Sort Algorithms

Efficiency and Analysis

Analysis of Simple Sorting Algorithms

- **Selection sort**
- **Bubble sort**

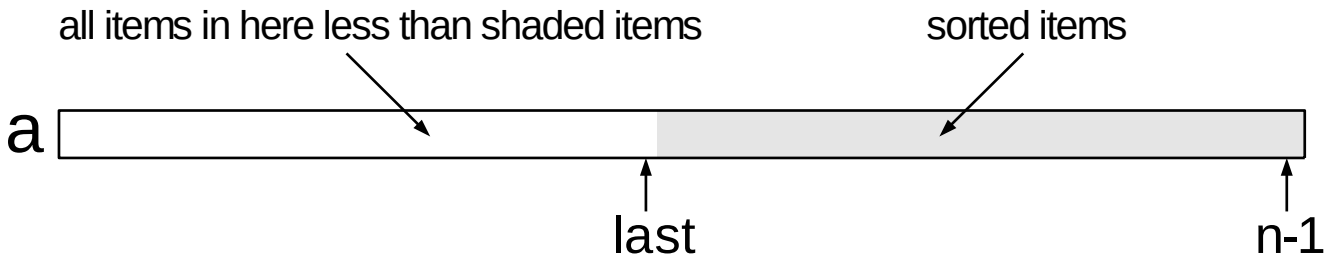
Analysis of selection sort

- Assume n element list
- Makes n-1 passes through the list
- Each pass has a sequential search of some portion of the n elements
- Analysis: $n * n = O(n^2)$

The maxSelect function

```
int maxSelect(int a[], int n)
{
   int maxPos(0), currentPos(1);
   while (currentPos < n) {
   // Invariant: a[maxPos] >= a[0] ... a[currentPos-1]
      if (a[currentPos] > a[maxPos])
         maxPos = currentPos;
      currentPos++;
   }
   return maxPos;
}
```

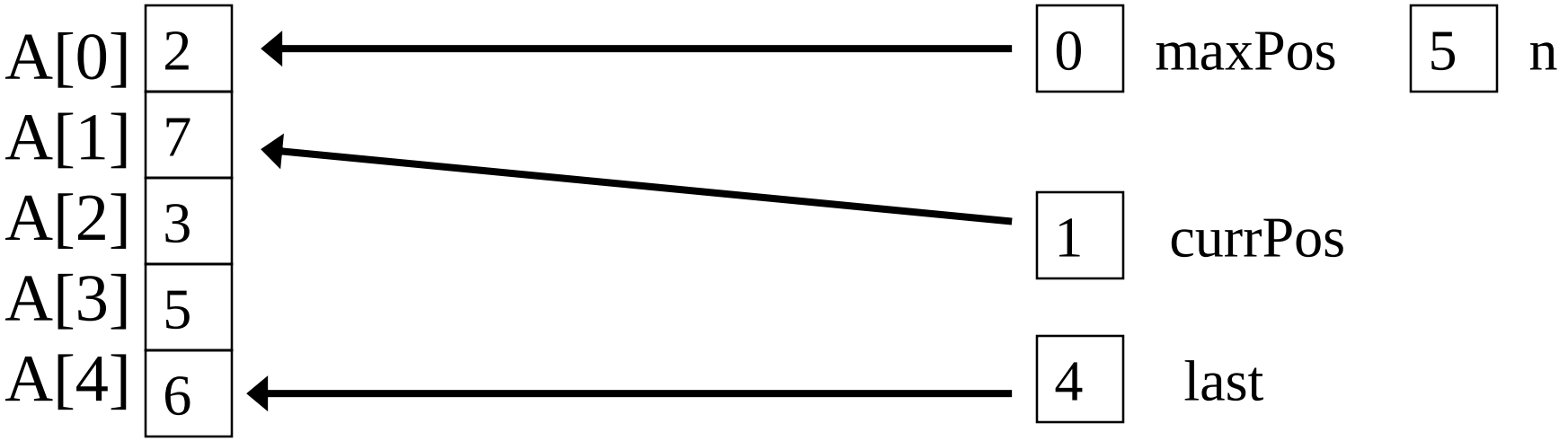
The operation of Selection Sort



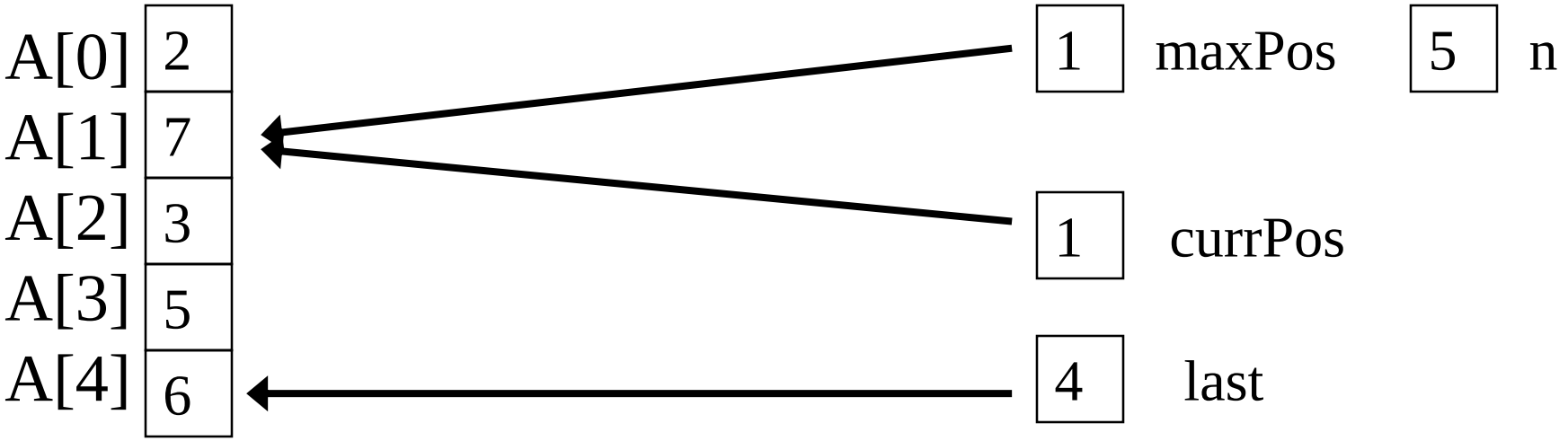
Selection Sort

```
void selectionSort(int a[], int n)
{
   int last(n-1);
   int maxPos;
   while (last > 0) {
   // invariant: a[last+1] ... a[n-1] is sorted &&
   //    everything in a[0] ... a[last] <= everything in a[last+1] ... a[n-1]
      maxPos = maxSelect(a, last+1); // last+1 is length from 0 to last
      swapElements(a, maxPos, last);
      last--;
   }
}
```

