COMSW 1003-1

Introduction to Computer Programming in C

Lecture 21

Spring 2011

Instructor: Michele Merler

http://www1.cs.columbia.edu/~mmerler/comsw1003-1.html

Big–O : Relationship among common cases

$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(a^n)$

<u>Example</u> : big-O when a function is the *sum of several statements*

```
int i=0;
for(i=0 ; i < n; i++){
    for(j=0 ; j < n; j++){
        if( (i!=j) && arr[i] == arr[j])
            dup[i][j] = 1;
    }
}
```

$RT = O(4n^2+n) = O(n^2)$

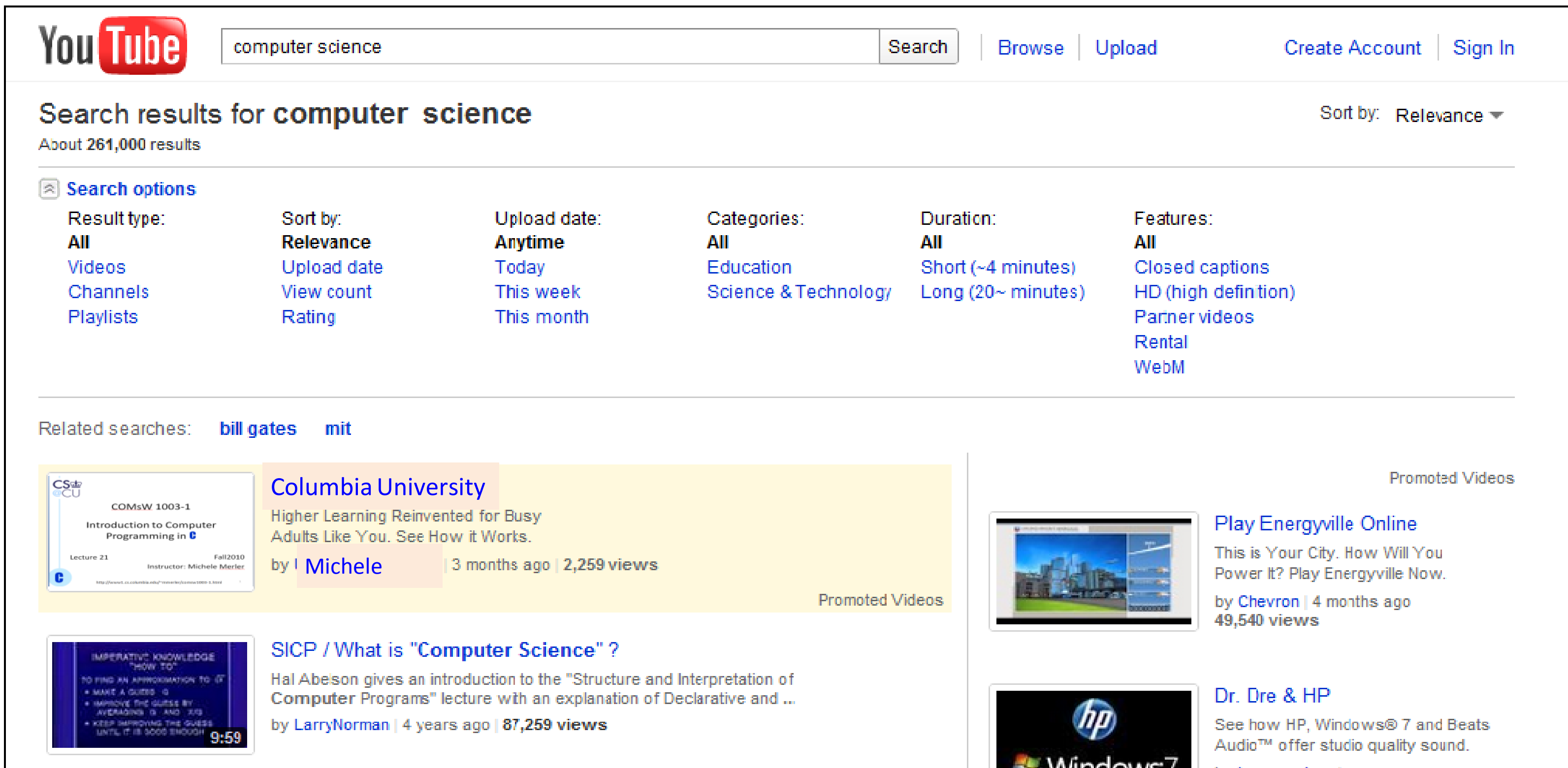
increment `i`
increment `j`
check `i!=j`
check `arr[i]==arr[j]`
`dup[i][j] = 1`

