Structured Programming

- Sorting

Donglong Chen

Sorting

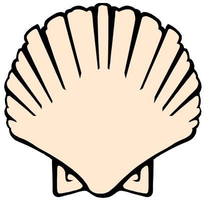
- To arrange a set of items in sequence.
- It is estimated that 25~50% of all computing power is used for sorting activities.
- Possible reasons:
  - Many applications require sorting;
  - Many applications perform sorting when they don't have to;
  - Many applications use inefficient sorting algorithms.

Sorting Applications

- To prepare a list of student ID, names, and scores in a table (sorted by ID or name) for easy checking.
- To prepare a list of scores before letter grade assignment.
- To produce a list of horses after a race (sorted by the finishing times) for payoff calculation.
- To prepare an originally unsorted array for ordered binary searching.

Some Sorting Methods

- Selection sort
- Bubble sort
- Shell sort (a simple but faster sorting method than above)
- Quick sort (a very efficient sorting method for most applications;)



Selection Sort

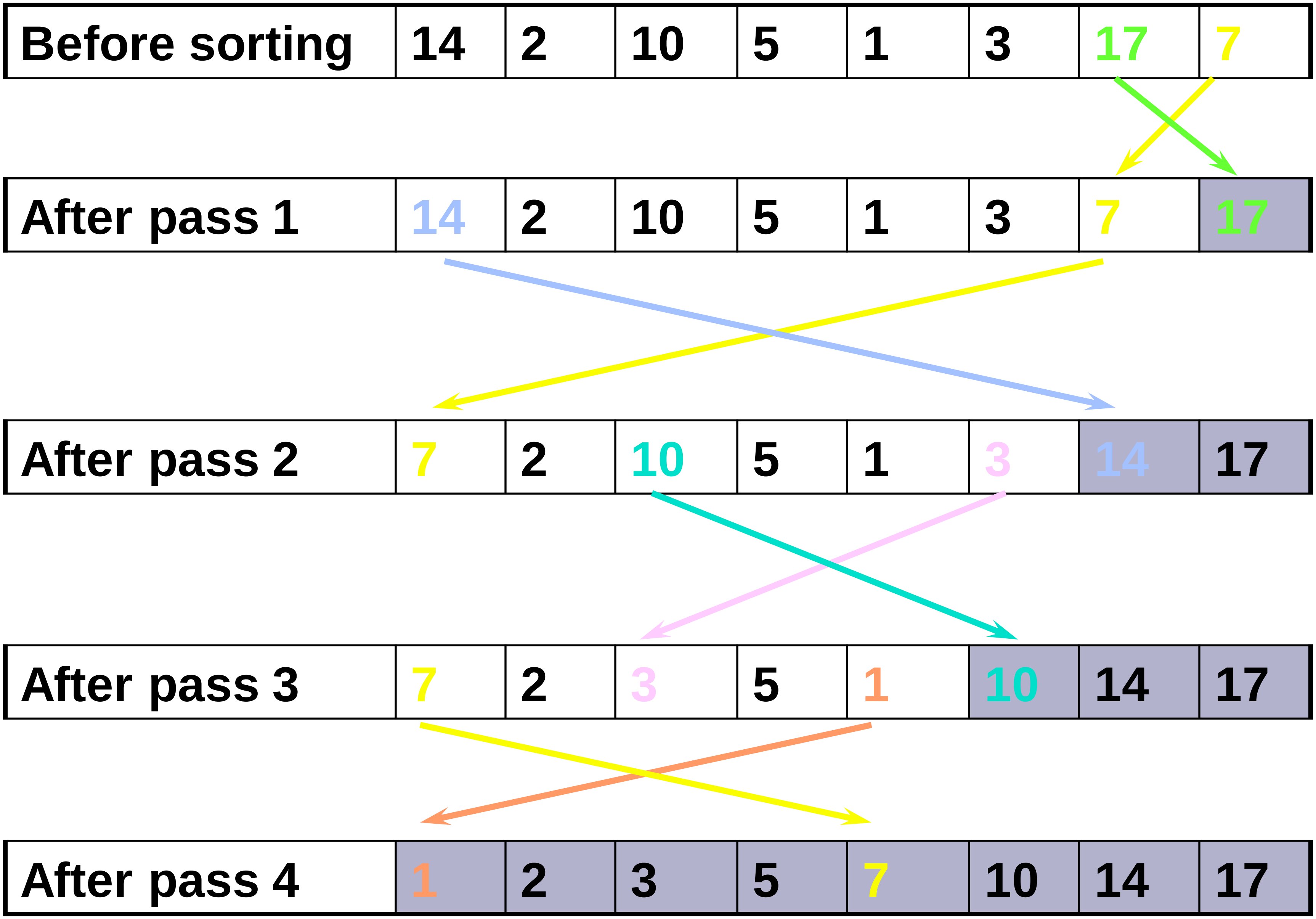
- Selection sort performs sorting by repeatedly putting the largest element in the unsorted portion of the array to the end of this unsorted portion until the whole array is sorted.

