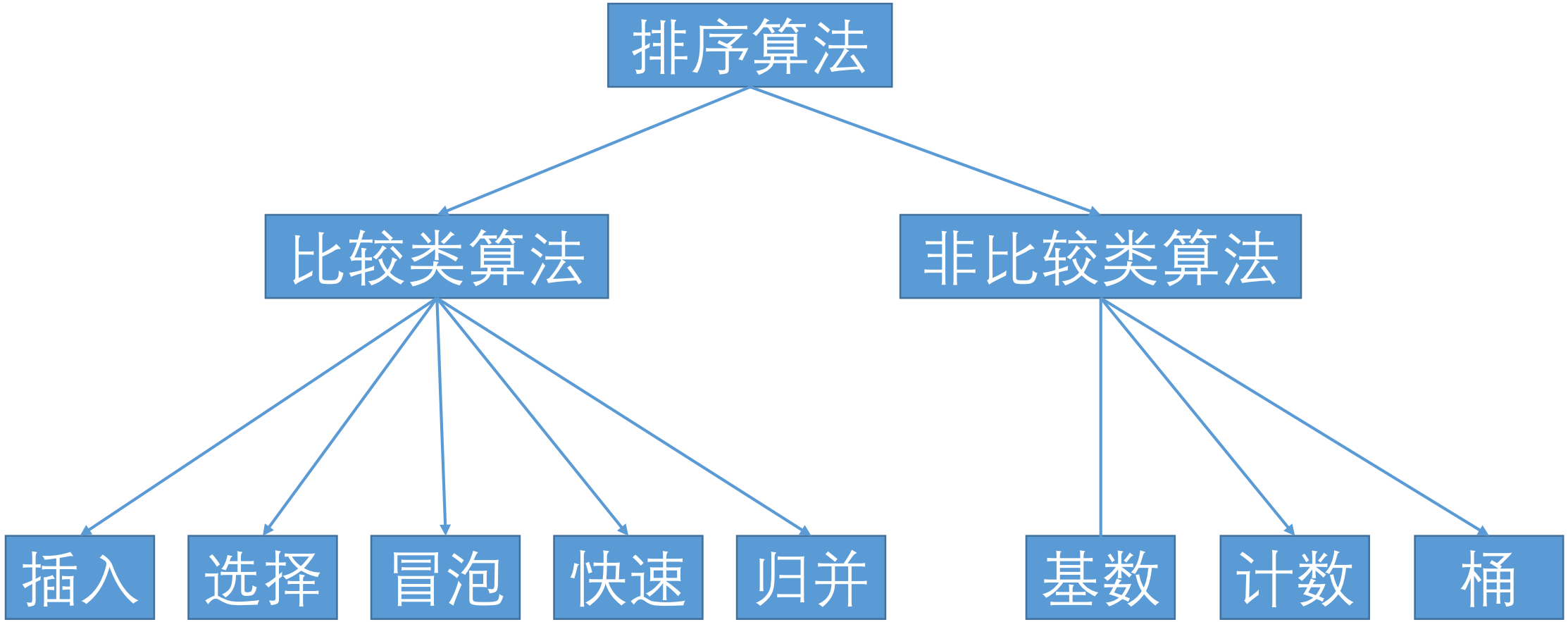
程序设计
Programming

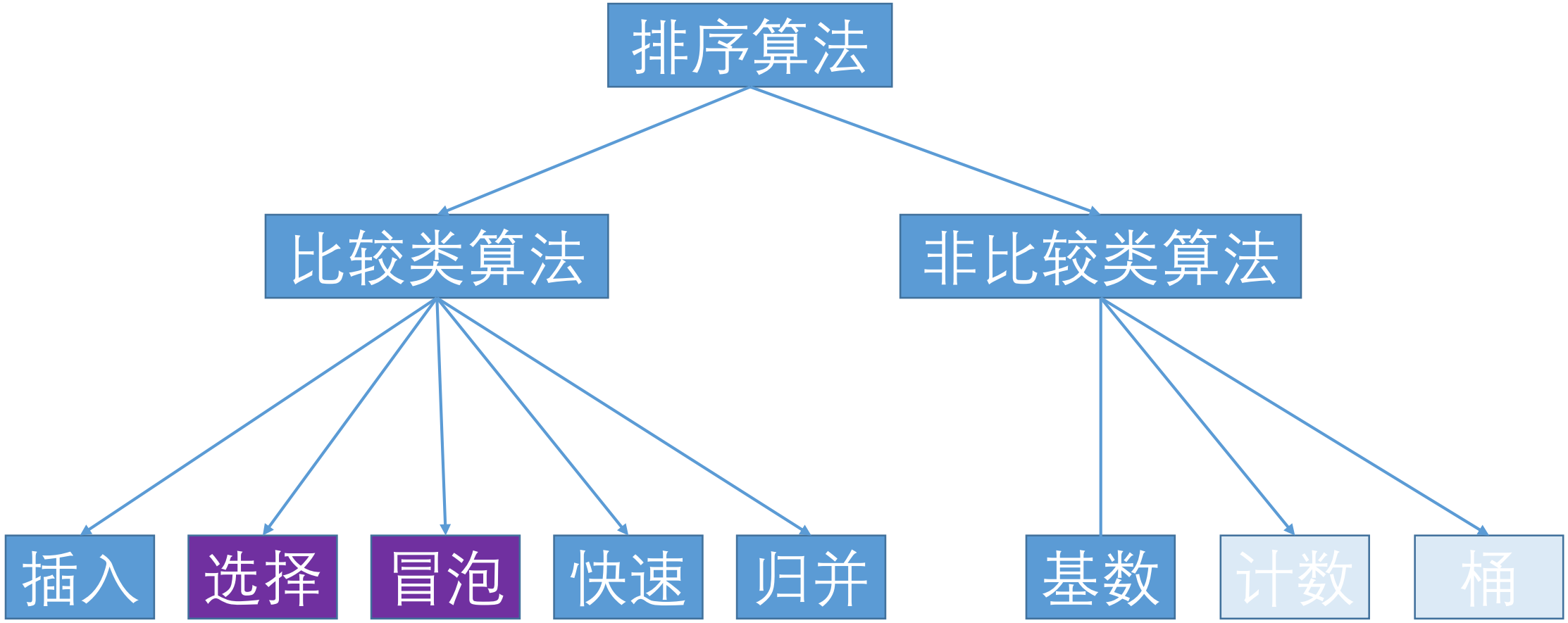
Lecture 13: 排序

经典的排序算法



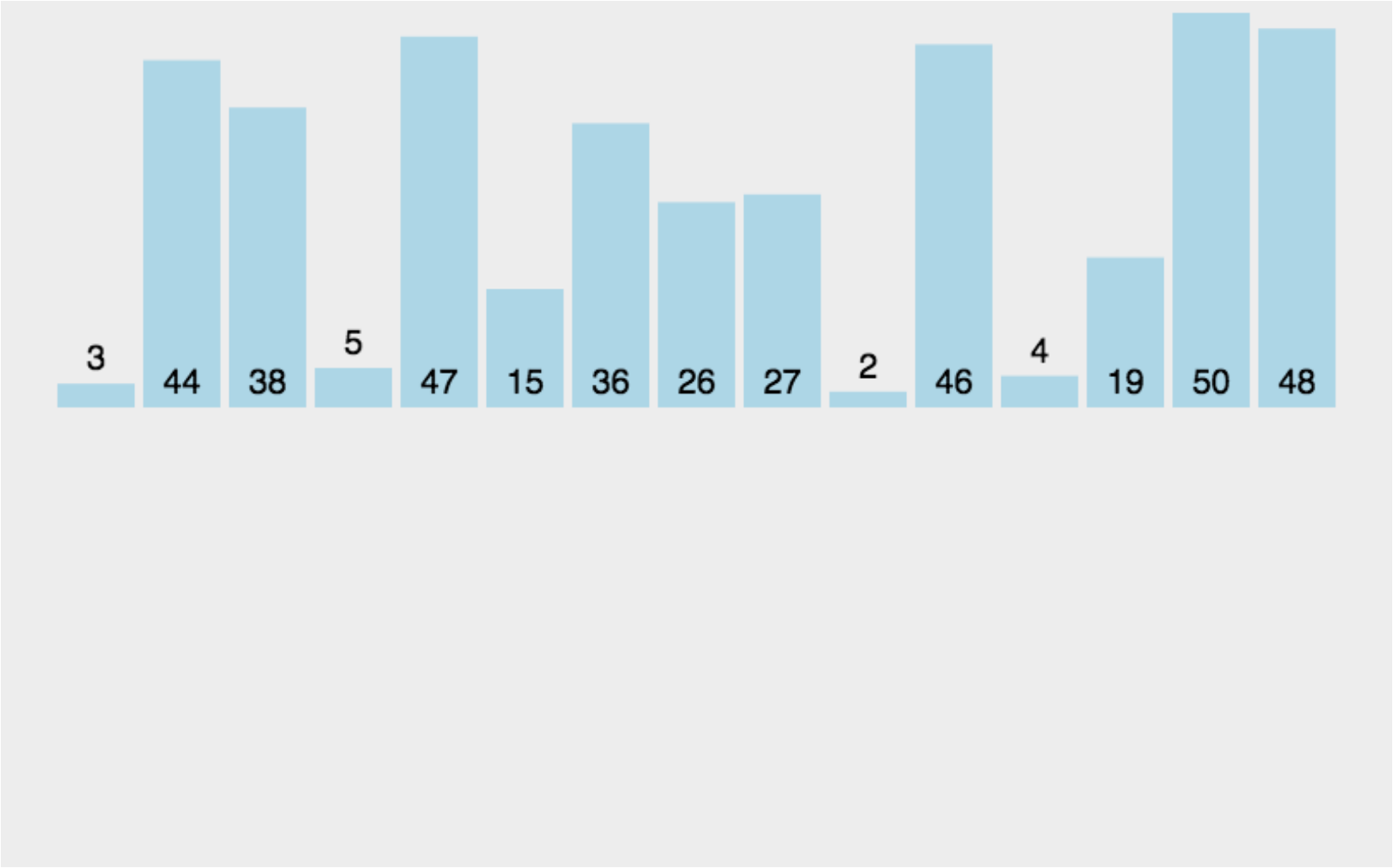
经典的排序算法

假设都采取从小到大排序



P155, 例7-5　P191, 例8-5

插入排序（Insertion Sort）



① 第1个元素被认为已经有序

