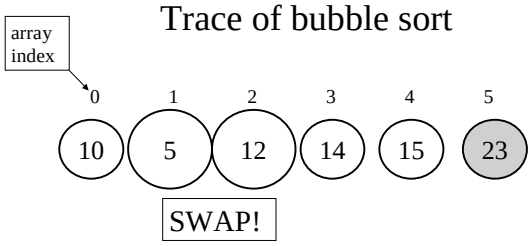
Trace of bubble sort



i = 4, second iteration of the outer loop

j = 1, comparing arr[1] with arr[2]

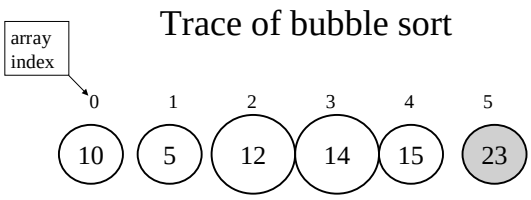
Trace of bubble sort



i = 4, second iteration of the outer loop

j = 1, comparing arr[1] with arr[2]

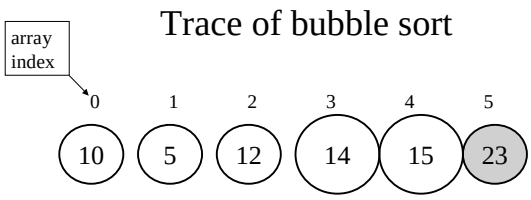
Trace of bubble sort



i = 4, second iteration of the outer loop

j = 2, comparing arr[2] with arr[3]

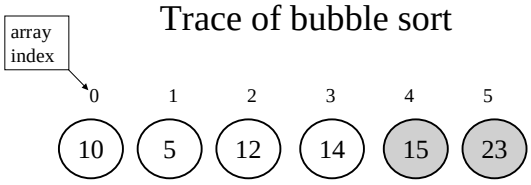
Trace of bubble sort



i = 4, second iteration of the outer loop

j = 3, comparing arr[3] with arr[4]

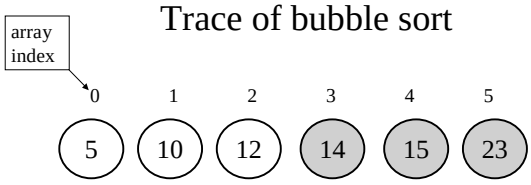
Trace of bubble sort



i = 4, second iteration of the outer loop

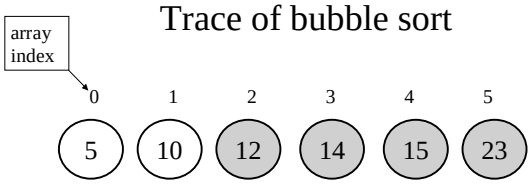
inner loop finished, second largest element in position 4, positions 0-3 unsorted

Trace of bubble sort



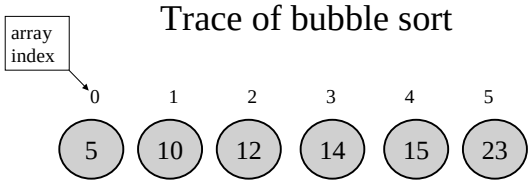
After third iteration…

Trace of bubble sort



After fourth iteration…

Trace of bubble sort



After fifth iteration…

Complexity of bubble sort

- For an array of size n, in the worst case:
  1st passage through the inner loop: n-1 comparisons and n-1 swaps
- ...
- (n-1)st passage through the inner loop: one comparison and one swap.
- All together: c ((n-1) + (n-2) + ... + 1), where c is the time required to do one comparison, one swap, check the inner loop condition and increment j.
- We also spend constant time k declaring i,j,temp and initialising i. Outer loop is executed n-1 times, suppose the cost of checking the loop condition and decrementing i is c_1.

Complexity of bubble sort

$c\,((n-1) + (n-2) + \ldots + 1) + k + c_1(n-1)$

$(n-1) + (n-2) + \ldots + 1 = n(n-1)/2$

so our function equals

$c\,n*(n-1)/2 + k + c_1(n-1) = 1/2c\,(n^2-n) + c(n-1) + k$

complexity $O(n^2)$.