- It is similar to the way that many people do their sorting.

Selection Sort

- Algorithm

1. Define the entire array as the unsorted portion of the array
2. While the unsorted portion of the array has more than one element:
   - ⇒ Find its largest element.
   - ⇒ Swap with last element (assuming their values are different).
   - ⇒ Reduce the size of the unsorted portion of the array by 1.



Before sorting	14	2	10	5	1	3	17	7
After pass 1	14	2	10	5	1	3	7	17
After pass 2	7	2	10	5	1	3	14	17
After pass 3	7	2	3	5	1	10	14	17
After pass 4	1	2	3	5	7	10	14	17

```
// Sort array of integers in ascending order
void select(int data[], // in/output: array
            int size)
{  // input: array size
   int temp;        // for swap
   int max_index;   // index of max value
   for (int rightmost=size-1; rightmost>0; rightmost--){
   //find the largest item in the unsorted portion
  //rightmost is the end point of the unsorted part of array
       max_index = 0; //points the largest element
       for ( int current=1; current<=rightmost; current++)
             if (data[current] > data[max_index])
                    max_index = current;
       //swap the largest item with last item if necessary
       if (data[max_index] > data[rightmost]){
             temp = data[max_index];   // swap
             data[max_index] = data[rightmost];
             data[rightmost] = temp;
       }
   }
}
```

Bubble Sort

- Bubble sort examines the array from start to finish, comparing elements as it goes.
- Any time it finds a larger element before a smaller element, it swaps the two.
- In this way, the larger elements are passed towards the end.
- The largest element of the array therefore "bubbles" to the end of the array.
- Then it repeats the process for the unsorted portion of the array until the whole array is sorted.

Bubble Sort

- Bubble sort works on the same general principle as shaking a soft drink bottle.
- Right after shaking, the contents are a mixture of bubbles and soft drink, distributed randomly.
- Because bubbles are lighter than the soft drink, they rise to the surface, displacing the soft drink downwards.
- This is how bubble sort got its name, because the smaller elements "float" to the top, while

  the larger elements "sink" to the bottom.

Bubble Sort

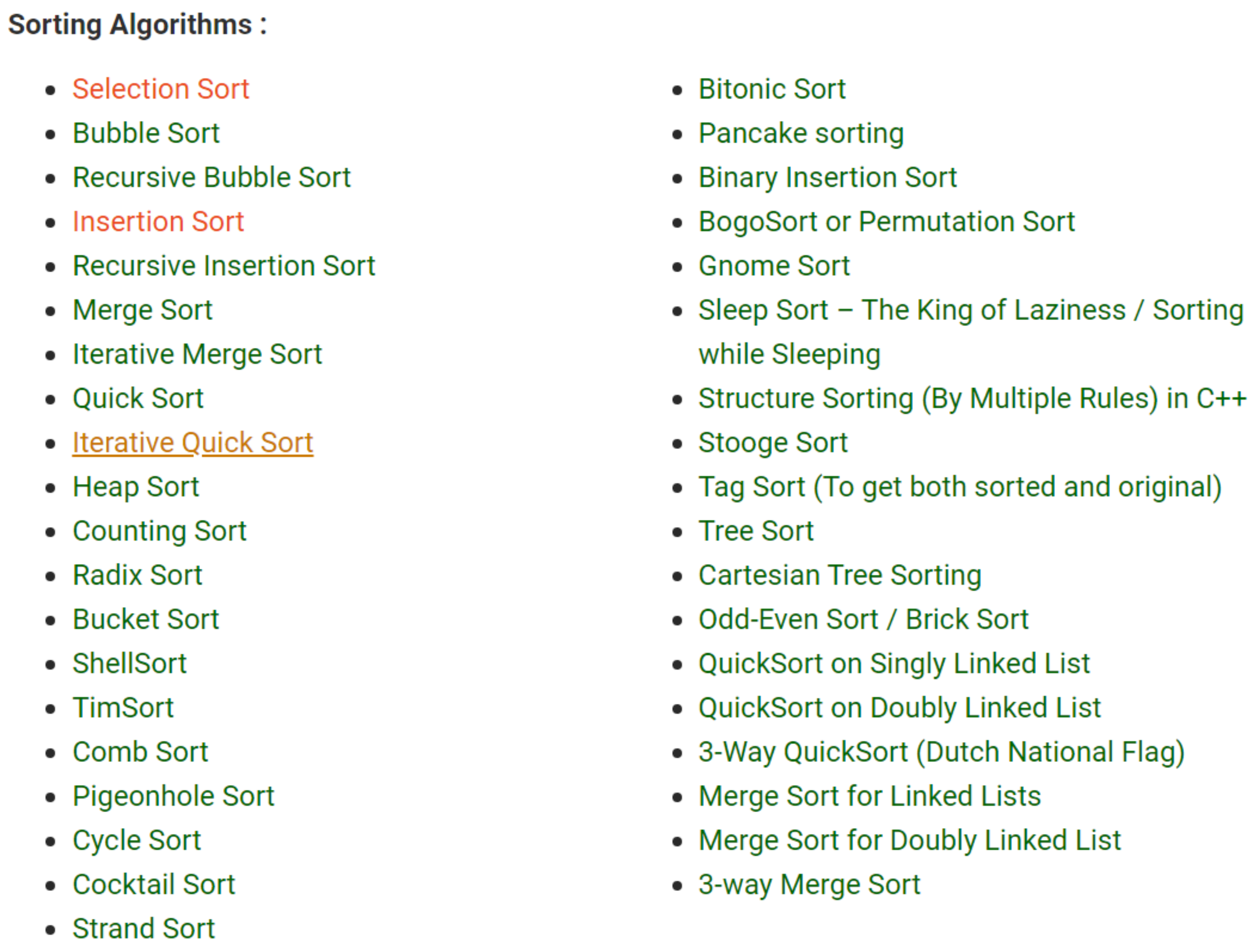
- Algorithm
  - Define the entire array as the unsorted portion of the array.
  - While the unsorted portion of the array has more than one element:
    1. For every element in the unsorted portion, swap with the next neighbor if it is larger than the neighbor.
    2. Reduce the size of the unsorted portion of the array by 1.