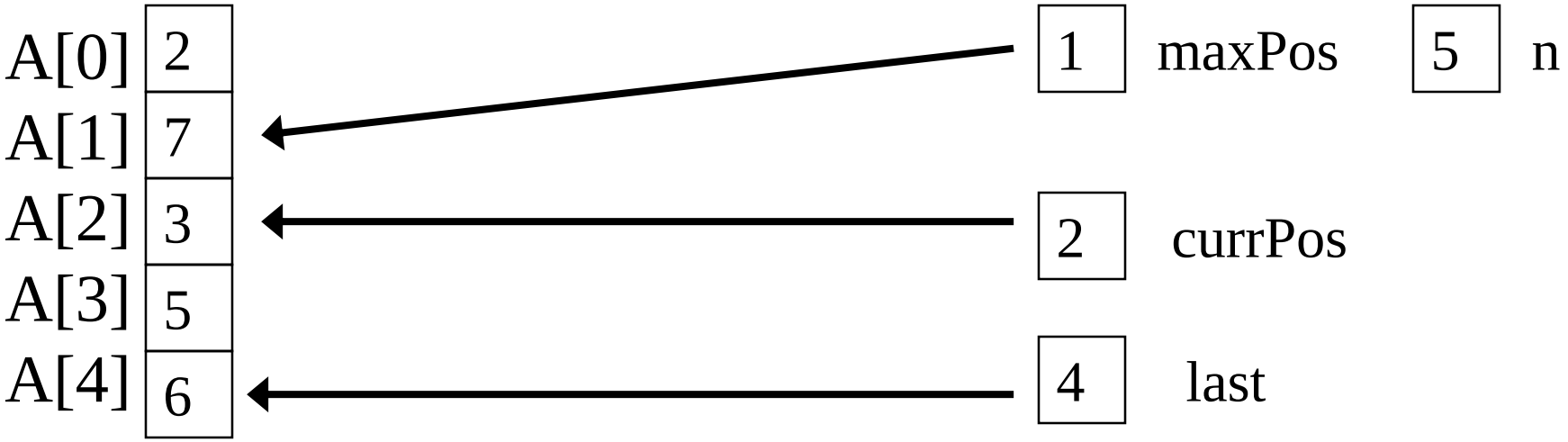
Selection Sort example



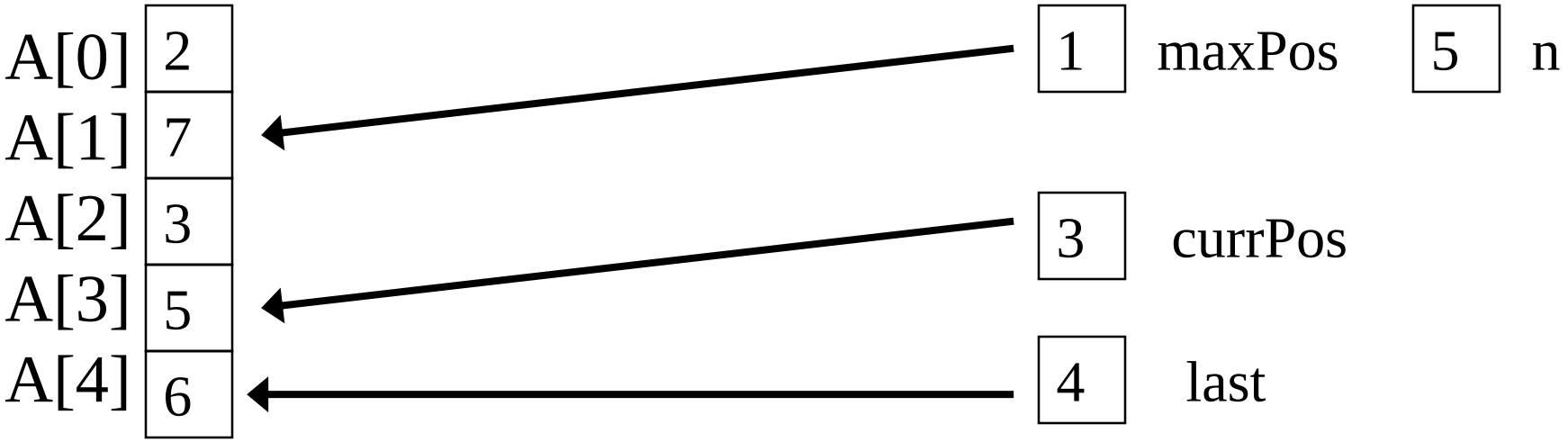
Selection Sort example



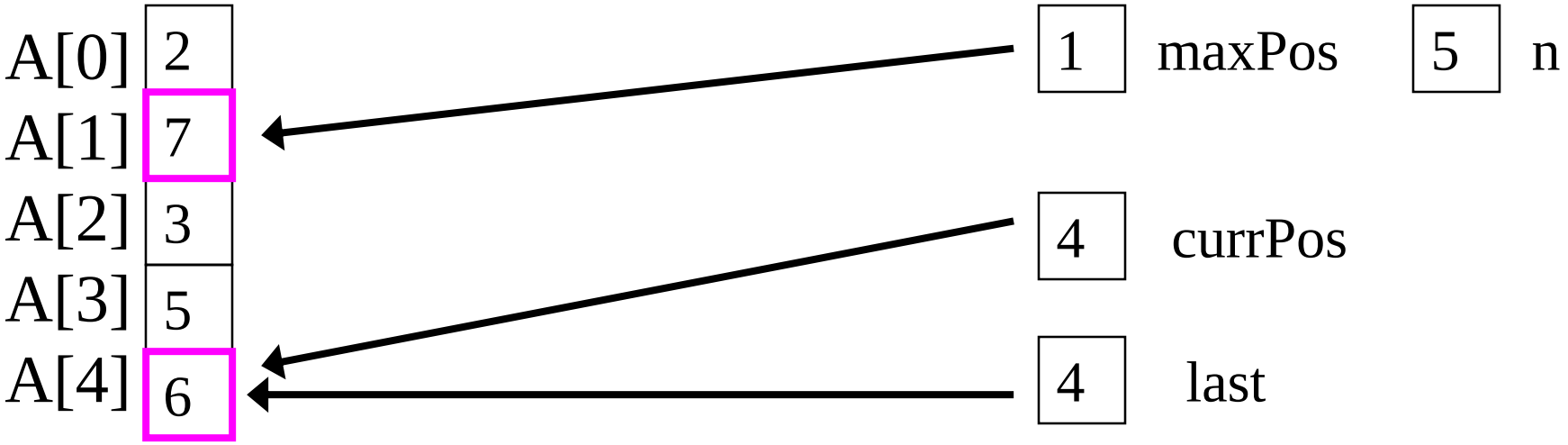
Selection Sort example



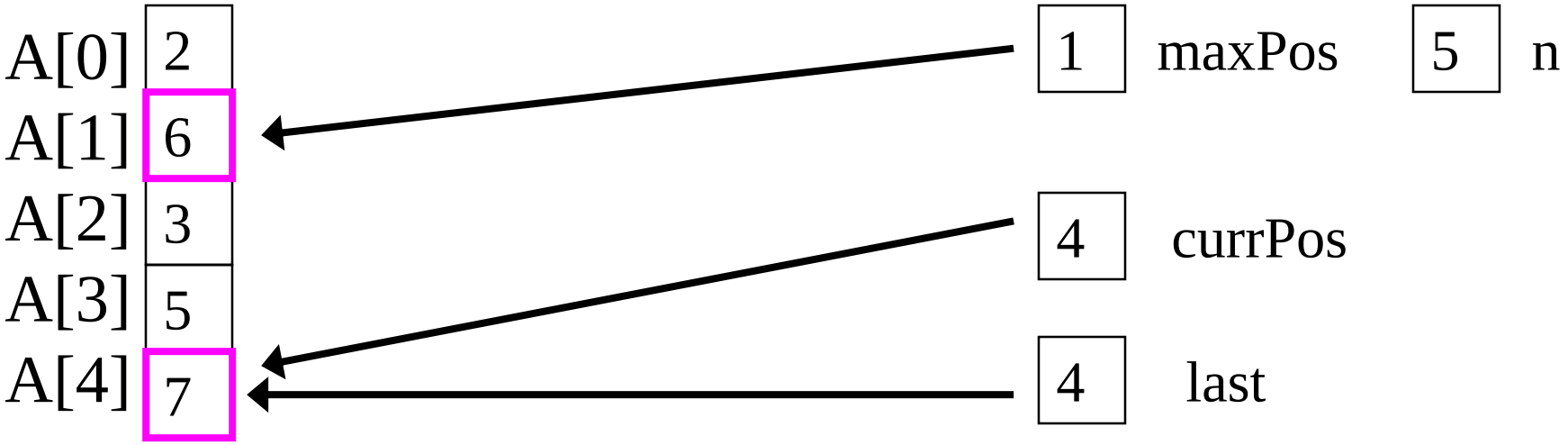
Selection Sort example



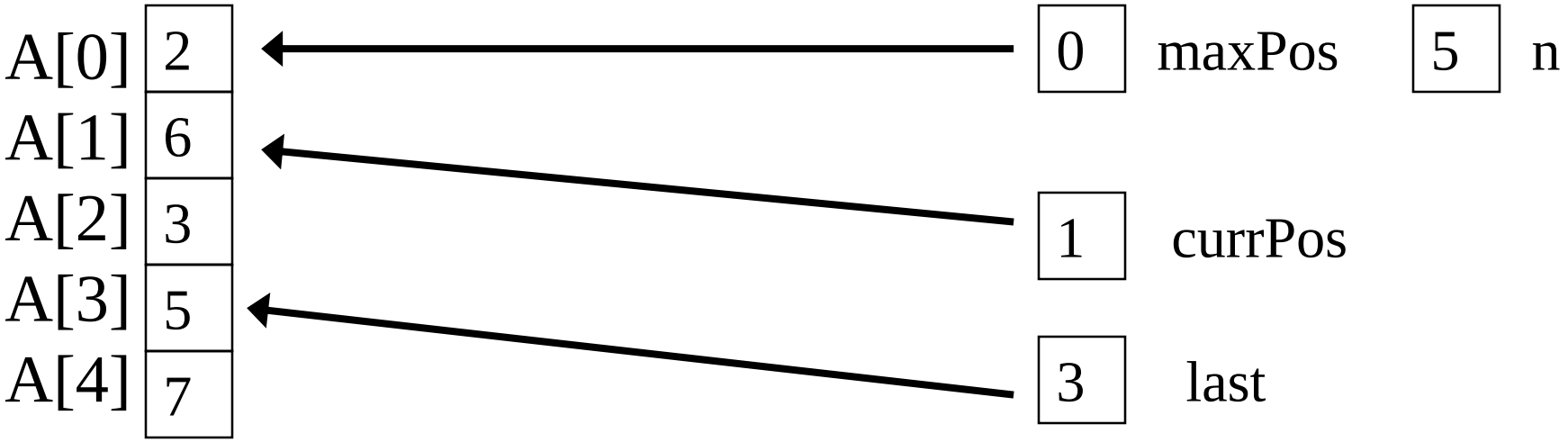
Selection Sort example



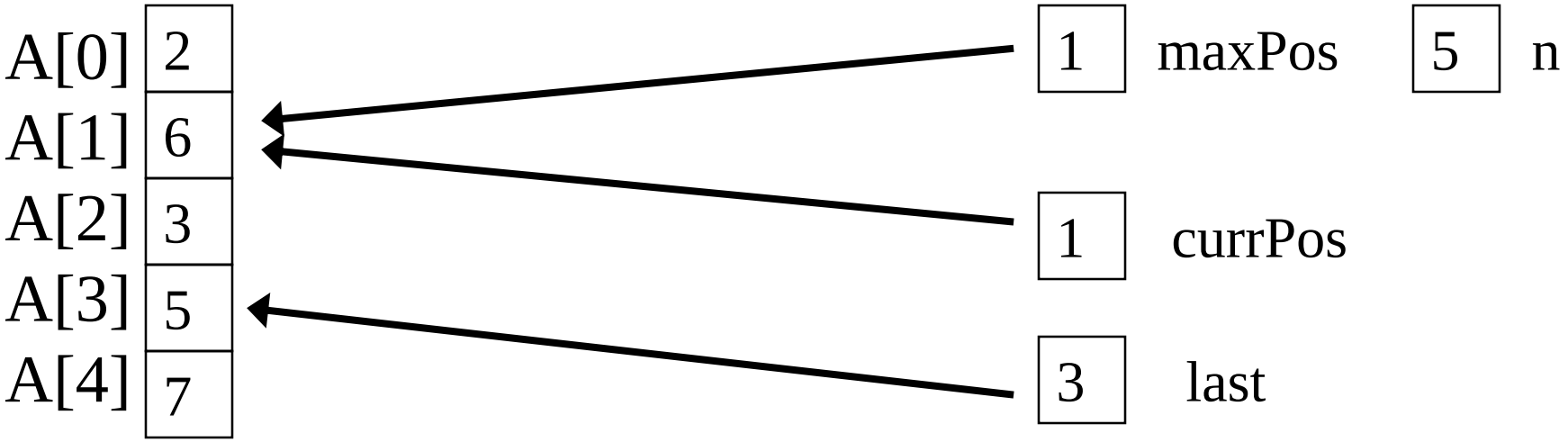
Swap, after one pass



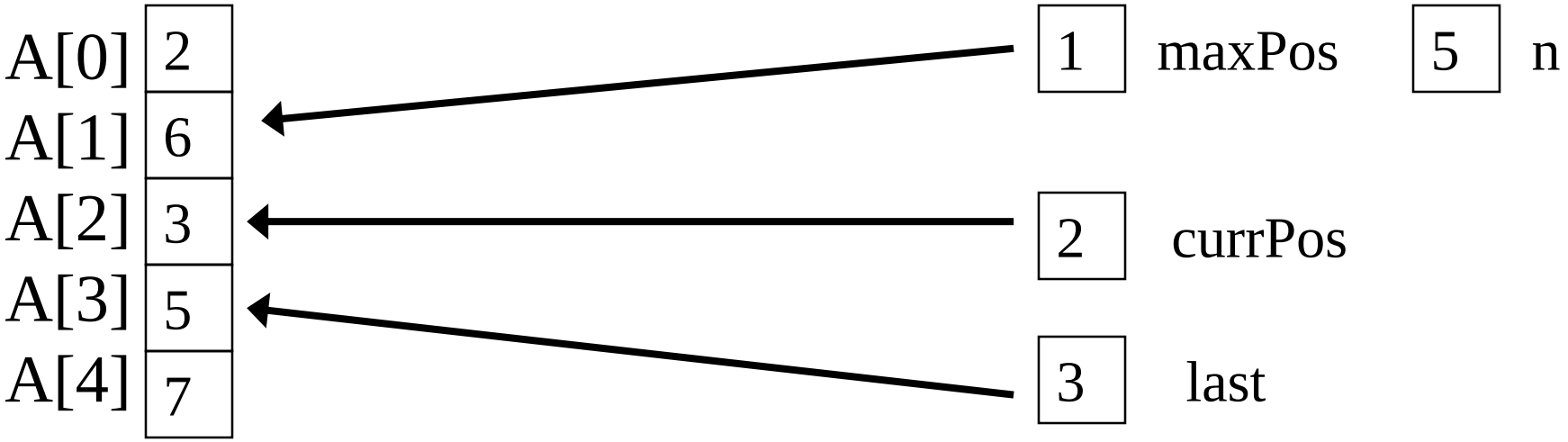
Selection Sort



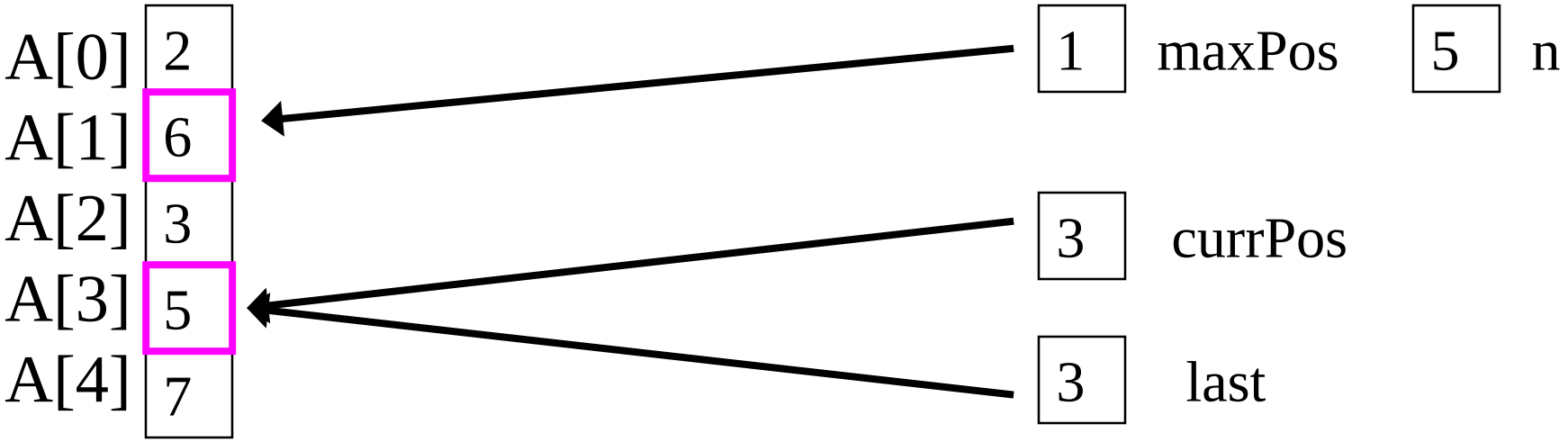
Selection Sort



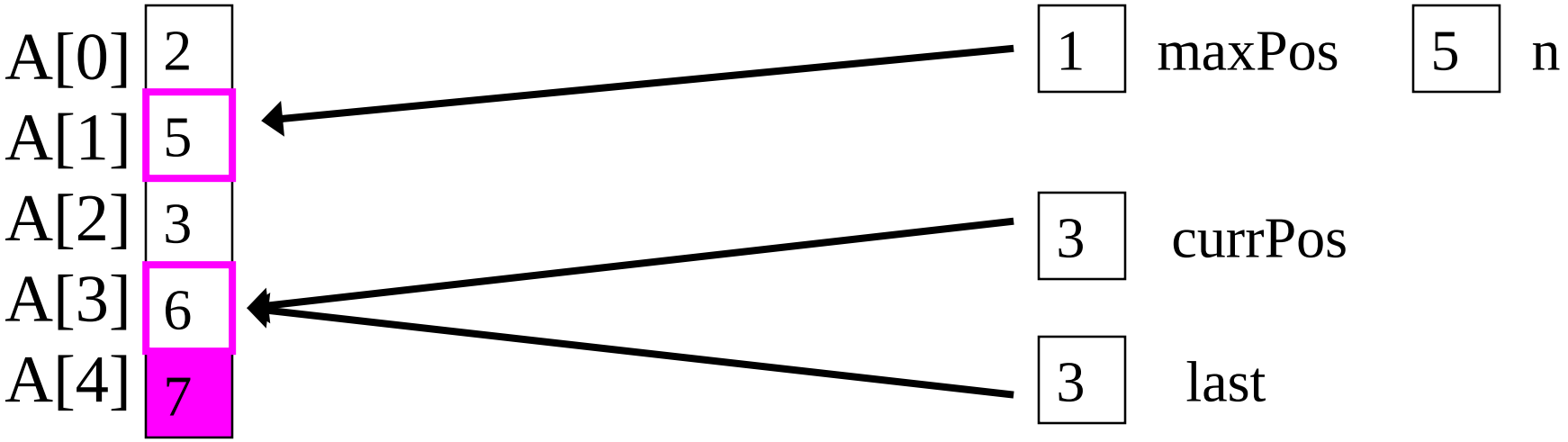
Selection Sort



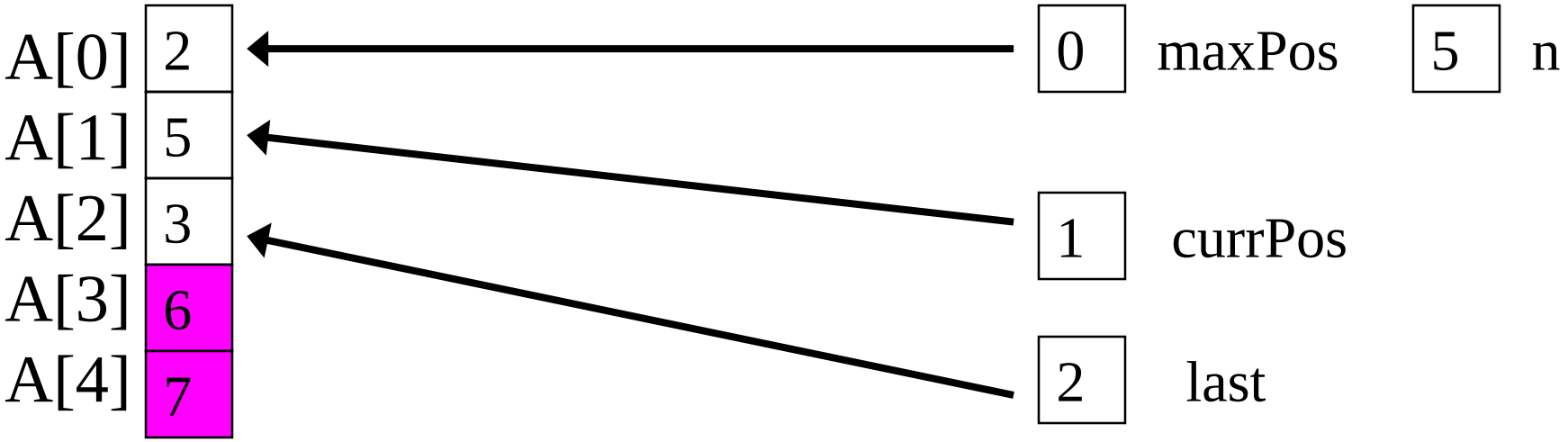
Selection Sort



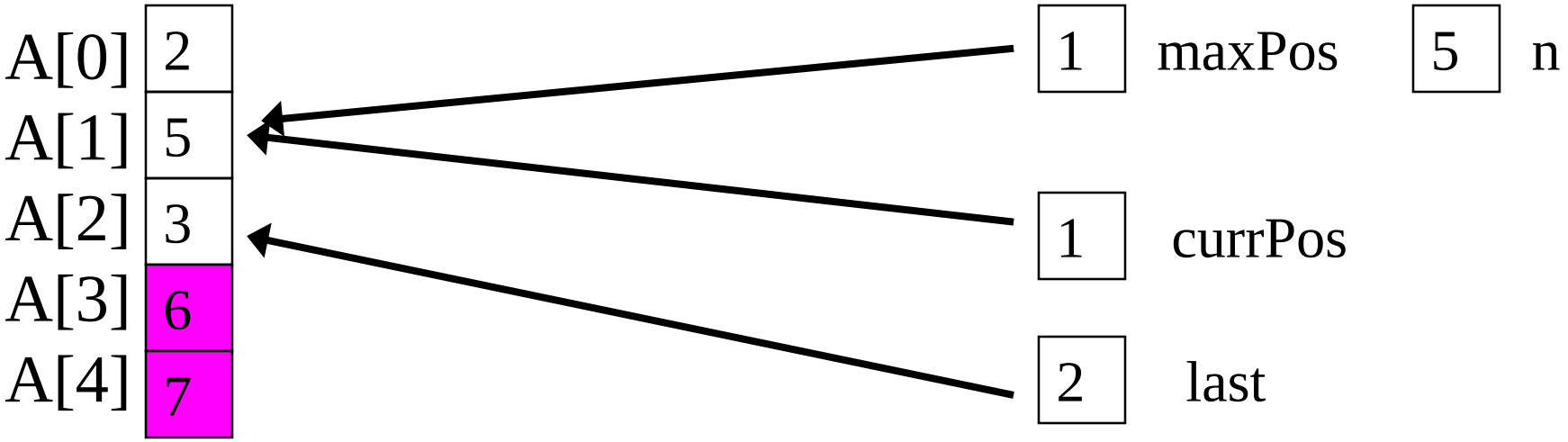
Selection Sort



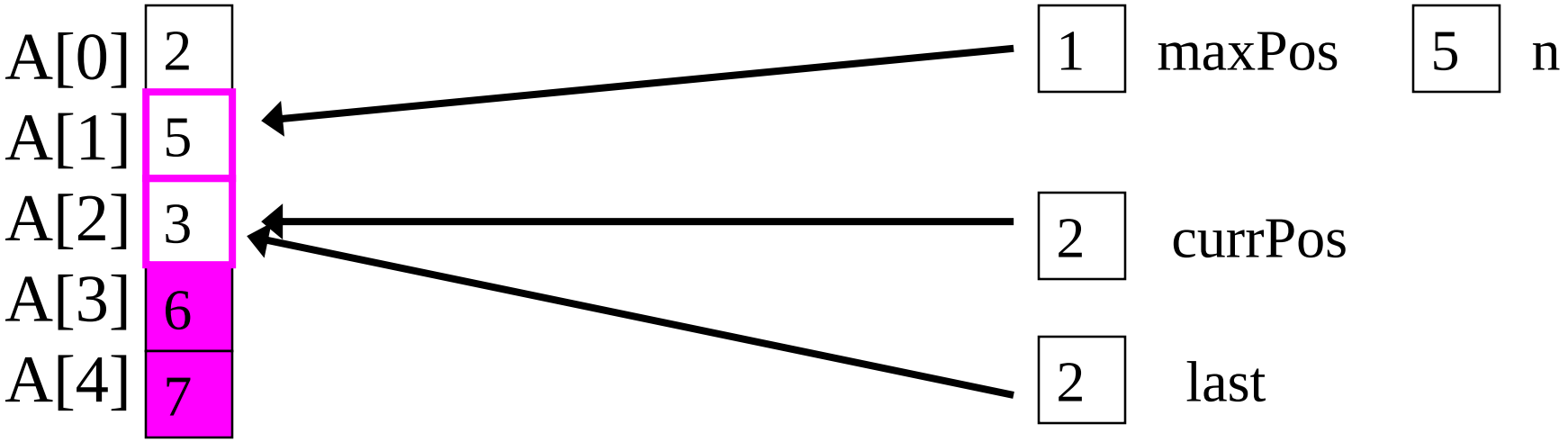
Selection Sort



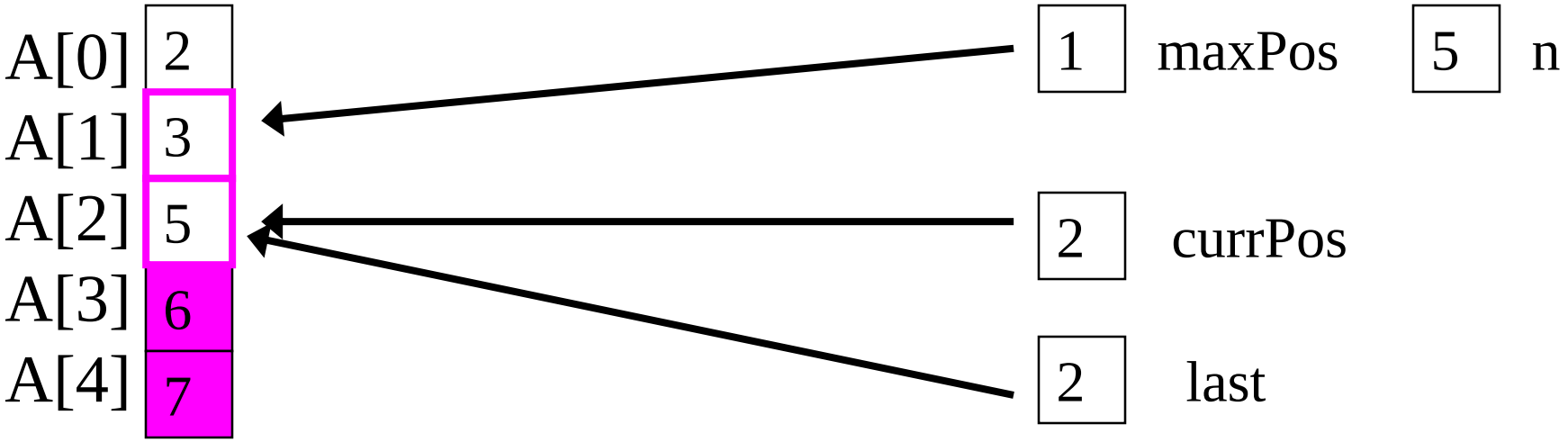
Selection Sort



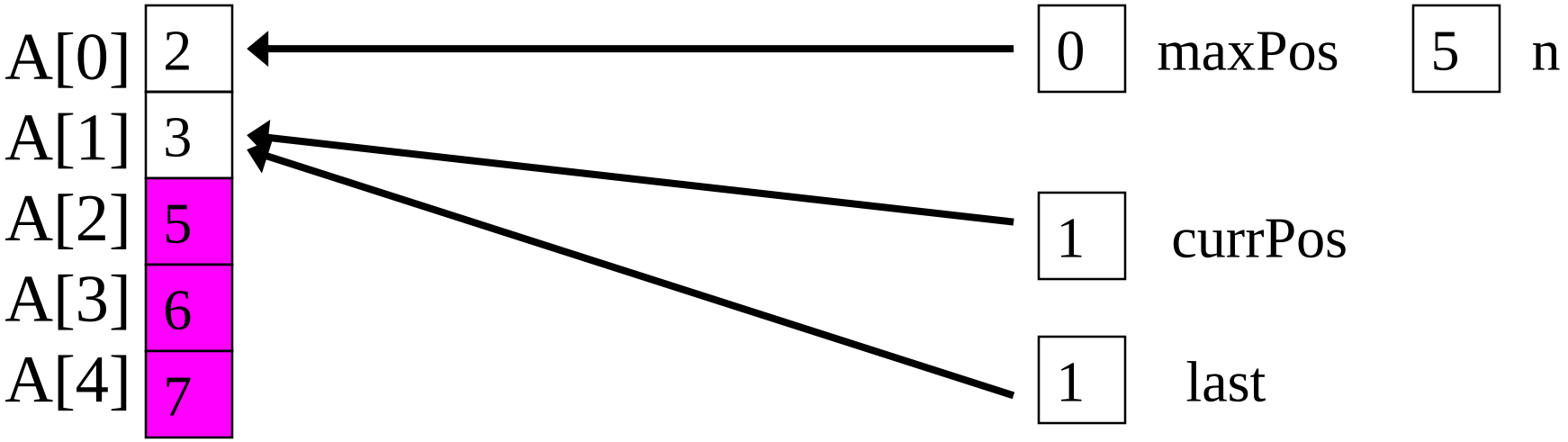
Selection Sort



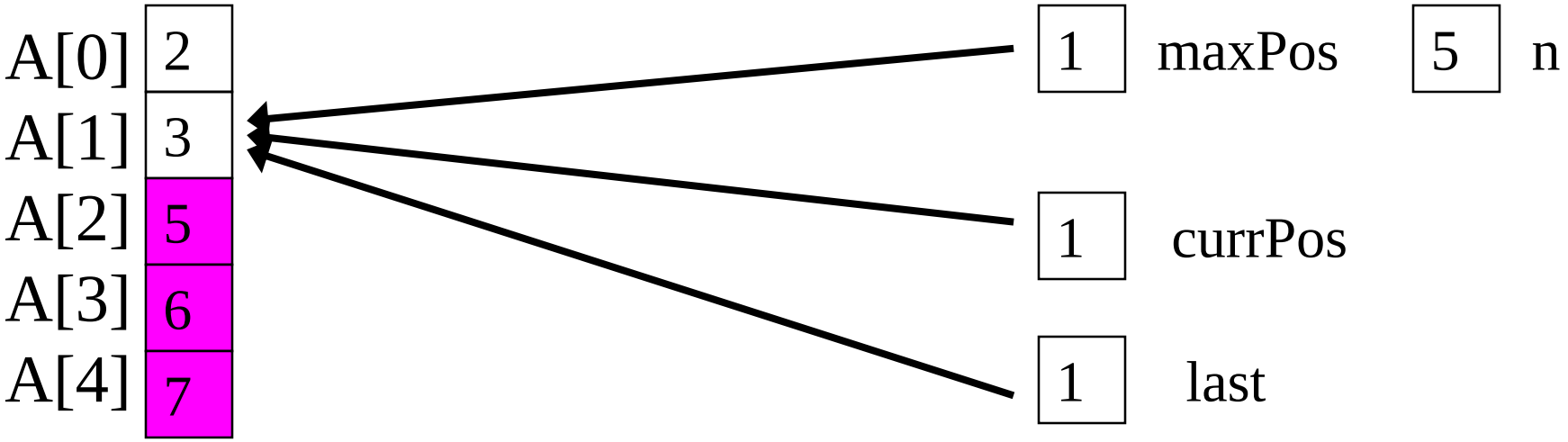
Selection Sort