Complexity of bubble sort

Need to find n_0 and K, such that for all $n \geq n_0$,
$1/2c\,(n^2-n) + c_1(n-1) + k \leq K^* n^2$

$1/2c\, n^2 - 1/2c\, n + c_1 n - c_1 + k \leq$
$1/2c\, n^2 + c_1 n + k \leq$
$c\, n^2 + c_1\, n^2 + k\, n^2$
Take $K = c + c_1 + k$ and $n_0 = 1$.

Selection sort

```
void  selectionSort(int arr[]){
  int i, j, temp, pos_greatest;
  for( i = arr.length-1; i > 0; i--){
     pos_greatest = 0;
     for(j = 0; j <= i; j++){
       if( arr[j] > arr[pos_greatest])
          pos_greatest =  j;
     }//end inner for loop
     temp = arr[i];
     arr[i] = arr[pos_greatest];
     arr[pos_greatest] = temp;
}//end outer for loop}//end selection sort
```

compare the current element to the largest seen so far; if it is larger, remember its index

swap the largest element to the end of range

Trace of selection sort

array index

0	1	2	3	4	5
23	10	12	5	14	15

i = 5, first iteration of the outer loop

Trace of selection sort

array index

0	1	2	3	4	5
23	10	12	5	14	15

j

i = 5, first iteration of the outer loop

j = 0, pos_greatest = 0

Trace of selection sort

array index

0	1	2	3	4	5
23	10	12	5	14	15

j

i = 5, first iteration of the outer loop

j = 1, pos_greatest = 0

Trace of selection sort

array index

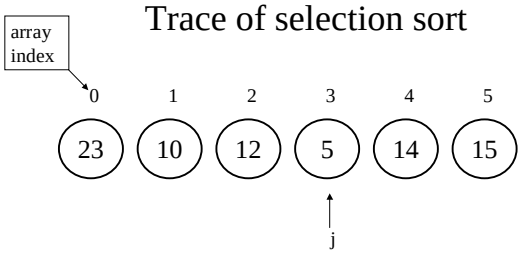
0	1	2	3	4	5