② 从第n=2个元素开始，每次将其从后向前，与已经有序的n-1个元素进行比较，直到找到一个元素小于或者等于第n个元素

③ 在该元素后面插入第n个元素

④ 重复步骤2和3直至所有元素排序完成

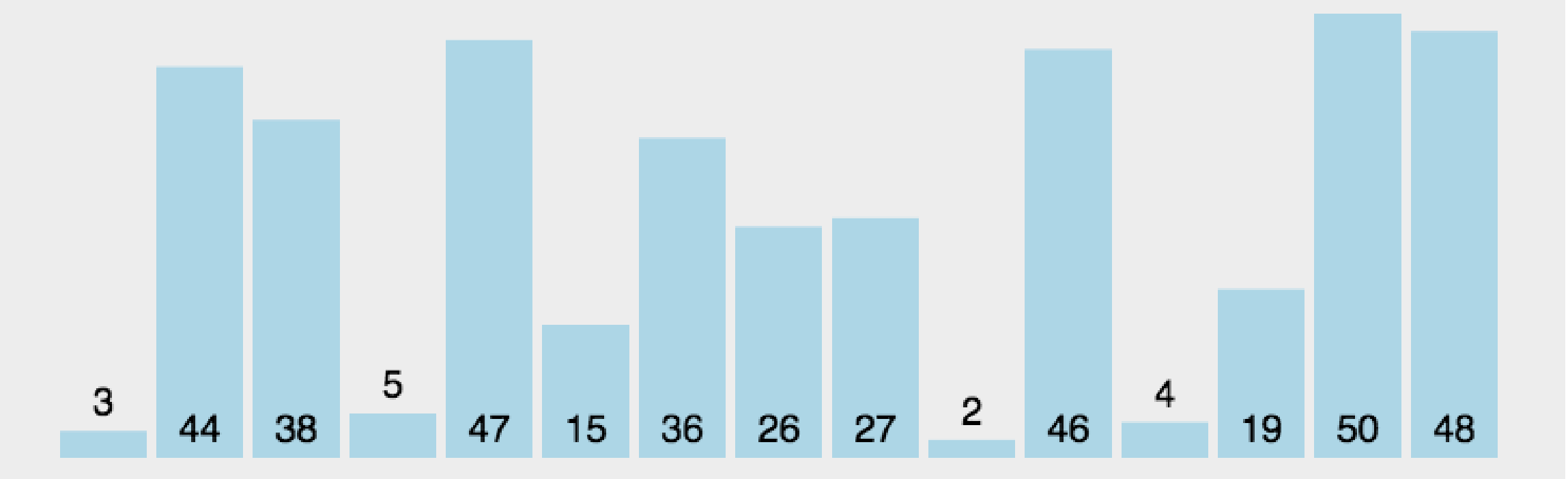
插入排序代码

```
void insertionSort(int array[], int length)
{
    for(int i = 1; i < length; i++)  //从第2个元素开始，逐个检查
    {
        int current = array[i];
        int j = i-1;

        array[j+1] = current;    //第i个元素插入相应的位置
    }
}
```

插入排序代码

```
void insertionSort(int array[], int length)
{
    for(int i = 1; i < length; i++)  //从第2个元素开始，逐个检查
    {
        int current = array[i];
        int j = i-1;
        for(; j >= 0; j--)   //取出第i个元素，从后往前依次与前面i-1个元素比较
        {
            if(array[j] <= current)  //找到小于等于第i个元素的元素
                break;
            array[j+1] = array[j];   //每个大于第i个元素的元素，往后挪一位
        }
        array[j+1] = current;   //第i个元素插入相应的位置
    }
}
```