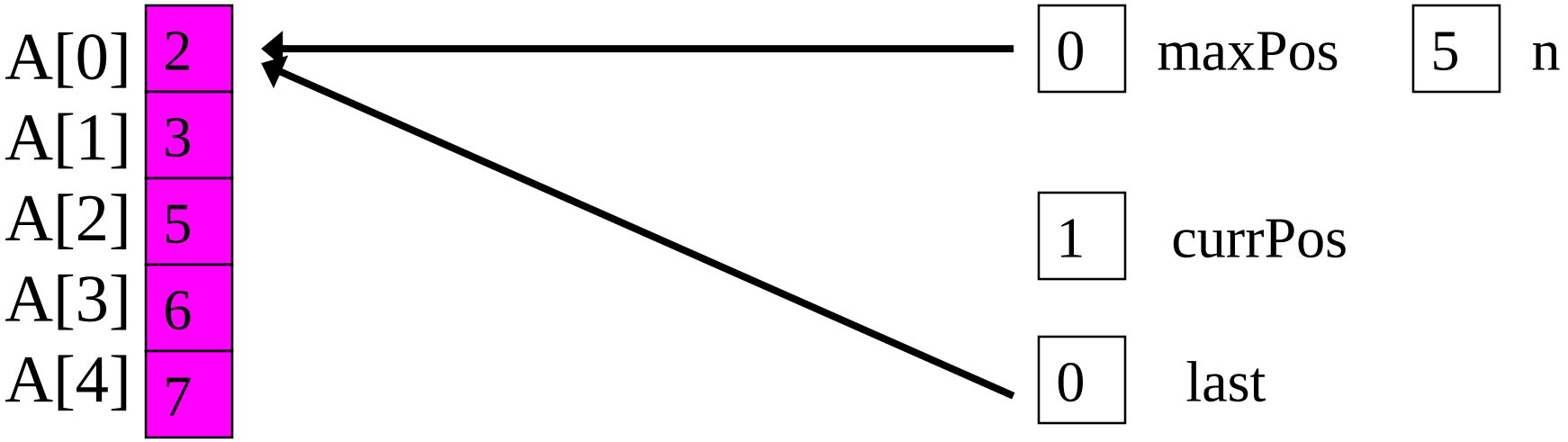
Selection Sort



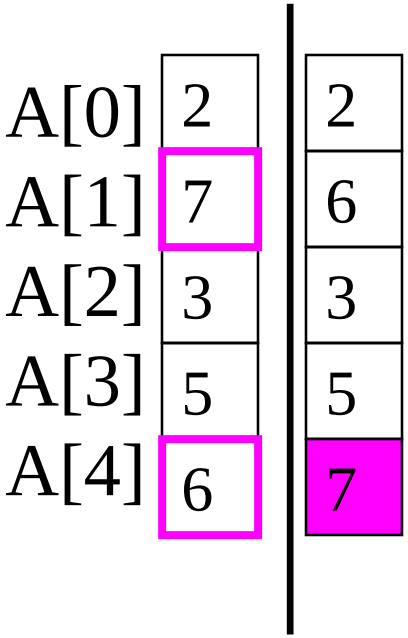
Selection Sort



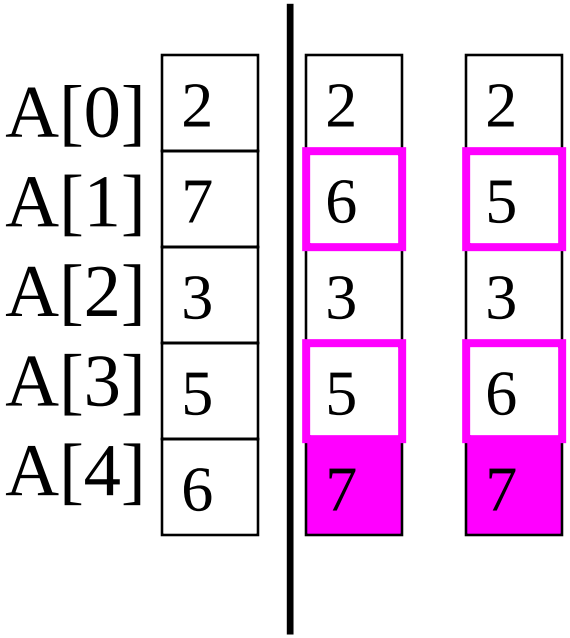
Selection Sort



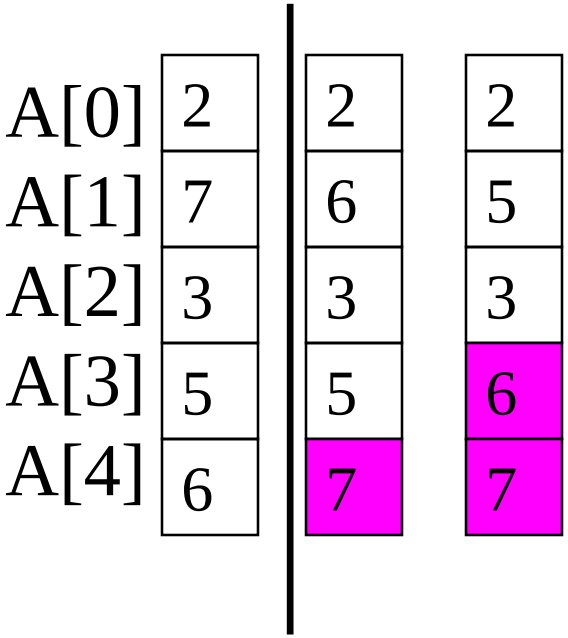
Result after each pass



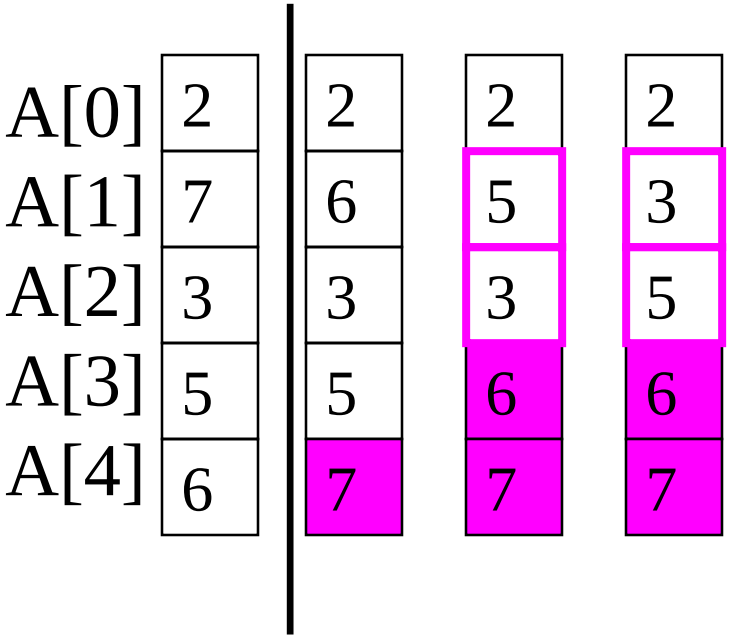
Result after each pass



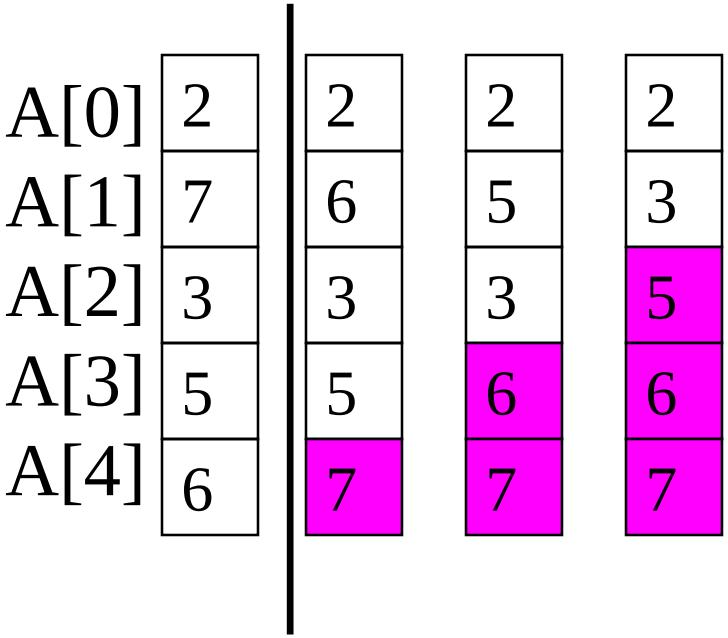
Result after each pass



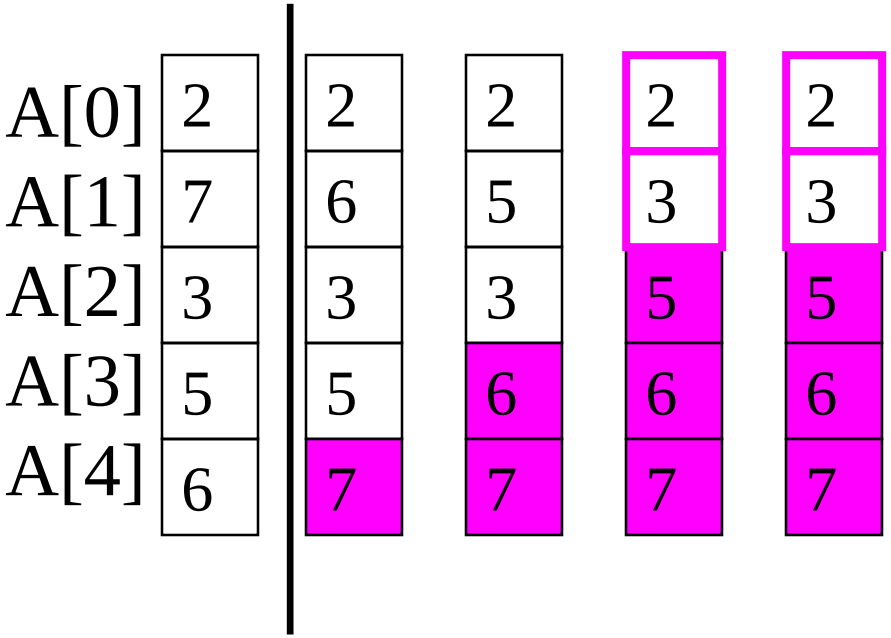
Result after each pass



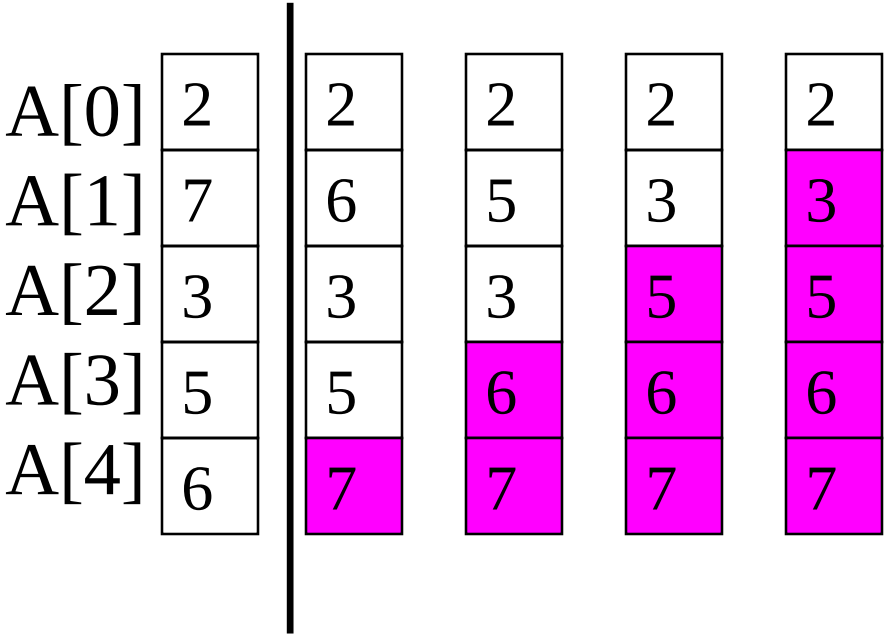
Result after each pass



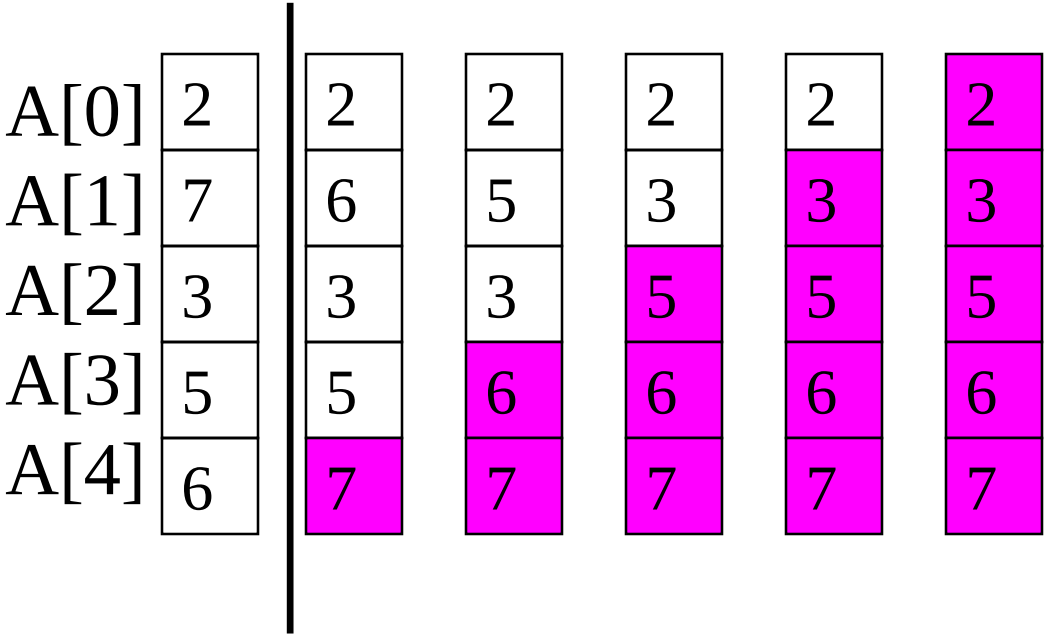
Result after each pass



Result after each pass



Result after each pass



Analysis

- **Number of passes: n-1**
  - the first pass guaranteed to place 1 item
  - the second guarantees a second
  - the third a third, etc.
  - After n-1 passes we have them all in place

Analysis: comparisons

- In the first pass we compared n-1 pairs
- In the second, n-2
- In the third, n-3, etc.