23	10	12	5	14	15

j

i = 5, first iteration of the outer loop

j = 2, pos_greatest = 0

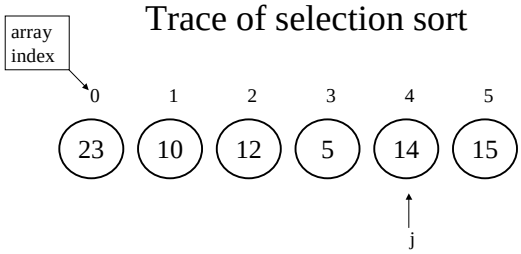
Trace of selection sort



i = 5, first iteration of the outer loop

j = 3, pos_greatest = 0

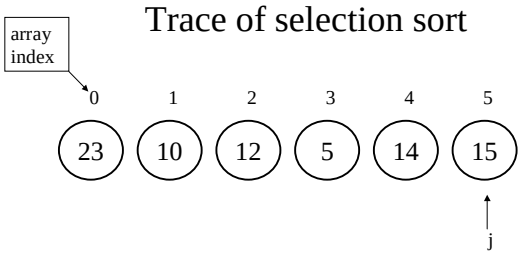
Trace of selection sort



i = 5, first iteration of the outer loop

j = 4, pos_greatest = 0

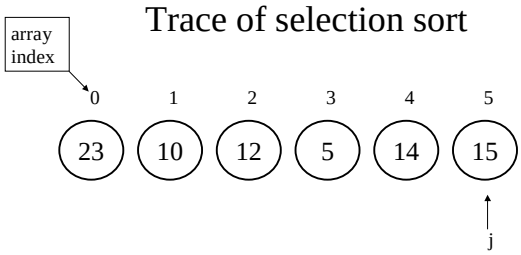
Trace of selection sort



i = 5, first iteration of the outer loop

j = 5, pos_greatest = 0

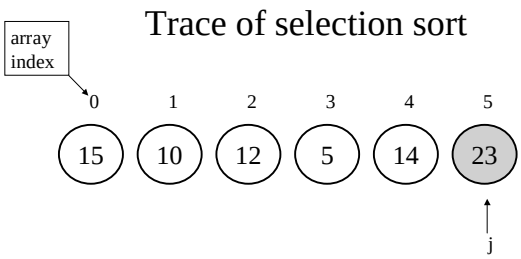
Trace of selection sort



i = 5, first iteration of the outer loop

swap element at pos_greatest to 5

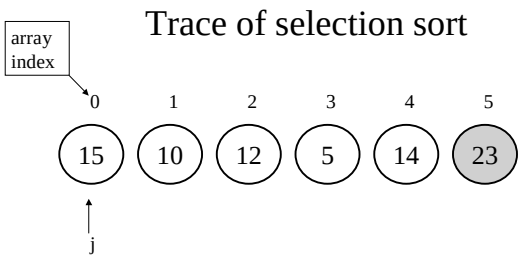
Trace of selection sort



i = 5, first iteration of the outer loop

swap element at pos_greatest to 5

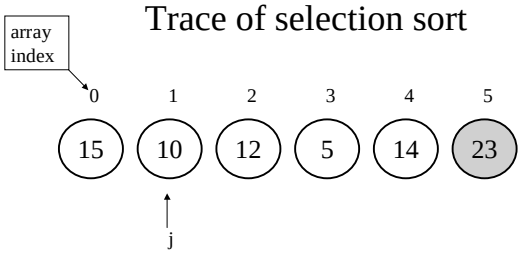
Trace of selection sort



i = 4, second iteration of the outer loop

j = 0, pos_greatest = 0

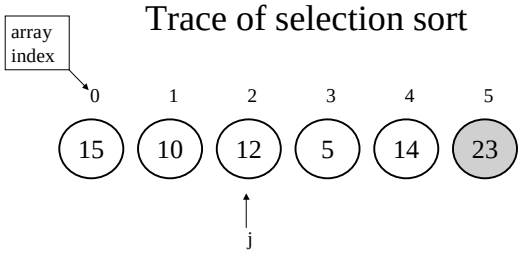
Trace of selection sort



i = 4, second iteration of the outer loop

j = 1, pos_greatest = 0

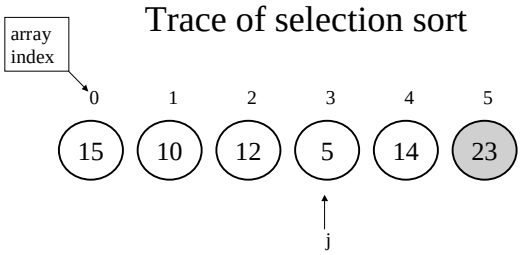
Trace of selection sort



i = 4, second iteration of the outer loop

j = 2, pos_greatest = 0

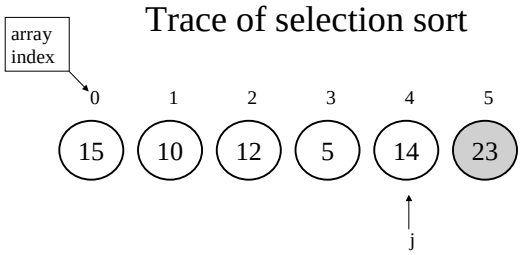
Trace of selection sort



i = 4, second iteration of the outer loop