选择排序（Selection Sort）



① 有序序列为空，无序序列为1...n

② 从无序序列中找到最小的元素，并和无序序列中第一个元素交换位置

③ 有序序列长度加1，无序序列长度减1

④ 重复步骤2和3直至所有元素排序完成

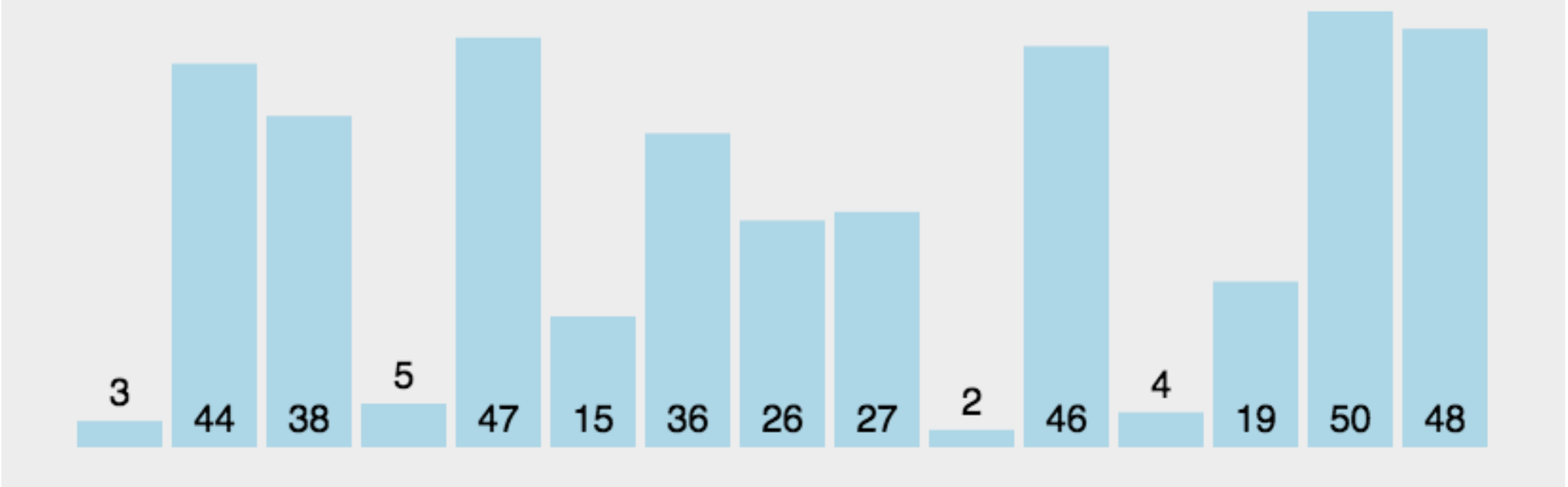
选择排序代码

```
void selectionSort(int array[], int length)
{
    for(int i = 0; i < length-1; i++)  //每次迭代都从无序序列的第一个元素开始检查
    {
        int min_pos = i;









        if(min_pos != i)    //交换无序序列第一个元素和最小元素的位置
        {
            int tmp = array[i];
            array[i] = array[min_pos];
            array[min_pos] = tmp;
        }
    }
}
```

选择排序代码

```
void selectionSort(int array[], int length)
{
    for(int i = 0; i < length-1; i++)  //每次迭代都从无序序列的第一个元素开始检查
    {
        int min_pos = i;
        for(int j = i+1; j < length; j++)   //寻找当前无序序列中的最小元素
        {
            if(array[j] < array[min_pos])
                min_pos = j;
        }
        if(min_pos != i)    //交换无序序列第一个元素和最小元素的位置
        {
            int tmp = array[i];
            array[i] = array[min_pos];
            array[min_pos] = tmp;
        }
    }
}
```