- Actual number of comparisons made across all passes, for this example was:
- 4+3+2+1 = 10

Order of complexity

- The number of passes is O(n)
- The number of comparisons in each pass is also O(n)
- Therefore, the order of complexity of the sort is O(n*n) = $O(n^2)$

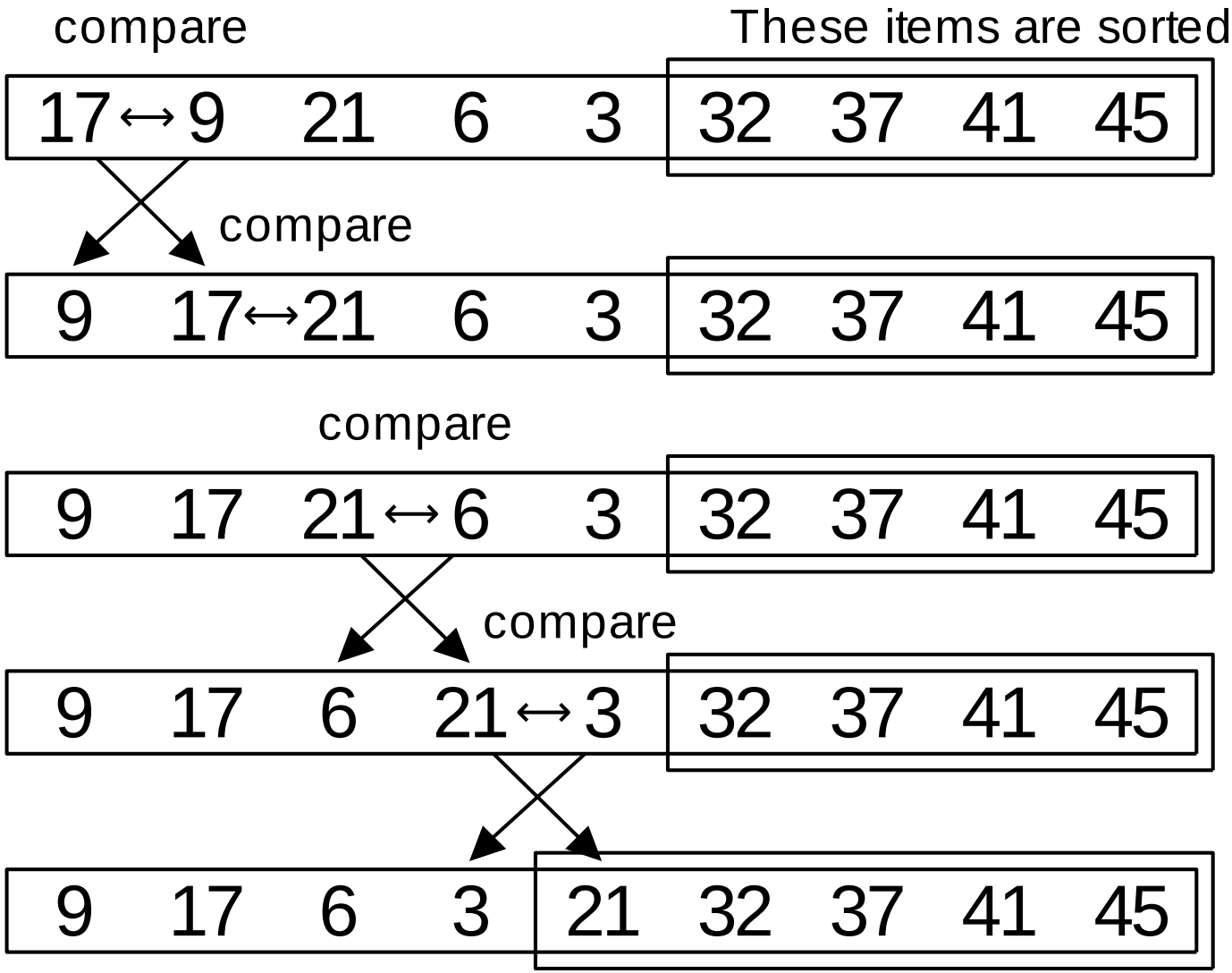
Analysis of selection sort (con't)

- **What if the list is sorted to begin with?**
- **This selection sort is a mindless one**
  - it would not know that it should stop.
- **Best case and worst case are the same (except for the swaps)**
- **The best, worst and average cases are all quadratic algorithms with $O(n^2)$**

Bubble Sort

- Also n-1 passes
- Also n-pass comparisons in each pass
- Order of complexity is therefore n-squared
- The same as the selection sort
- The main difference is that swapping may occur as many as n-1 times on a single pass, with the selection sort it only occurs once, at the end of the pass.