j = 3, pos_greatest = 0

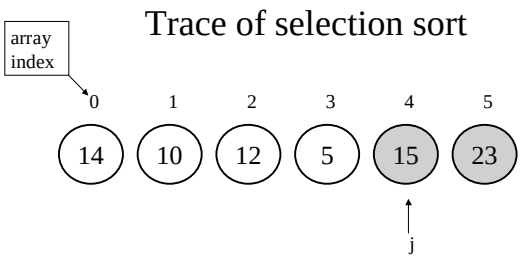
Trace of selection sort



i = 4, second iteration of the outer loop

j = 4, pos_greatest = 0

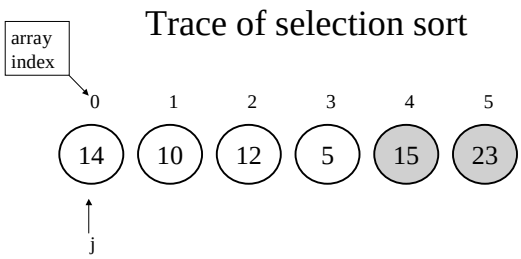
Trace of selection sort



i = 4, second iteration of the outer loop

Swap element at pos_greatest and 4

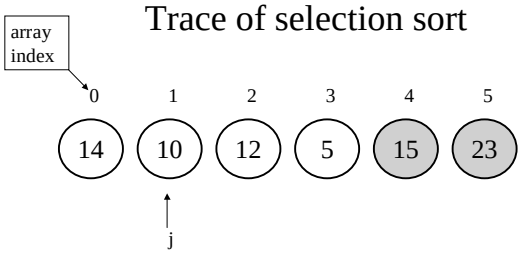
Trace of selection sort



i = 3, third iteration of the outer loop

j = 0, pos_greatest = 0

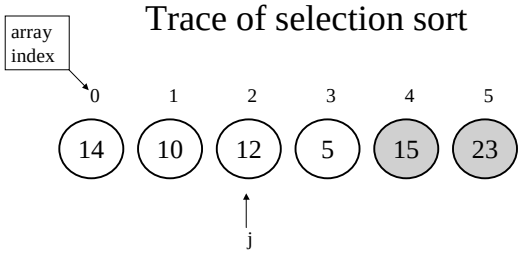
Trace of selection sort



i = 3, third iteration of the outer loop

j = 1, pos_greatest = 0

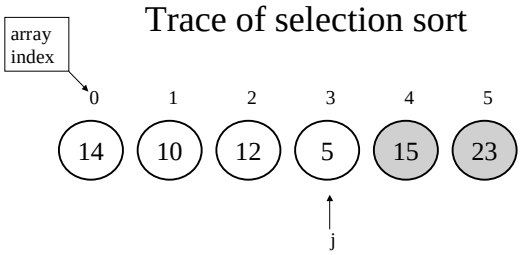
Trace of selection sort



i = 3, third iteration of the outer loop

j = 2, pos_greatest = 0

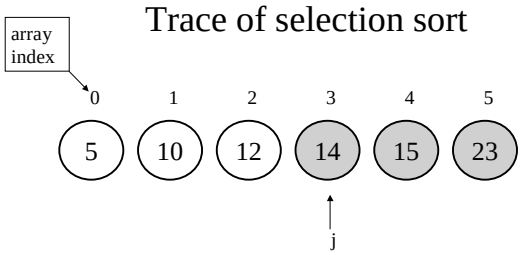
Trace of selection sort



i = 3, third iteration of the outer loop

j = 3, pos_greatest = 0

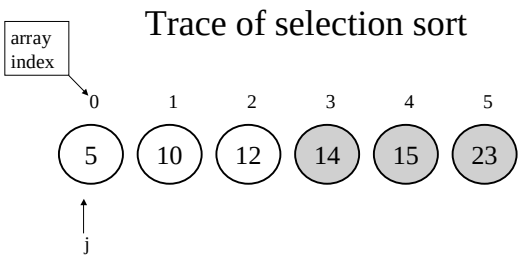
Trace of selection sort



i = 3, third iteration of the outer loop

swap elements at pos_greatest and 3

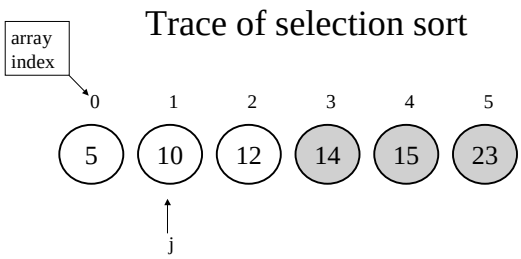
Trace of selection sort



i = 2, fourth iteration of the outer loop

j = 0, pos_greatest = 0

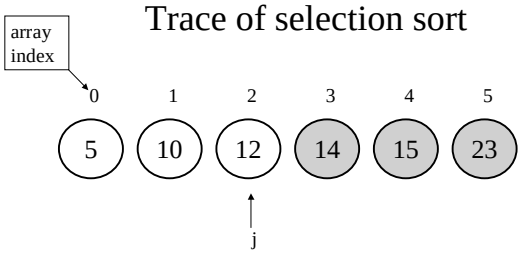
Trace of selection sort



i = 2, fourth iteration of the outer loop

j = 1, pos_greatest = 1 (changed!)