Longest operation dominates (worst case)

Sorting

Sorting

- Given a set of N elements, put them in order according to some **criteria** (alphabetical, relevance, date, smallest to largest, etc.)
- One of the most studied problems in Computer Science
- Everybody uses it every day



Sorting

- Given a set of N elements, put them in order according to some criteria

- Compare pairs of elements

- Many algorithms, some of the most famous are:
  - Bubble sort
  - Selection sort
  - Insertion sort
  - Merge sort
  - Counting sort

- In following examples, we'll see smallest to biggest sorting

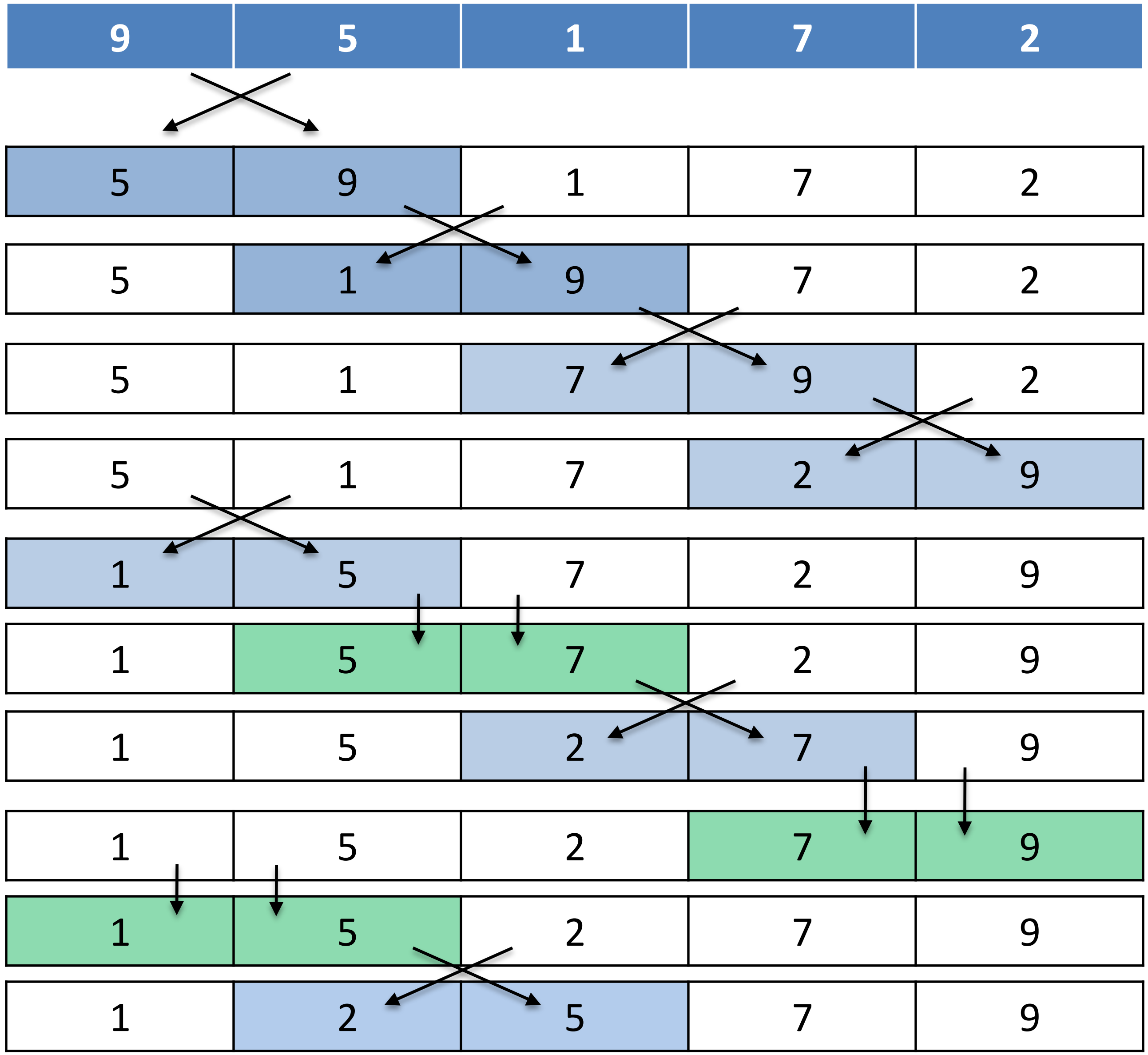
Bubble Sort

1. Start with the first two elements
2. If first element > second element
    - Swap
3. Iterate for all following pairs
4. Repeat steps 1 to 3 until no swaps are necessary

Complexity = $O(n^2)$

Count number of comparisons and swaps

Bubble Sort


9 5 1 7 2
5 9 1 7 2
5 1 9 7 2
5 1 7 9 2
5 1 7 2 9
1 5 7 2 9
1 5 7 2 9
1 5 2 7 9
1 5 2 7 9
1 5 2 7 9
1 2 5 7 9

Bubble Sort

9	5	1	7	2	
5	9	1	7	2	
5	1	9	7	2	
5	1	7	9	2	n-1 checks
5	1	7	2	9	
1	5	7	2	9	
1	5	7	2	9	
1	5	2	7	9	n-1 checks
1	5	2	7	9	
1	5	2	7	9	⋮
1	2	5	7	9	