Example of one phase of Bubble Sort



bubbleSortPhase

```
// void swapElements(int a[], int maxPos, int last);

void bubbleSortPhase(int a[], int last)
{
   // Precondition: a is an array indexed from a[0] to a[last]
   // Move the largest element between a[0] and a[last] into a[last],
   // by swapping out of order pairs
   int pos;
```

bubbleSortPhase

```
for (pos = 0; pos < last - 1; pos++)
      if (a[pos] > a[pos+1]) {
         swapElements(a, pos, pos+1);
      }
   // Postconditions: a[0] ... a[last]
         // contain the same elements,
   // possibly reordered; a[last] >= a[0] ... a[last-1]
}
```

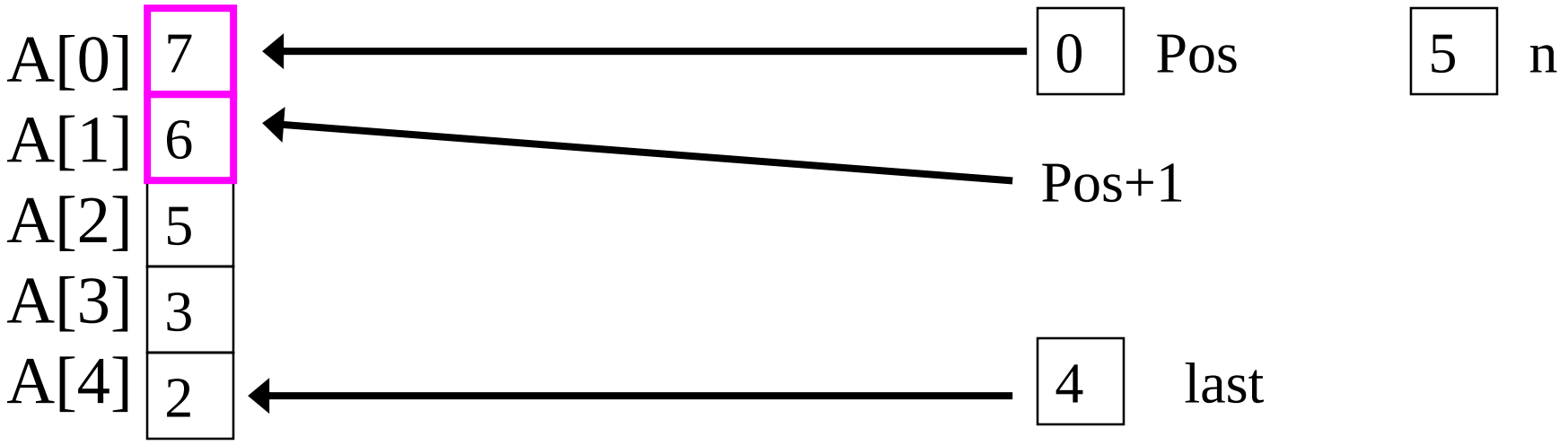
Bubble Sort

```
void bubbleSortPhase(int a[], int last);
void bubbleSort(int a[], int n)
{ // Precondition: a is an array indexed from a[0] to a[n-1]
        int i;
        for (i = n - 1; i > 0; i--)
                bubbleSortPhase(a, i);
        // Postcondition: a is sorted
}
```