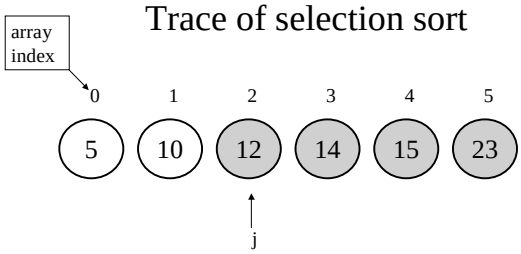
Trace of selection sort



i = 2, fourth iteration of the outer loop

j = 2, pos_greatest = 2 (changed again!)

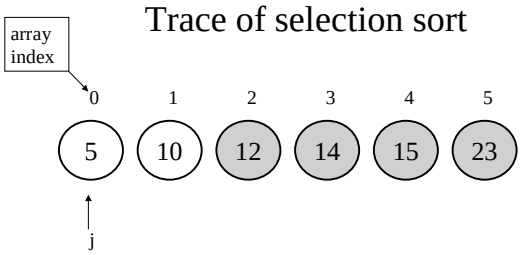
Trace of selection sort



i = 2, fourth iteration of the outer loop

swap elements at pos_greatest and 2 (element 12 with itself…)

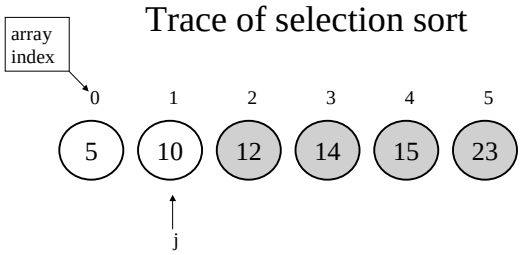
Trace of selection sort



i = 1, fifth iteration of the outer loop

j = 0, pos_greatest = 0

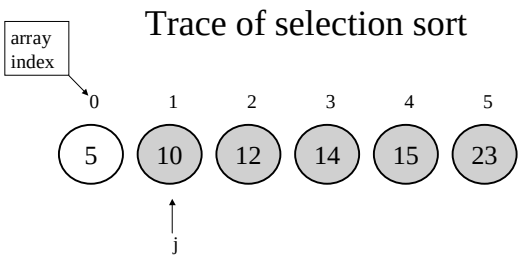
Trace of selection sort



i = 1, fifth iteration of the outer loop

j = 1, pos_greatest = 1 (changed)

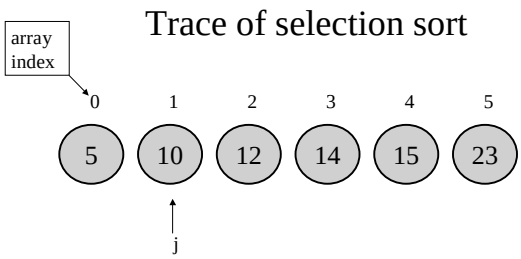
Trace of selection sort



i = 1, fifth iteration of the outer loop

swap element at pos_greatest with element at position 1 (10 with itself)

Trace of selection sort



i = 1, fifth iteration of the outer loop

done

Complexity of selection sort

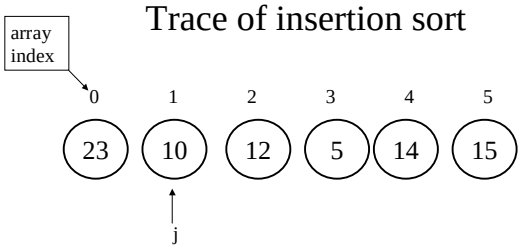
- Same number of iterations
- Same number of comparisons in the worst case
- fewer swaps (one for each outer loop = n-1)
- also $O(n^2)$

Insertion sort

```
void insertionSort(int arr[]){
  int i,j,temp;
  for(j=1; j < arr.length-1; j++){
    temp = arr[j];
    i = j; // range 0 to j-1 is sorted
    while(i > 0 && arr[i-1] >= temp){
      arr[i] = arr[i-1];
      i--;
    }
    arr[i] = temp;
  } // end outer for loop
}  // end insertion sort
```