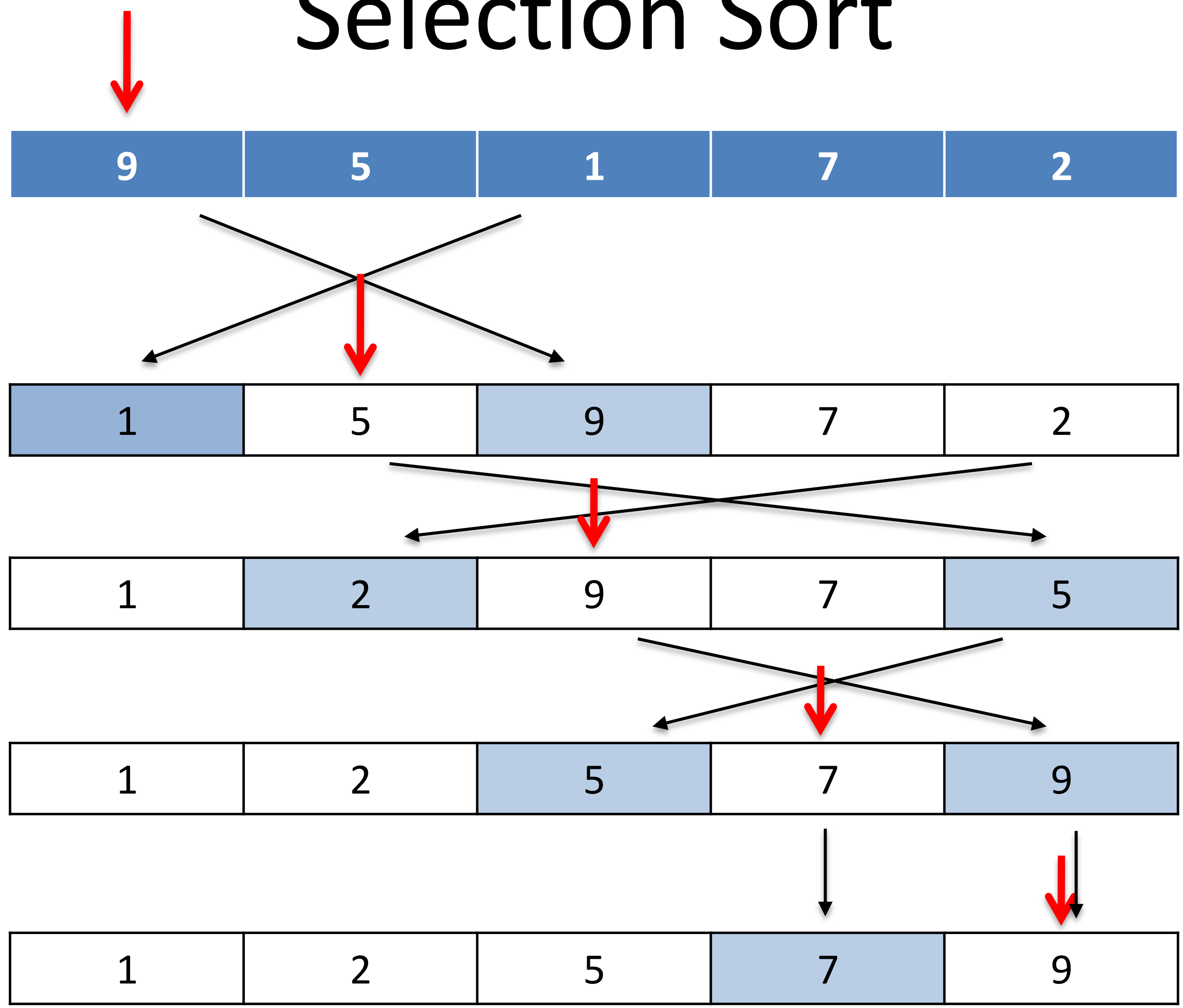
Selection Sort

- Smarter algorithm, but same complexity (worst case)

1. Find smallest unsorted element
2. Swap with first unsorted element
3. Repeat steps 1 and 2 until no more unsorted elements

Complexity = $O(n^2)$

Selection Sort



Insertion Sort

- Main idea: keep 2 separate sets (one sorted, one unsorted), and move elements from unsorted to sorted set one at a time
- Better performance in case many elements are already sorted, quadratic in worst case