冒泡排序（Bubble Sort）



① 无序序列为1...n，有序序列为空

② 从无序序列第一个元素开始直至倒数第二个元素，依次比较相邻元素；如果大的元素在前则交换位置

③ 第i趟后，无序序列变为1...n-i，有序序列变为n-i+1...n

④ 重复步骤2和3直至所有元素都有序

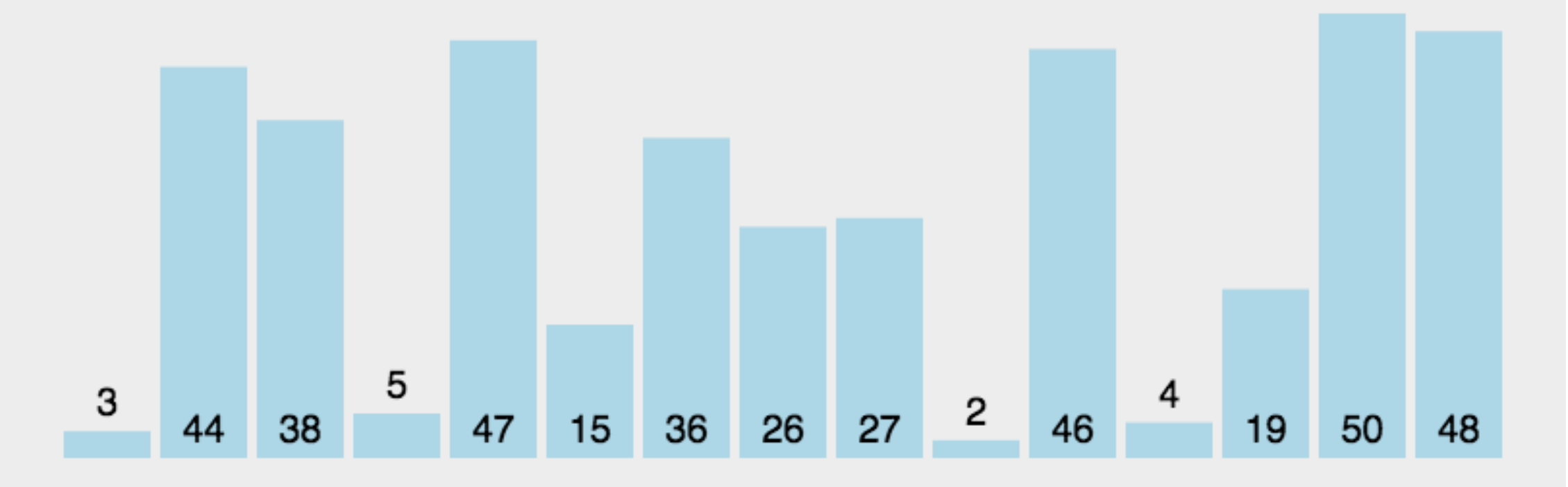
冒泡排序代码

```
void bubbleSort(int array[], int length)
{


}
```

冒泡排序代码

```
void bubbleSort(int array[], int length)
{
    for(int i = 0; i < length-1; i++)   //length-1趟比较
        for(int j = 0; j < length-i-1; j++)  //从头开始比较相邻元素
        {
            if(array[j] > array[j+1])   //交换位置
            {
                int tmp = array[j];
                array[j] = array[j+1];
                array[j+1] = tmp;
            }
        }
}
```

快速排序（Quick Sort）



① 选择一个元素作为“基准”（pivot），通常可以选第一个元素

② 将小于等于基准的元素放到基准左侧，大于基准的元素放到基准右侧。这一步称为“分区”（partition）

③ 使用递归的方式，对基准左右的序列重复执行步骤1和2，直至所有元素排列有序

快速排序代码：quickSort()

```
void quickSort(int array[], int left, int right)
{
    if                              //递归出口
        return;

                                                //分区函数
                                        //递归调用
                                        //递归调用
}
```

快速排序代码：quickSort()

```
void quickSort(int array[], int left, int right)
{
    if(left >= right)        //递归出口
        return;
    int pivot = partition(array, left, right);    //分区函数
    quickSort(array, left, pivot-1);      //递归调用
    quickSort(array, pivot+1, right);     //递归调用
}
```