Find a place to insert temp in the sorted range; as you are looking, shift elements in the sorted range to the right

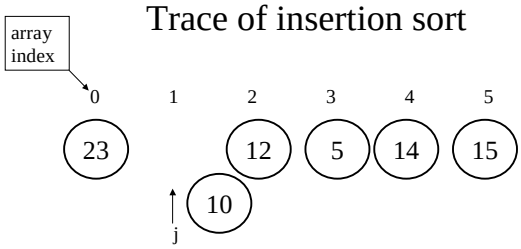
Trace of insertion sort



j = 1, first iteration of the outer loop

temp = 10; i = 1; arr[i-1] >= 10

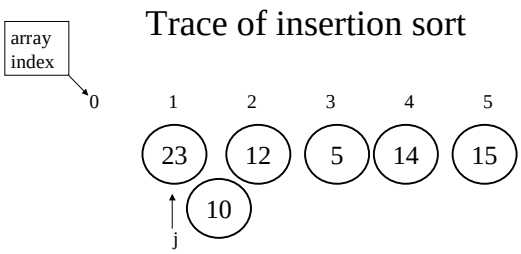
Trace of insertion sort



j = 1, first iteration of the outer loop

temp = 10; i = 1; arr[i-1] >= 10

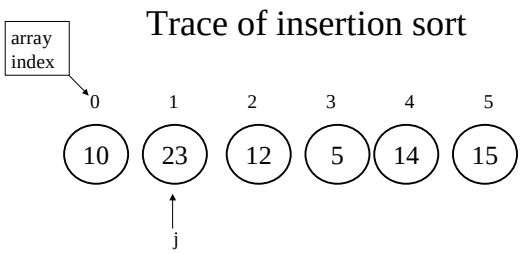
Trace of insertion sort



j = 1, first iteration of the outer loop

arr[i] = arr[i-1]

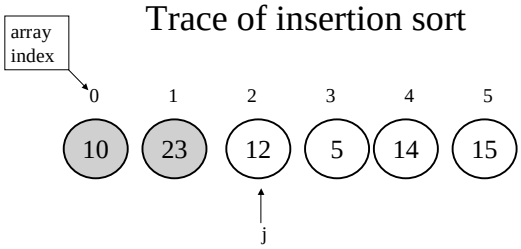
Trace of insertion sort



j = 1, first iteration of the outer loop

arr[i] = temp

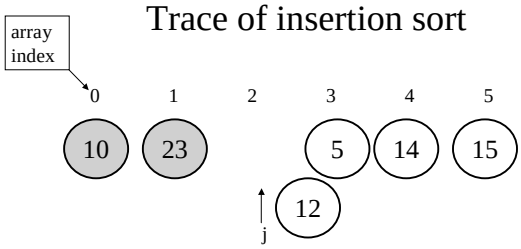
Trace of insertion sort



j = 2, second iteration of the outer loop

temp = 12; arr[i-1] >= temp

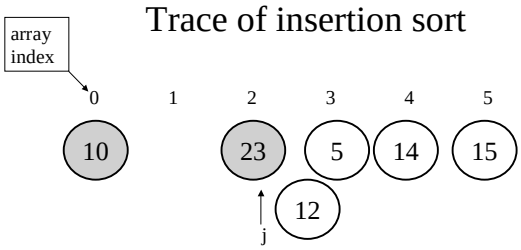
Trace of insertion sort



j = 2, second iteration of the outer loop

temp = 12; arr[i-1] >= temp

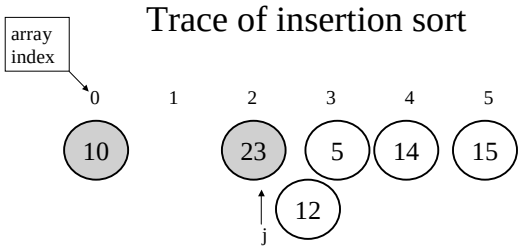
Trace of insertion sort



j = 2, second iteration of the outer loop

arr[i-1] = arr[i]