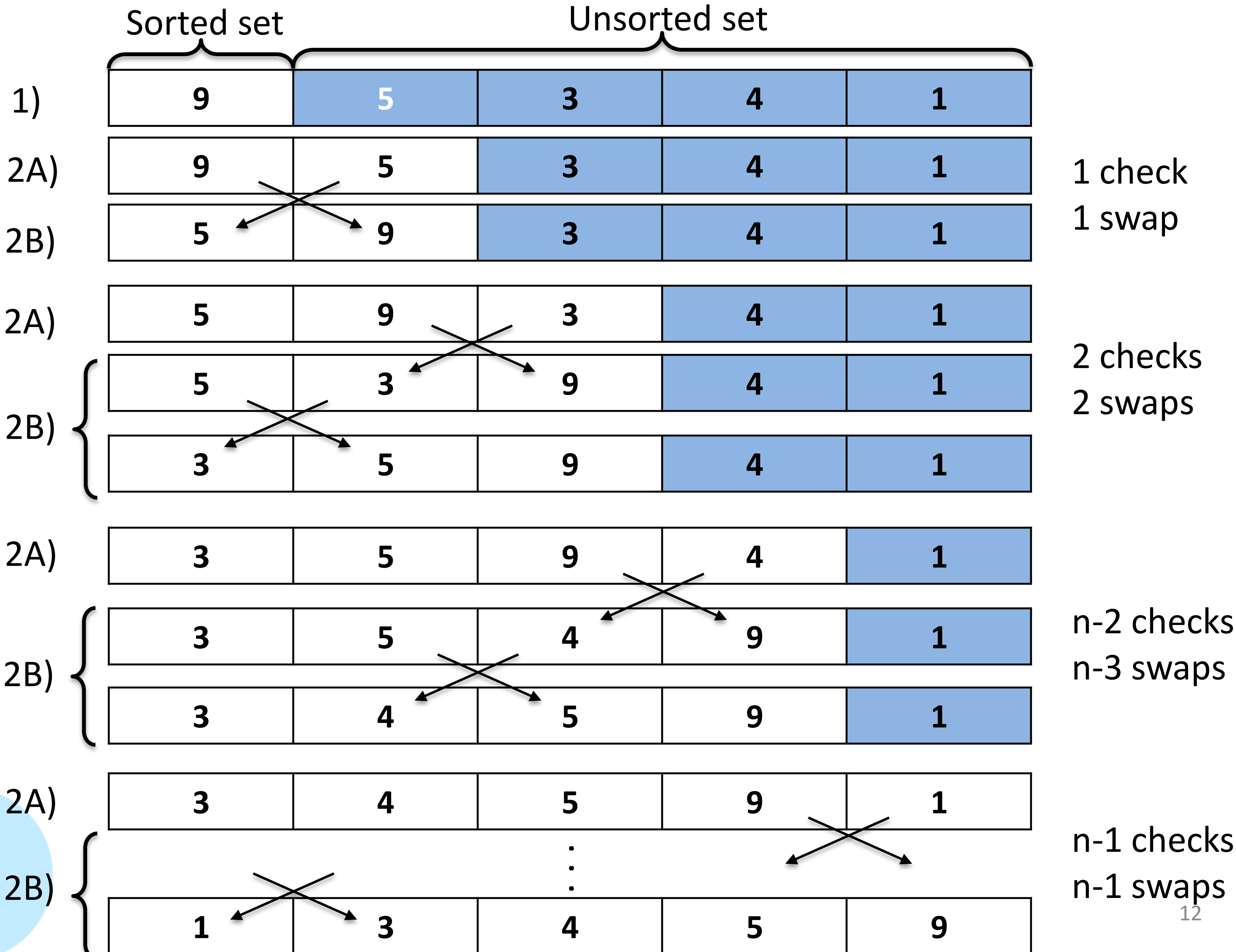
1) Initialize 2 sets
   - One set of sorted elements (contains only first element in the array)
   - One set of unsorted elements (all the other elements in the array)

2) A) Take first element in unsorted set and
   B) Insert it into sorted set at proper position