Before sorting	14	2	10	5	1	3	17	7
outer=7, inner=0	2	14	10	5	1	3	17	7
outer=7, inner=1	2	10	14	5	1	3	17	7
outer=7, inner=2	2	10	5	14	1	3	17	7
outer=7, inner=3	2	10	5	1	14	3	17	7
outer=7, inner=4	2	10	5	1	3	14	17	7
outer=7, inner=6	2	10	5	1	3	14	7	17
outer=6, inner=1	2	5	10	1	3	14	7	17
outer=6, inner=2	2	5	1	10	3	14	7	17
outer=6, inner=3	2	5	1	3	10	14	7	17
outer=6, inner=5	2	5	1	3	10	7	14	17
outer=5, inner=1	2	1	5	3	10	7	14	17
outer=5, inner=2	2	1	3	5	10	7	14	17
outer=5, inner=4	2	1	3	5	7	10	14	17
outer=4, inner=0	1	2	3	5	7	10	14	17
---								
outer=1, inner=0	1	2	3	5	7	10	14	17

```
//Example: Bobble sort
// Sort an array of integers in ascending order
void bubble(int data[],       // in/output: array
            int size)         // input: array size
{ int temp;       // for swap
  for(int outer=size-1;  outer > 0; outer--){
      for (int inner=0; inner < outer; inner++) {
            // traverse the nested loops
            if ( data[inner] > data[inner+1] ) {
                  // swap current element with next
                  // if the current element is greater
                  temp = data[inner];
                  data[inner] = data[inner+1];
                  data[inner+1] = temp;
            }
      }  // inner for loop
  }  // outer for loop
}
```

Reference reading

- https://www.geeksforgeeks.org/sorting-algorithms/



Sorting Algorithms :

- Selection Sort
- Bubble Sort
- Recursive Bubble Sort
- Insertion Sort
- Recursive Insertion Sort
- Merge Sort
- Iterative Merge Sort
- Quick Sort
- Iterative Quick Sort
- Heap Sort
- Counting Sort
- Radix Sort
- Bucket Sort
- ShellSort
- TimSort
- Comb Sort
- Pigeonhole Sort
- Cycle Sort
- Cocktail Sort
- Strand Sort
- Bitonic Sort
- Pancake sorting
- Binary Insertion Sort
- BogoSort or Permutation Sort
- Gnome Sort
- Sleep Sort – The King of Laziness / Sorting while Sleeping
- Structure Sorting (By Multiple Rules) in C++
- Stooge Sort
- Tag Sort (To get both sorted and original)
- Tree Sort
- Cartesian Tree Sorting
- Odd-Even Sort / Brick Sort
- QuickSort on Singly Linked List
- QuickSort on Doubly Linked List
- 3-Way QuickSort (Dutch National Flag)
- Merge Sort for Linked Lists
- Merge Sort for Doubly Linked List
- 3-way Merge Sort