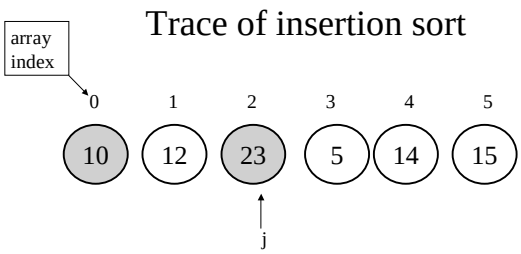
Trace of insertion sort



j = 2, second iteration of the outer loop

arr[i-1] < temp

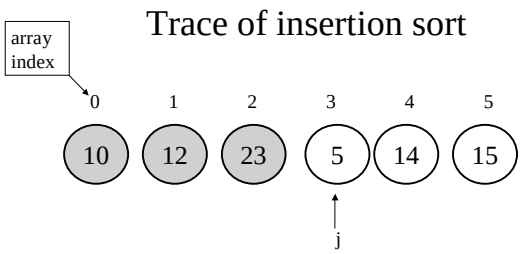
Trace of insertion sort



j = 2, second iteration of the outer loop

arr[i-1] = temp

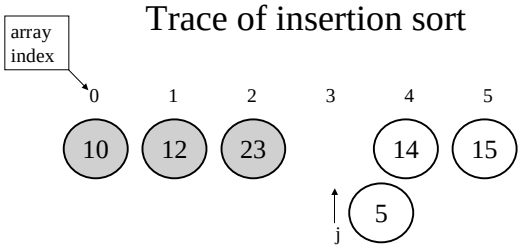
Trace of insertion sort



j = 3, third iteration of the outer loop

temp = 5

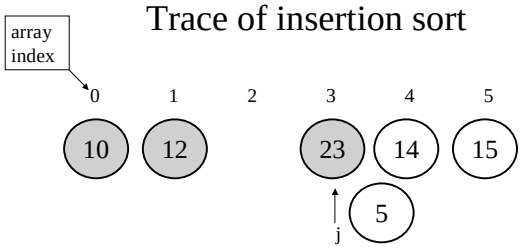
Trace of insertion sort



j = 3, third iteration of the outer loop

arr[i-1] >= temp

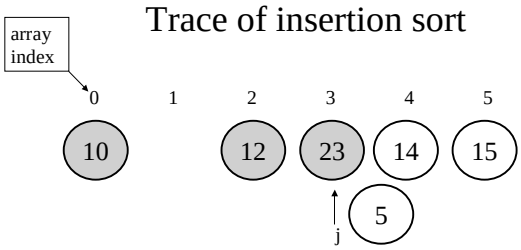
Trace of insertion sort



j = 3, third iteration of the outer loop

arr[i-1] >= temp

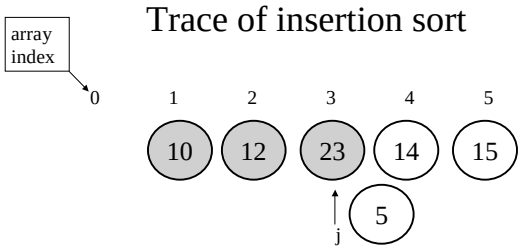
Trace of insertion sort



j = 3, third iteration of the outer loop

arr[i-1] >= temp

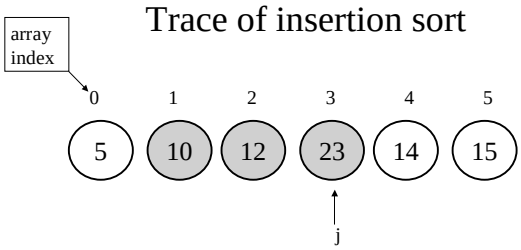
Trace of insertion sort



j = 3, third iteration of the outer loop

arr[i-1] >= temp

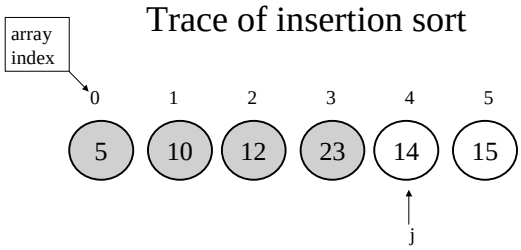
Trace of insertion sort



j = 3, third iteration of the outer loop

arr[i-1] = temp

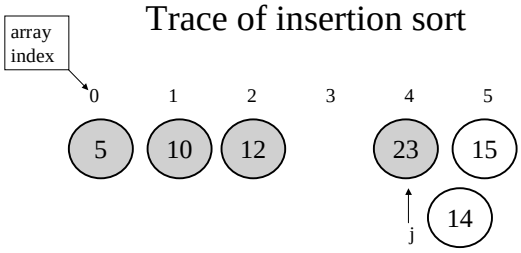
Trace of insertion sort



j = 4, fourth iteration of the outer loop

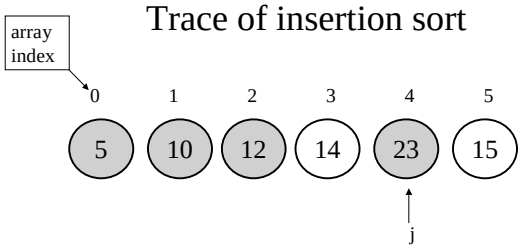