3) Repeat steps 2A) and 2B) until unsorted set is empty

Complexity = $O(n^2)$

Insertion sort



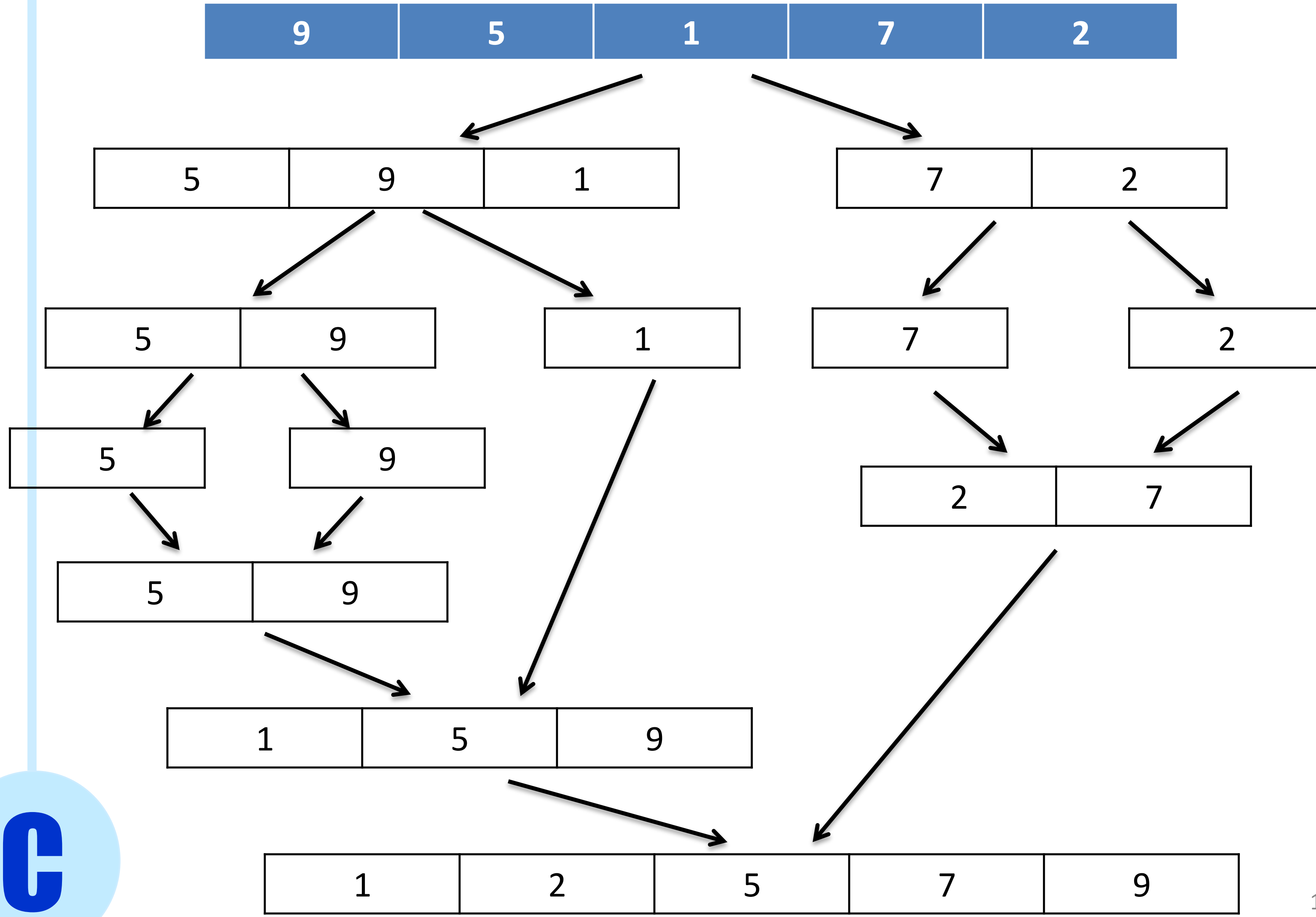
Merge Sort

- One of the fastest algorithms, divide and conquer principle
- Uses recursion
- Sorting small sets is faster than sorting large sets
- Merging 2 sets into a sorted union is faster if the sets are already sorted