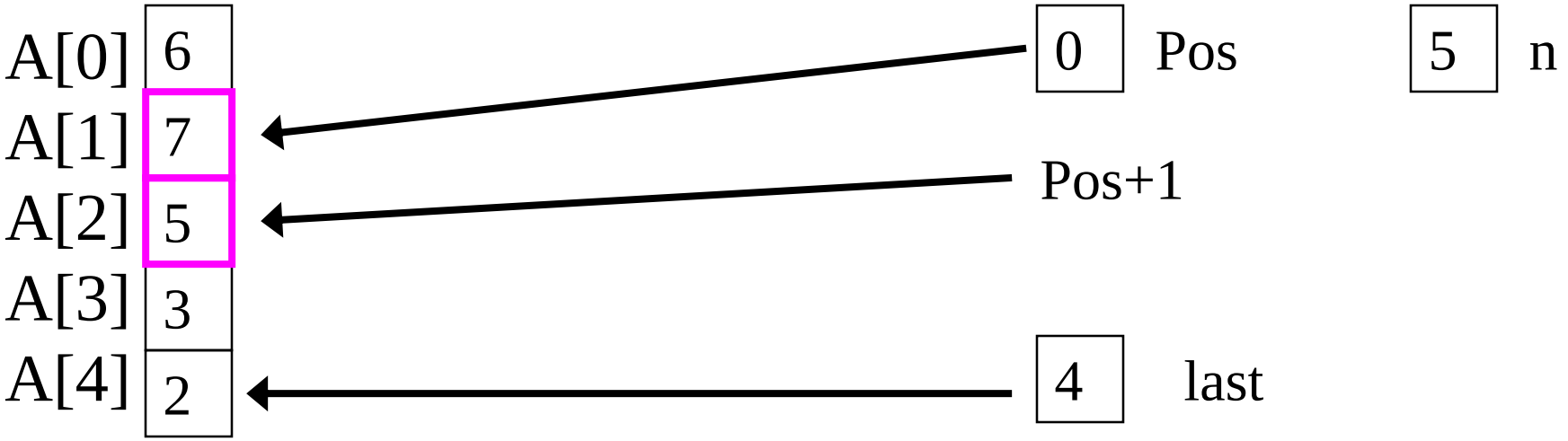
Version 1 of bubble sort

- This version uses n-1 passes
- During each pass, n-1 pairs are compared
- Every time a pair needs to be swapped this is done before the pass can continue