temp = 14

Trace of insertion sort



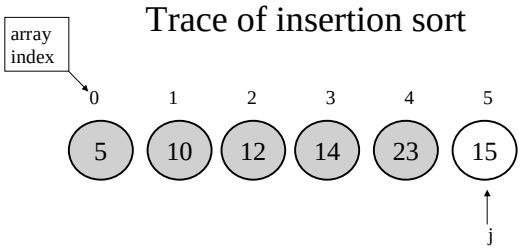
j = 4, fourth iteration of the outer loop
arr[i-1] >= temp

Trace of insertion sort



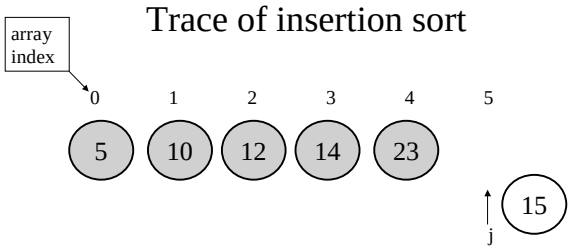
j = 4, fourth iteration of the outer loop
arr[i-1] = temp

Trace of insertion sort



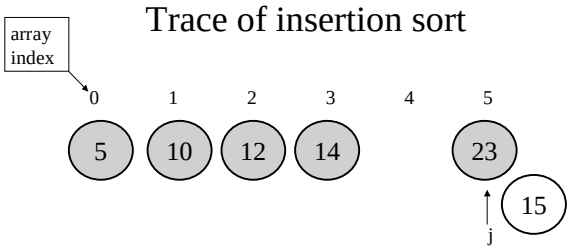
j = 5, fifth iteration of the outer loop
temp = 15

Trace of insertion sort



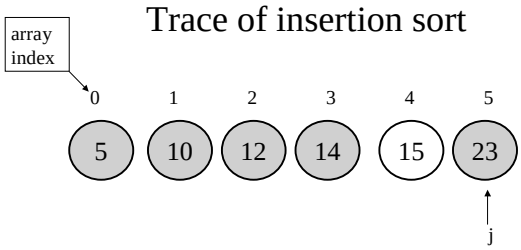
j = 5, fifth iteration of the outer loop
arr[i-1] >= temp

Trace of insertion sort



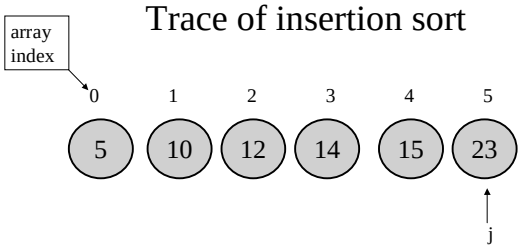
j = 5, fifth iteration of the outer loop
arr[i-1] >= temp

Trace of insertion sort



j = 5, fifth iteration of the outer loop
arr[i-1] = temp

Trace of insertion sort



j = 5, fifth iteration of the outer loop

arr[i-1] = temp

Complexity of insertion sort

- In the worst case, has to make n(n-1)/2 comparisons and shifts to the right
- also $O(n^2)$ worst case complexity
- best case: array already sorted, no shifts.

Reading and informal coursework

- Shaffer, Chapter 8.1, 8.2
- Informal coursework: prove that the simple sorting algorithms above are not in O(N).