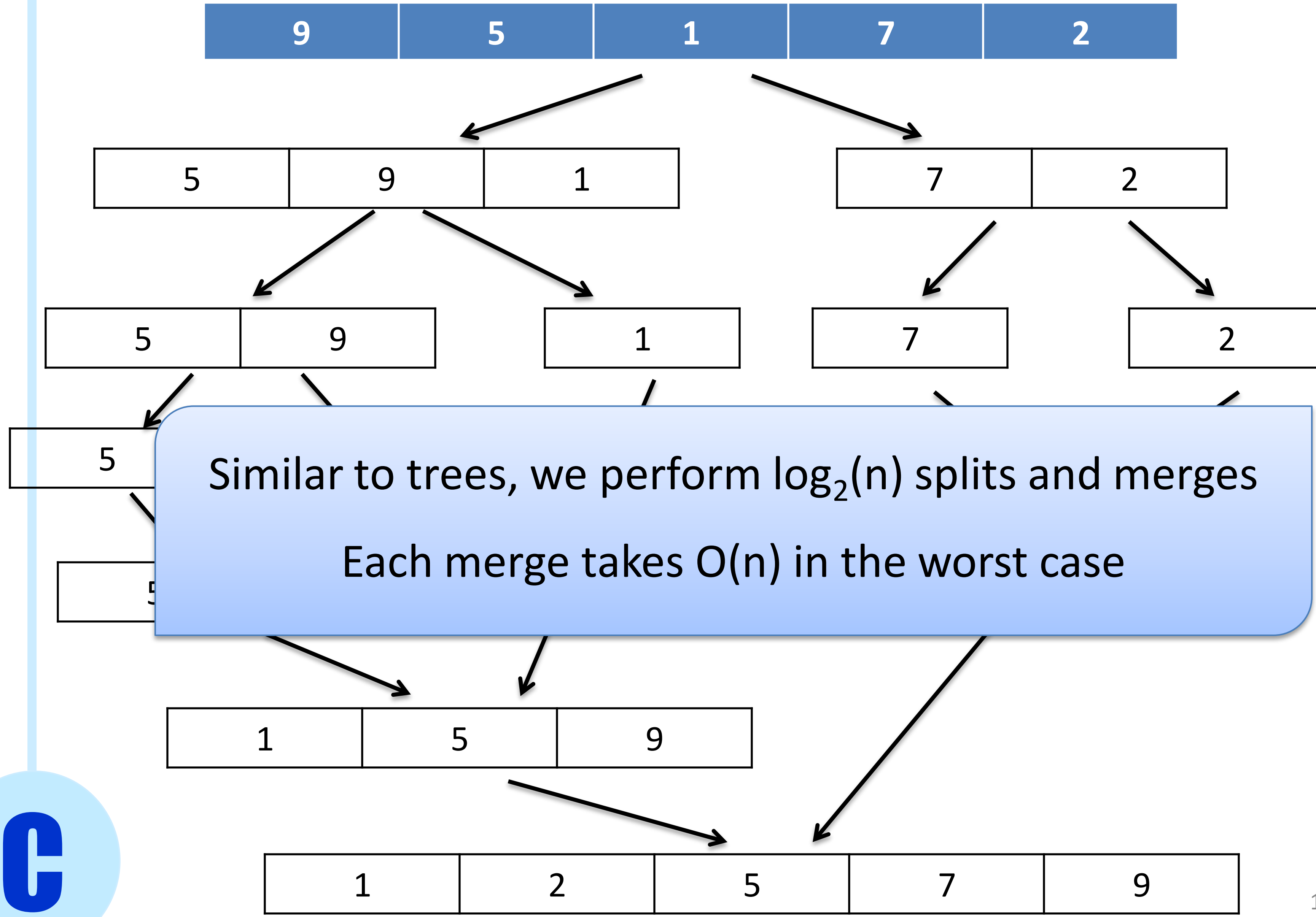
1. If set H has 1 element, stop
2. else
   - Split set into 2 halves H1 and H2 of (approximately) same size
   - Sort H1 and H2 with merge sort — recursion
   - Merge the sorted H1 and H2 into a sorted set

Complexity = O(n log(n))

Merge Sort