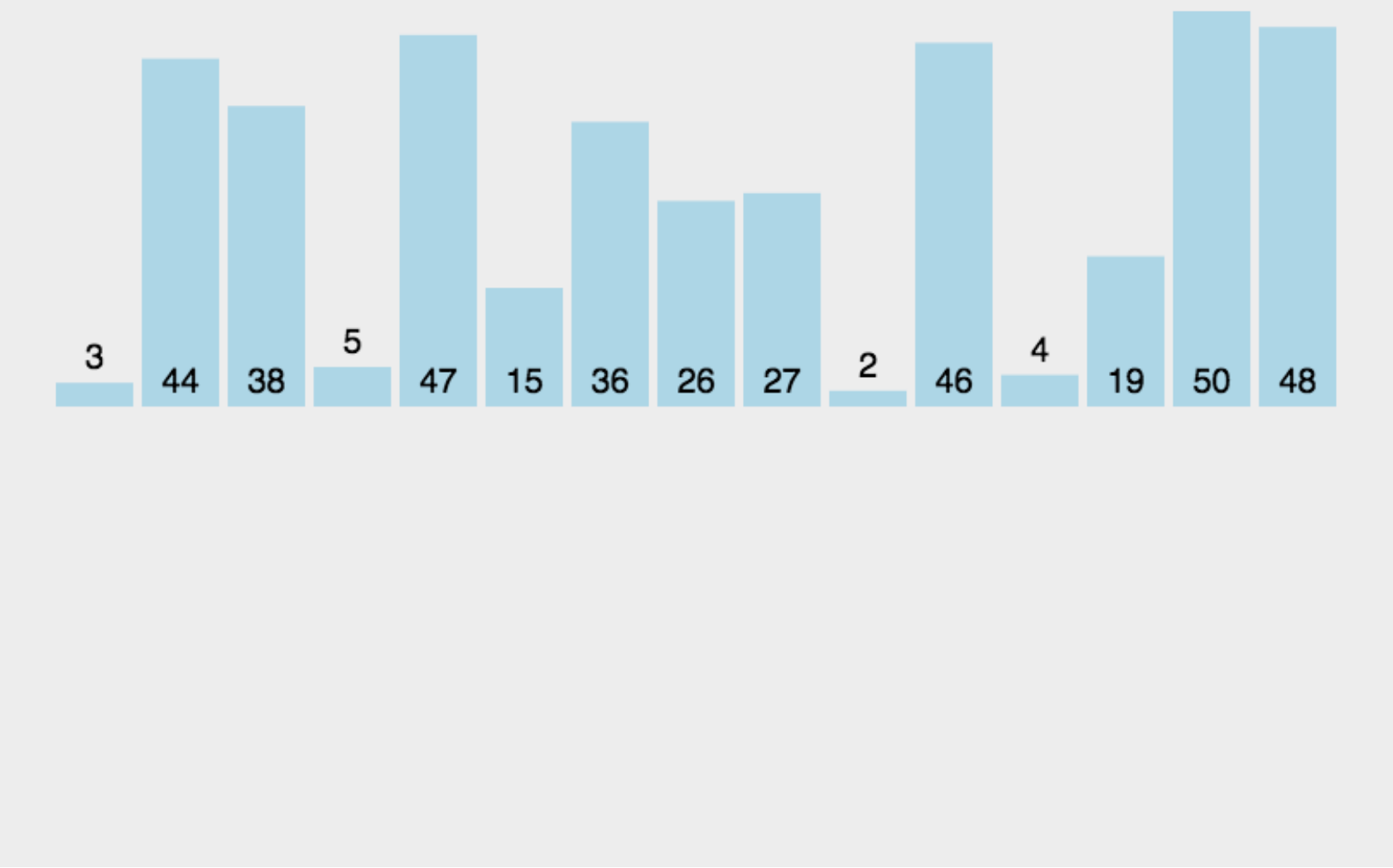
快速排序代码：partition()

```
int partition(int array[], int left, int right)
{
    int pivot = left;
    int current = pivot+1;
    for(int i = left+1; i <= right; i++)   //从第二个元素开始，依次与pivot比较
    {
        if (array[i] < array[pivot])    //如果小于pivot，则和current位置元素交换位置
        {



        }
    }
    int tmp = array[pivot];        //交换pivot和current前面那个元素的位置
    array[pivot] = array[current-1];
    array[current-1] = tmp;
    return current-1;
}
```

快速排序代码：partition()

```
int partition(int array[], int left, int right)
{
    int pivot = left;
    int current = pivot+1;
    for(int i = left+1; i <= right; i++)   //从第二个元素开始，依次与pivot比较
    {
        if (array[i] < array[pivot])    //如果小于pivot，则和current位置元素交换位置
        {
            int tmp = array[current];
            array[current] = array[i];
            array[i] = tmp;
            current++;
        }
    }
    int tmp = array[pivot];        //交换pivot和current前面那个元素的位置
    array[pivot] = array[current-1];
    array[current-1] = tmp;
    return current-1;
}
```