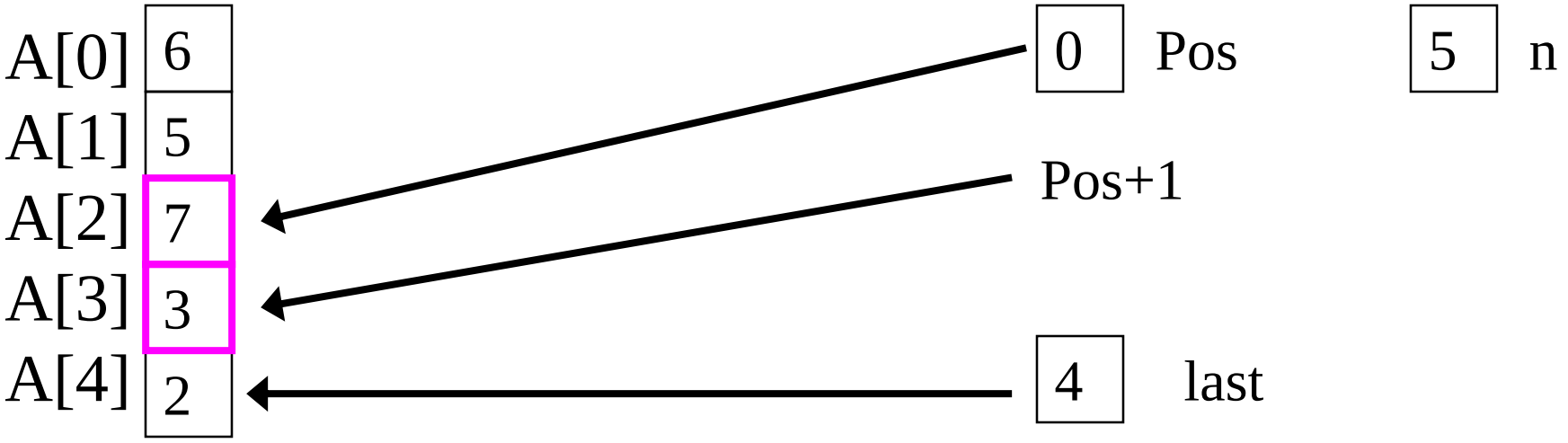
Bubble Sort: difficult example



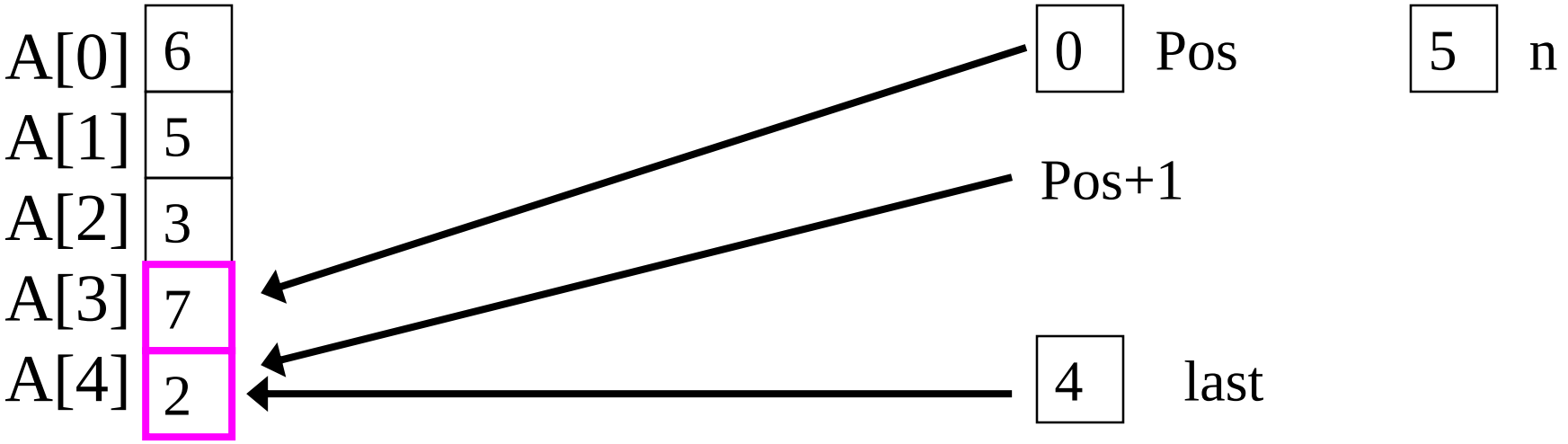
Bubble Sort: difficult example



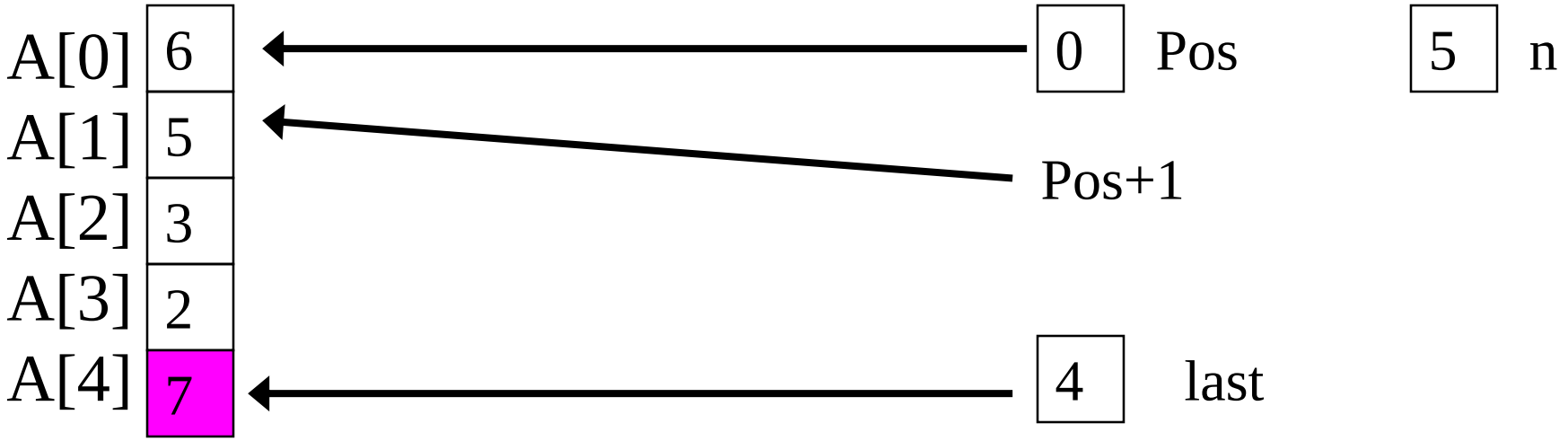
Bubble Sort: difficult example



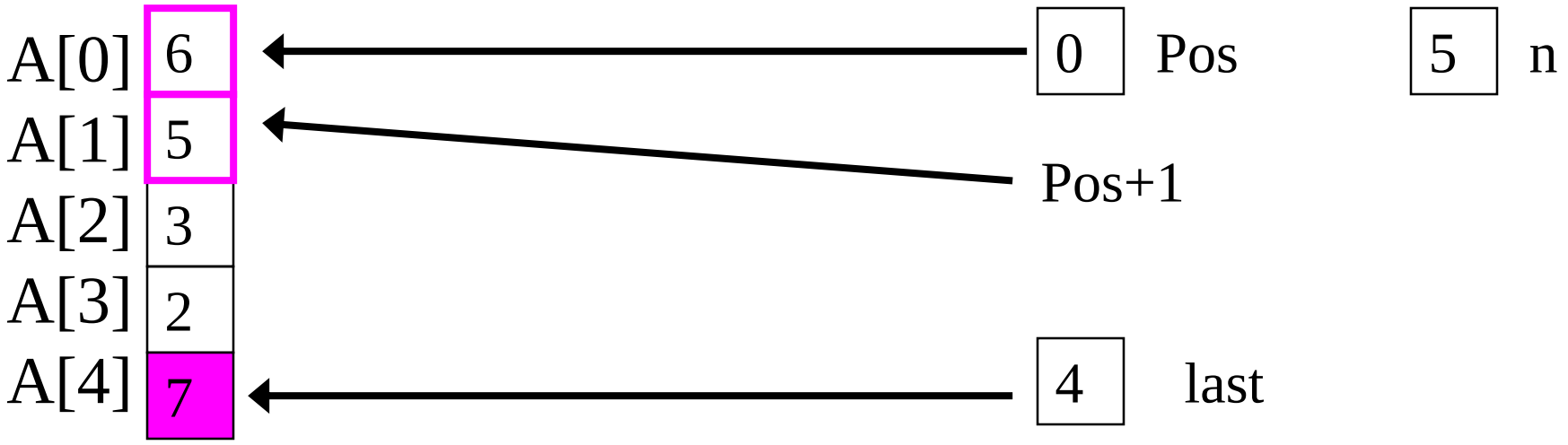
Bubble Sort: difficult example



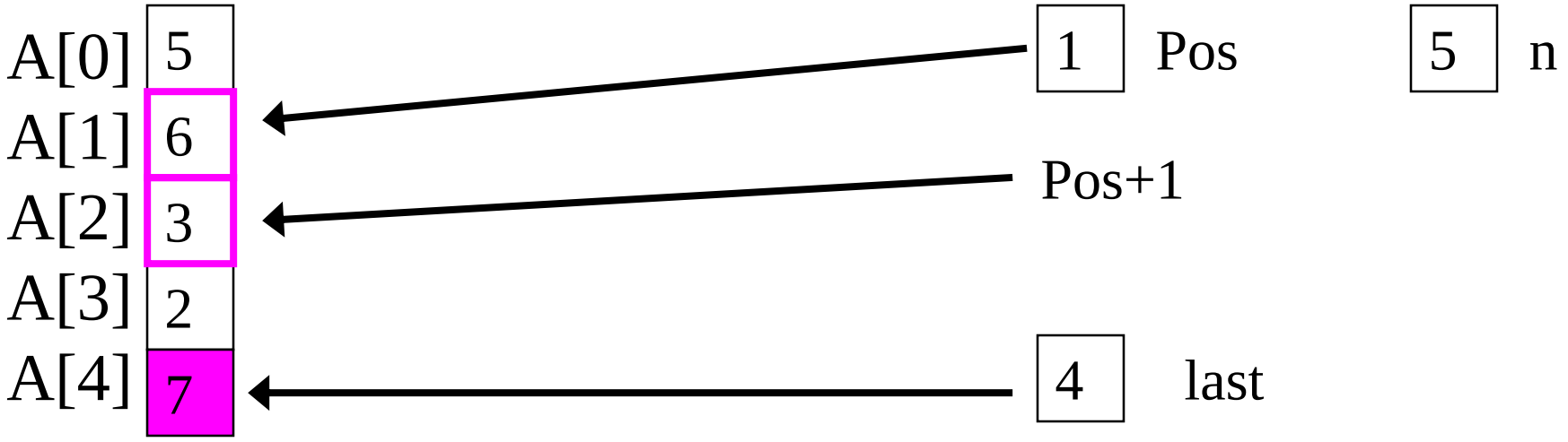
Bubble Sort: difficult example



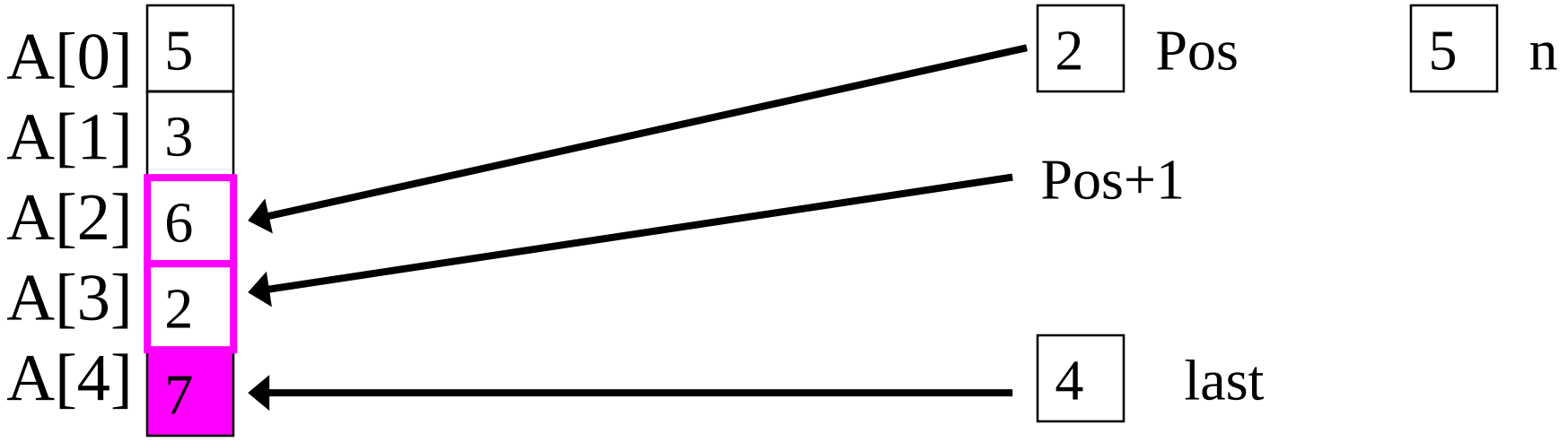
Bubble Sort: difficult example



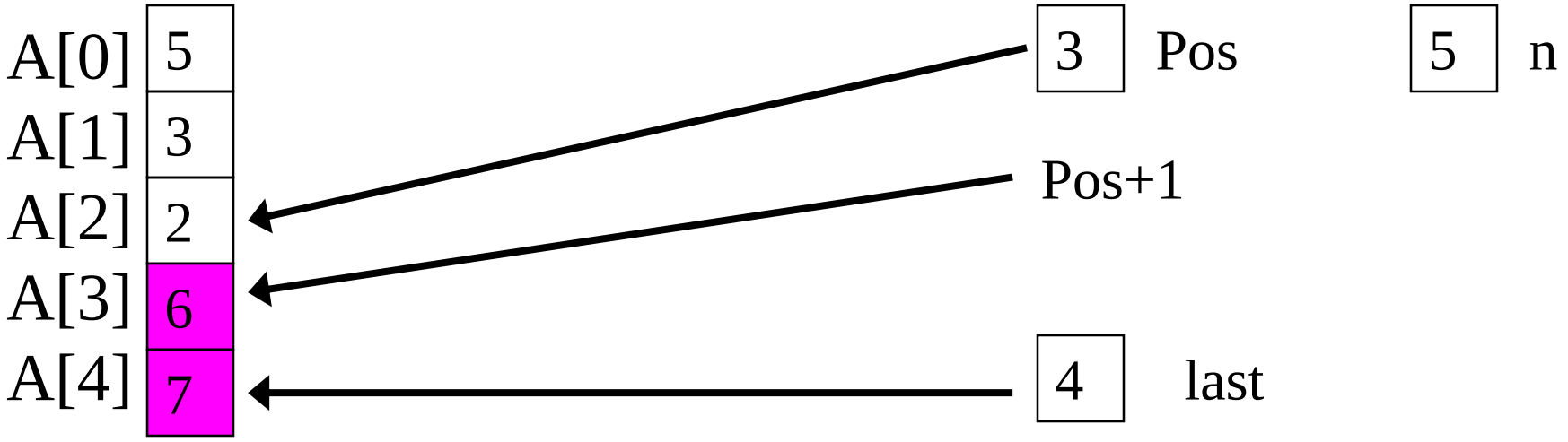
Bubble Sort: difficult example



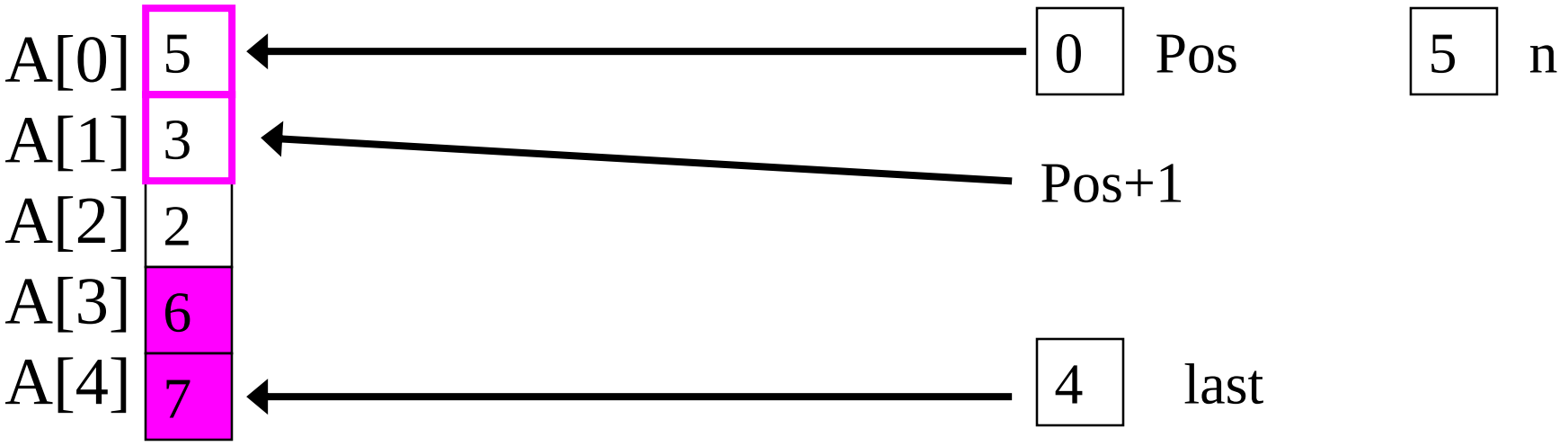
Bubble Sort: difficult example



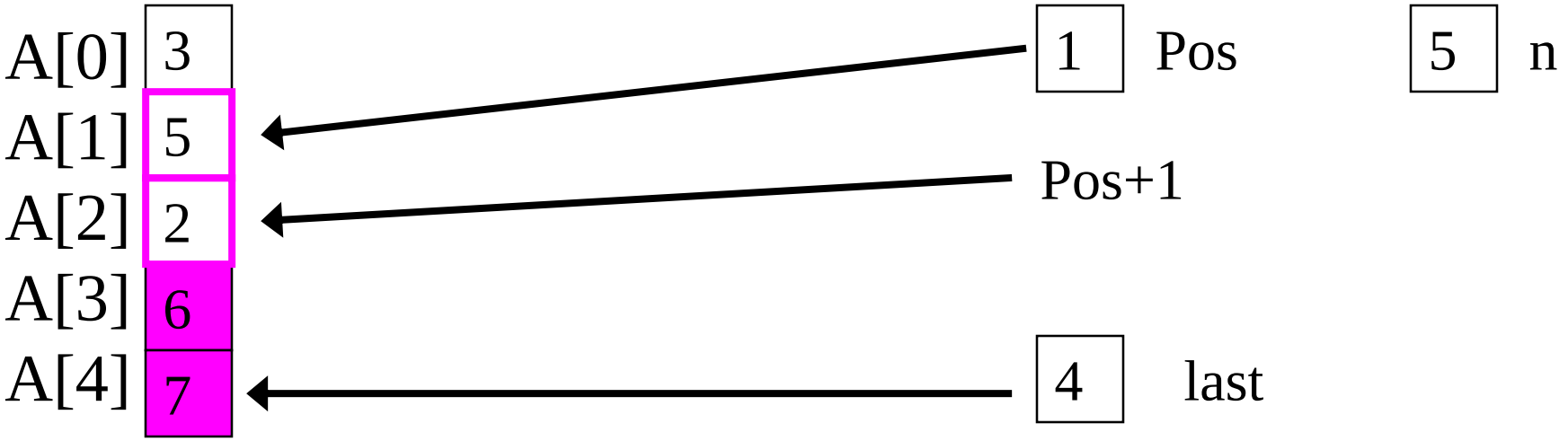
Bubble Sort: difficult example



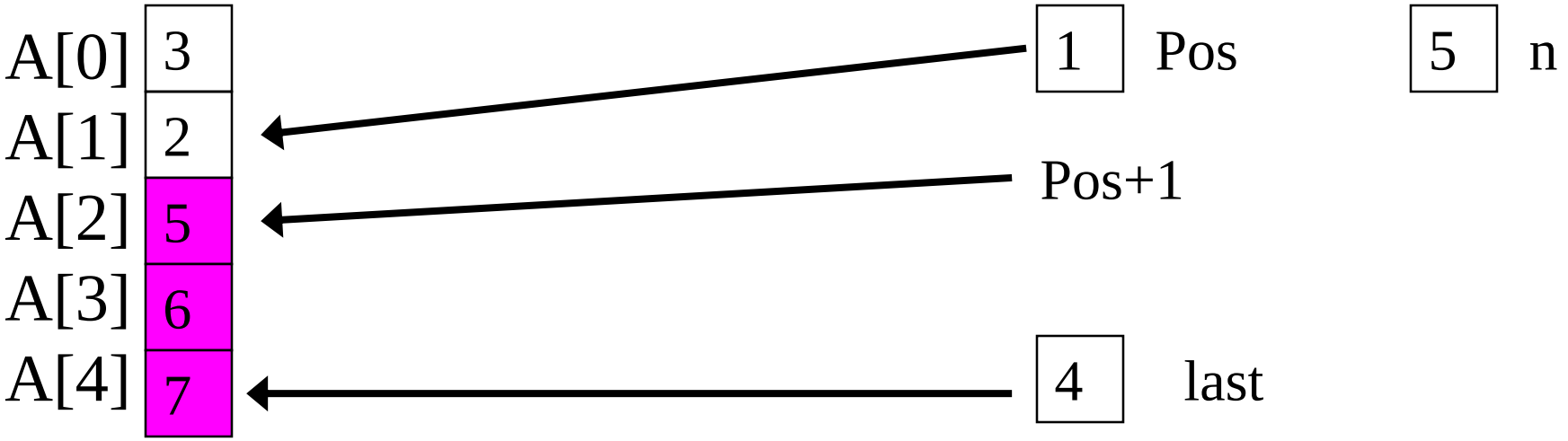
Bubble Sort: difficult example



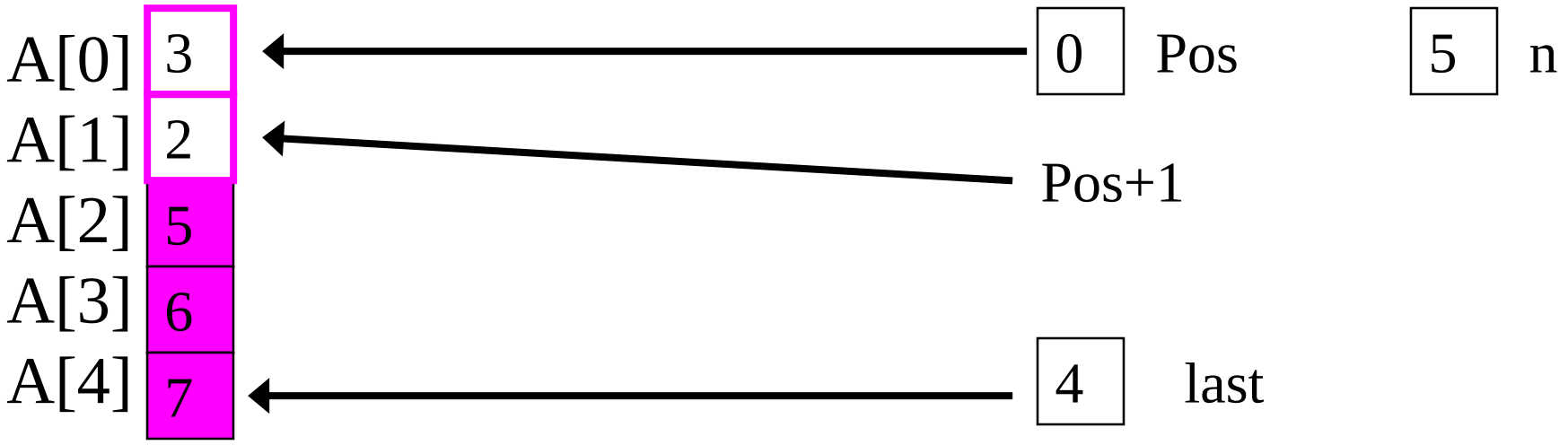
Bubble Sort: difficult example



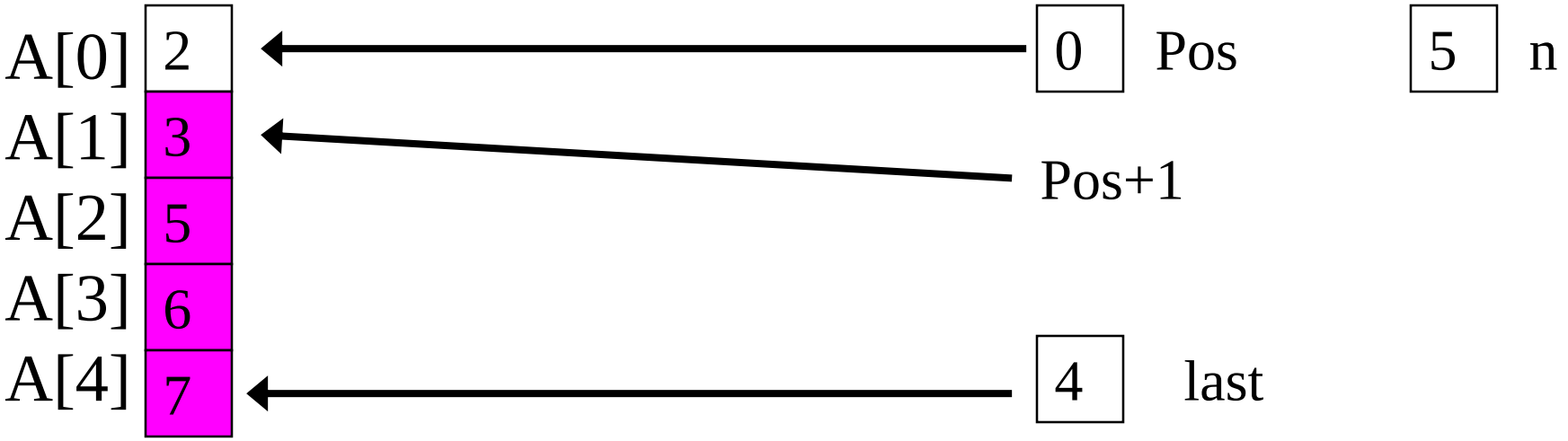
Bubble Sort: difficult example



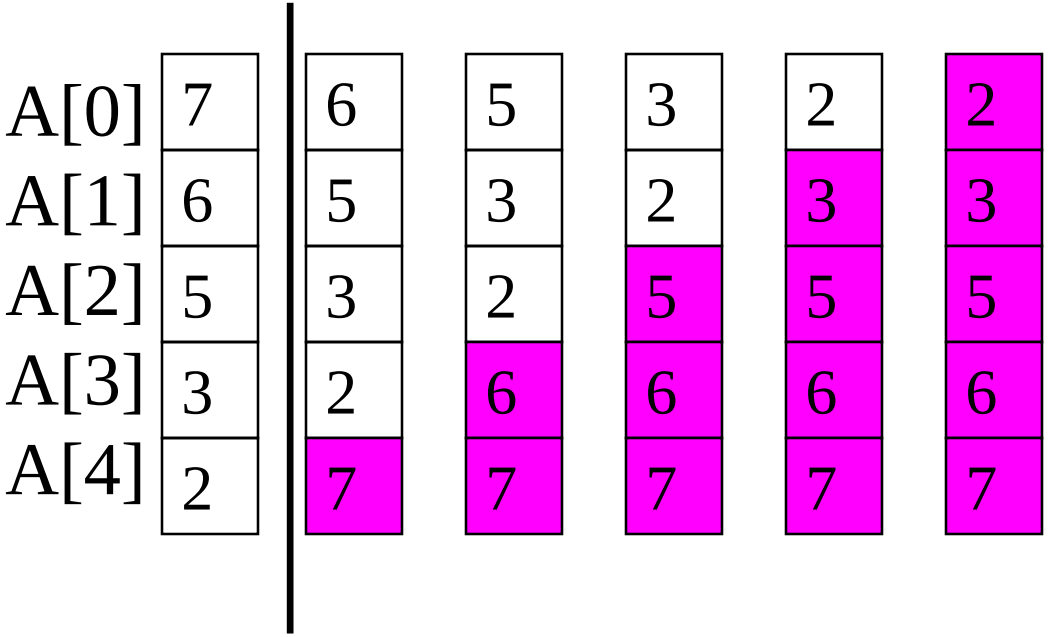
Bubble Sort: difficult example



Bubble Sort: difficult example



Result after each pass



Analysis

- For this example, we made n-1 passes through the array
- Each time we looked at n-1 pairs
- The number of passes is O(n)
- The number of pairs processed in each pass is O(n)
- So, overall we get O(n*n) = $O(n^2)$

Selection and Bubblesort comparison

- Both are $O(n^2)$ sorts
- If we count up the number of critical operations for both sorts, handling n-1, n-2, etc. pairs for each pass, using the data in the last example, we get
- Selection sort: 10 comparisons
- Bubble sort: 10 comparisons

What about swaps?

- The same swap function can be used for both programs.
- Let's say it takes 3 operations, then the actual number of operations is
- Selection sort: 10 + 3(n-1) = 22
- Bubble sort: 10 + 3(10) = 40
- The selection sort uses roughly half the number of operations as the bubble sort

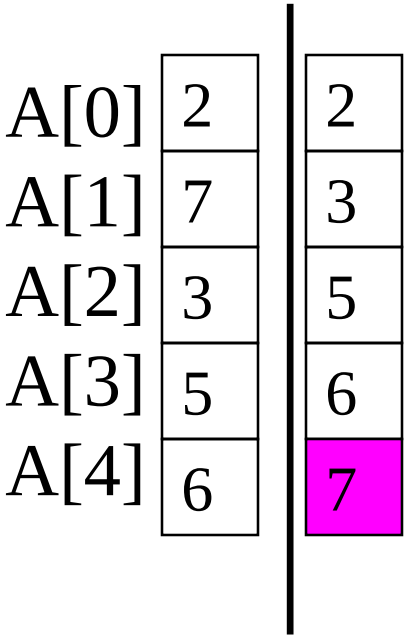
Moral

- **Two sorts of the same order $O(n^2)$ may not be the same speed.**
- **It depends on the data sets they are sorting.**
- **It also depends on the way they are implemented.**

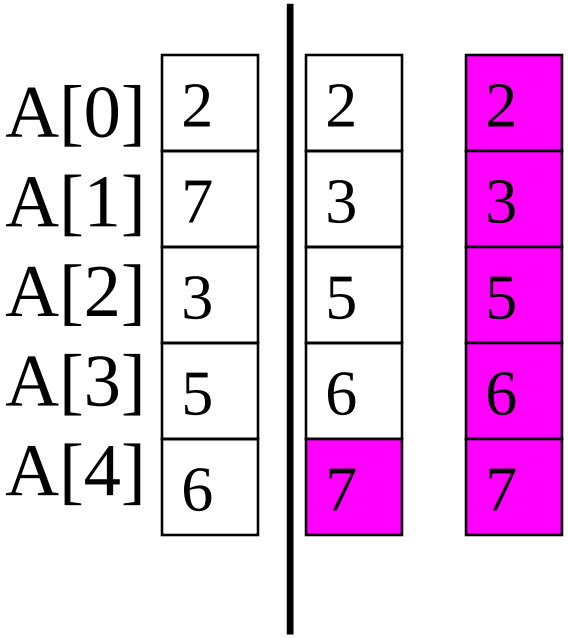
Data sets

- **The data set we chose for the bubble sort was the worst one possible.**
- **What happens if we run it on the data set we originally used for the selection sort?**

Result after pass 1



Result after pass 2