Merge Sort



Similar to trees, we perform $\log_2(n)$ splits and merges

Each merge takes O(n) in the worst case

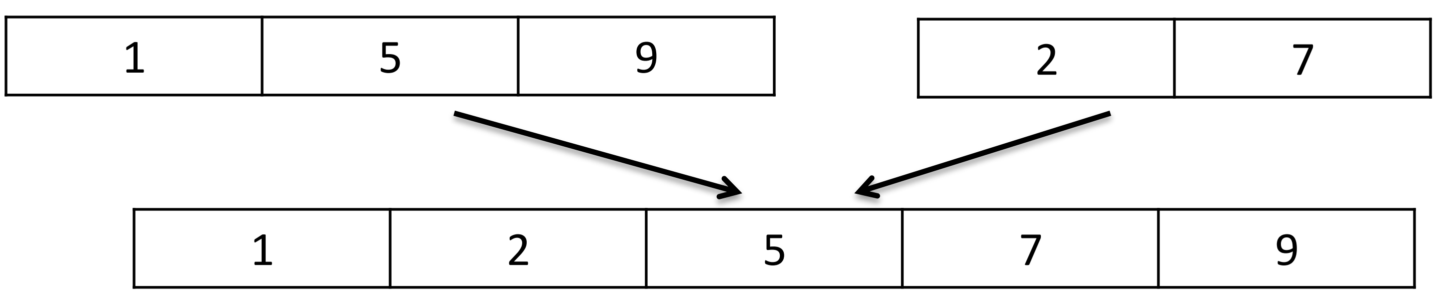
Merge Sort

Similar to trees, we perform $\log_2(n)$ splits and merges
Each merge takes O(n) in the worst case

Merge routine:

Given H1 and H2 of size n1 and n2 respectively, create H of length n = n1 + n2

```
int c1=0, c2=0;
for (i=0; i<n; i++){
    if( (c1<n1) && ((H1[c1] < H2[c2]) || (c2==n2)) ){
            H[i] = H1[c1];
            c1++;
    }
    else{
        H[i] = H2[c2];
        c2++;
    }
}
```



Counting sort

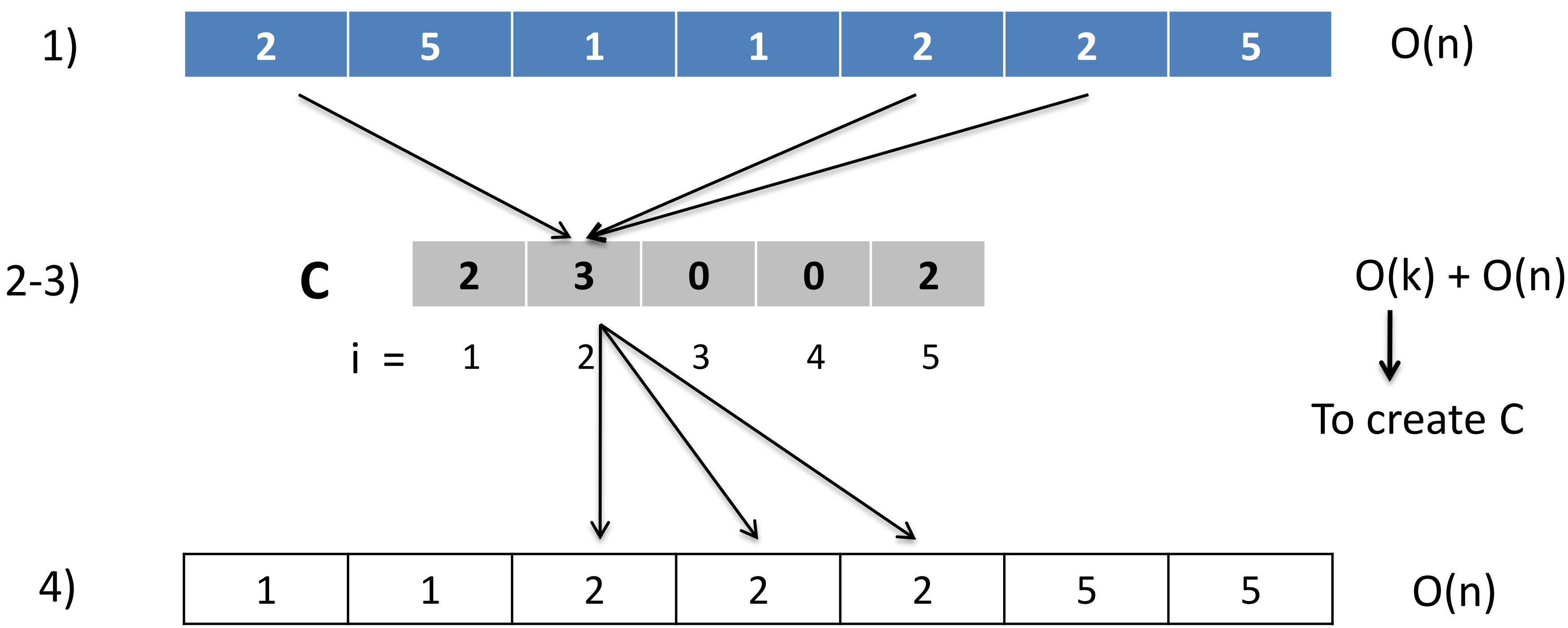
- Intuition: exploit range *k* of values in set
- Efficient if *k* is not much larger than *n*

1. Find biggest and smallest values in the set
   (k = maxVal – minVal+1)
2. Create an array C of k elements
3. Count occurrences *C(i)* of each value *i* in the set
4. Fill ordered set by inserting *C(i)* elements of value *i*, for each value in range *k*

Complexity = O(n + k)

Counting sort

Example: range of values in set is [1, 5], k = 5



Homework 4 Solution