归并排序（Merge Sort）



① 将无序序列对半分成左右两个子序列

② 采用递归方式不断将无序子序列对半分成更小的子序列，直至子序列长度为1

③ 将步骤2中生成的子序列两两合并为有序序列，直至最终合并成一个有序序列

归并排序代码：mergeSort()

```
void mergeSort(int array[], int length)
{
    if (length < 2)  //出口，只有一个元素
        return;
    int middle = length/2;
    int la = middle;
    int lb = length-middle;
    int a[la], b[lb];  //将array拆分成a和b两个子序列

                                        //合并排序已经有序的两个子序列
}
```

归并排序代码：mergeSort()

```
void mergeSort(int array[], int length)
{
    if (length < 2)  //出口，只有一个元素
        return;
    int middle = length/2;
    int la = middle;
    int lb = length-middle;
    int a[la], b[lb];  //将array拆分成a和b两个子序列
    for(int i = 0; i < la; i++)
        a[i] = array[i];
    for(int i = 0; i < lb; i++)
        b[i] = array[i+middle];
    mergeSort(a, la);  //递归拆分
    mergeSort(b, lb);
    merge(a, b, la, lb, array);  //合并排序已经有序的两个子序列
}
```

归并排序代码：merge()

```
void merge(int a[], int b[], int la, int lb, int c[])
{
    int i=0, j=0, k=0;



}
```

归并排序代码：merge()

```
void merge(int a[], int b[], int la, int lb, int c[])
{
    int i=0, j=0, k=0;
    for(; i < la && j < lb; k++)
    {
        c[k] = a[i] < b[j] ? a[i++] : b[j++];
    }
    while(i < la) c[k++] = a[i++];
    while(j < lb) c[k++] = b[j++];
}
```

基数排序（Radix Sort）

3	44	38	5	47	15	36	26	27	2	46	4	19	50	48

低位基数排序

① 取得数组中最大数的位数

② 根据最低位的哈希值，依次将无序序列中的元素放入相应的桶中

③ 按顺序从桶中取出元素重新组成序列，然后根据次低位重新进行哈希

④ 重复步骤2和3直到哈希至最高位

基数排序代码

```
void radixSort(int array[], int length, int digits)
{
    int bucket[10][length];  //10个桶
    int bsize[10];  //每个桶中元素个数
    for(int i = 0, dev = 1; i < digits; i++, dev *= 10)  //从低位到高位进行hash
    {
        for(int j = 0; j < 10; j++)
            bsize[j] = 0;
                                        //将array中的元素放入相应桶中
                                                    //顺序从桶中取出元素重新组合成array
    }
}
```

基数排序代码

```
void radixSort(int array[], int length, int digits)
{
    int bucket[10][length];  //10个桶
    int bsize[10];  //每个桶中元素个数
    for(int i = 0, dev = 1; i < digits; i++, dev *= 10)  //从低位到高位进行hash
    {
        for(int j = 0; j < 10; j++)
            bsize[j] = 0;
        for(int j = 0; j < length; j++)  //将array中的元素放入相应桶中
        {
            int hash = (array[j] / dev) % 10;
            bucket[hash][bsize[hash]++] = array[j];
        }
        int ii = 0;
        for(int j = 0; j < 10; j++)
            for(int k = 0; k < bsize[j]; k++)   //顺序从桶中取出元素重新组合成array
                array[ii++] = bucket[j][k];
    }
}
```

排序算法平均时间复杂度

算法	时间复杂度（平均）
插入排序	$O(n^2)$
选择排序	$O(n^2)$
冒泡排序	$O(n^2)$
快速排序	$O(n\log n)$
归并排序	$O(n\log n)$
基数排序	$O(nk)$

http://106.75.225.141/xuesong/programming-course/tree/master/sort