Improvement needed

- Neither the selection sort, nor the bubble sort know enough to stop if the list is sorted early!
- We should be able to come up with a smart version of the bubble sort that can do this.
- Instead of the outer for loop, let's use a while loop that runs until the array is sorted.

Further improvements

- We will also make sure the inner loop runs the minimum number of times (n-pass)
- and that we keep track of whether a swap was needed for the pass we are on.
- If a swap was needed then we cannot assume the array is sorted.
- If a swap was not needed, then the array is sorted and we should send that signal to the outer loop.

Bubble sort improvements

- **What improvements can you suggest for the bubble sort that we have seen?**

Improvements

- **Do not look at portions of the array that are already in sorted order**
- **Leave when the array is sorted**
  - this is something the insertion and selection sorts we have seen could not do.

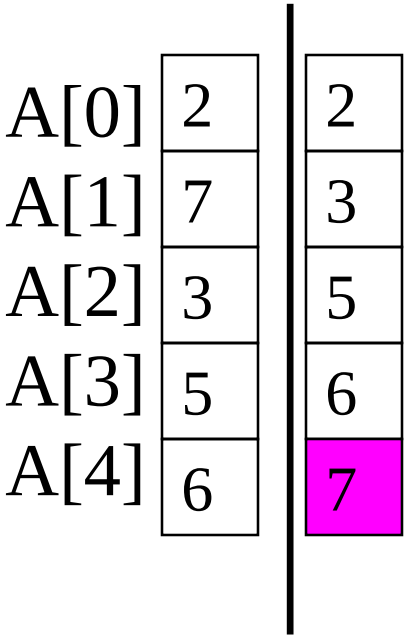
Improvements

- How much of an operational difference does the smart bubble sort have over it's dumb form?
- How much of an operational difference does the smart bubble sort have over the selection sort?
- Do these improvements make a difference in Big-O?
- Which form of sort is best?

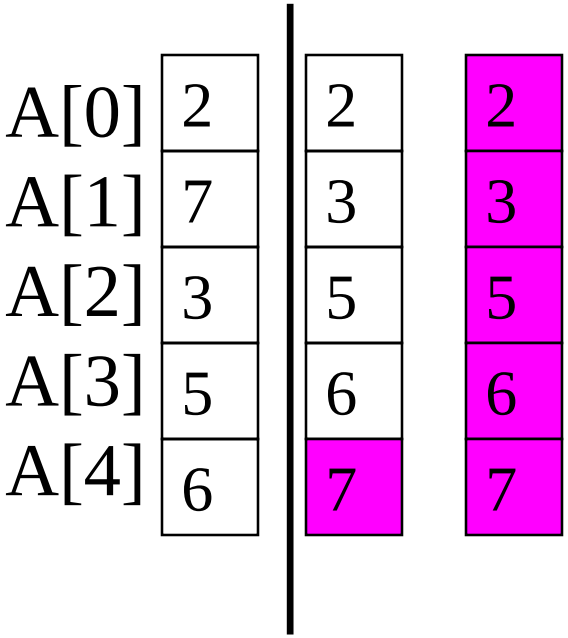
Bubble Sort

```
void bubbleSort(int a[], int n)
{ // Precondition: a is an array indexed from a[0] to a[n-1]
      bool sorted(false);  int pass(0);
      while (!sorted) {
          sorted = true;
          for (pos = 0; pos < last - 1; pos++)
              if (a[pos] > a[pos+1])
                  { swapElements(pos, pos+1); sorted = false; }
          pos++;
      }  // Postcondition: a is sorted
}
```

Result after pass 1



Result after pass 2



Analysis

- Only two passes are made
- 4 pairs are checked on pass 1
- swap is called 3 times on pass 1
- 3 pairs are checked on pass 2, no swaps
- Total operations:
- 4+3+3 = 10
- Better than the selection sort.
- The savings would be even bigger for larger lists that were sorted early.

Bubble sort: final analysis

- Not counting swapping, the best case is only n-1 operations!
- Worst case is still n-squared
- Order of complexity same as selection sort no matter what.

Equation 5-7

$$n-1+n-2+\ldots+1=\sum_{i=1}^{n-1} i=O(N^2)$$