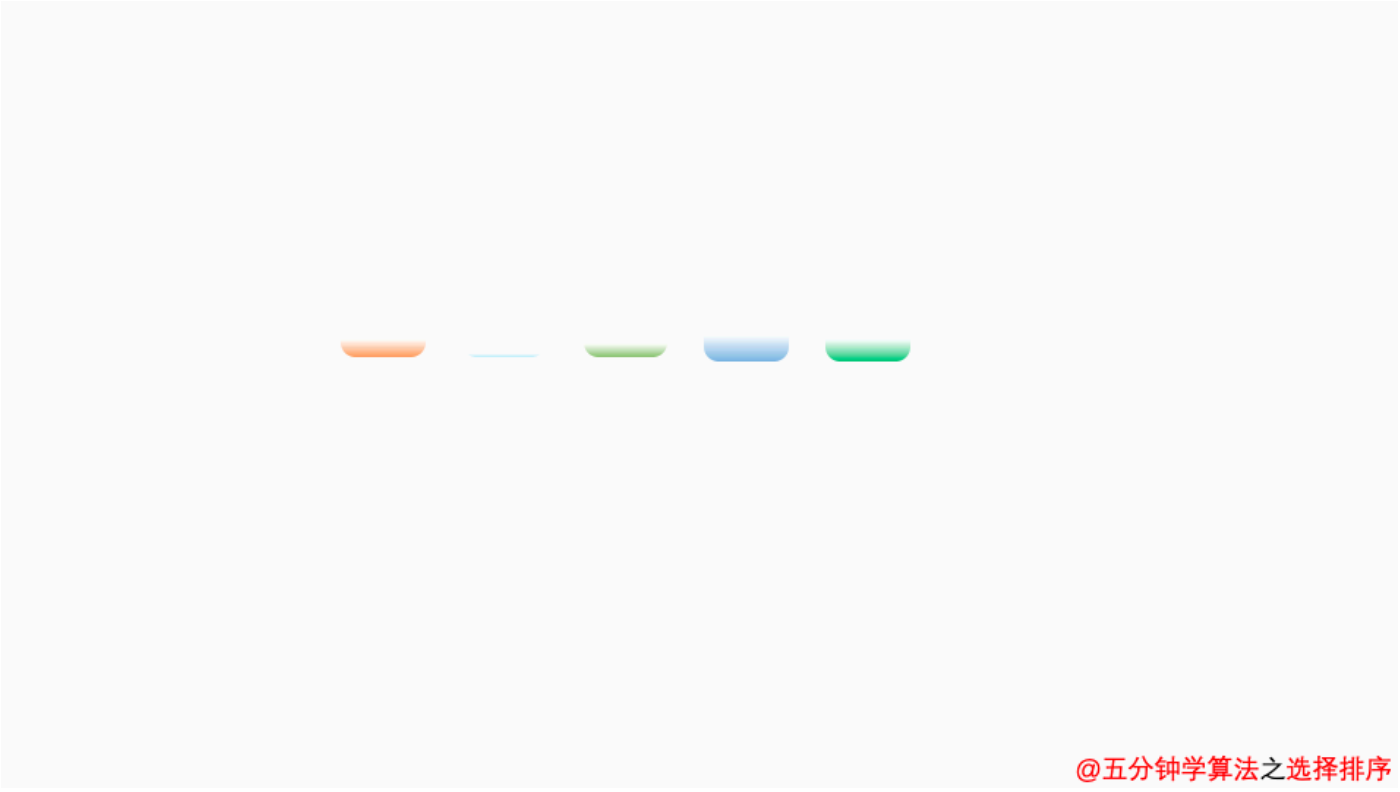
排序



@五分钟学算法之选择排序

选择排序

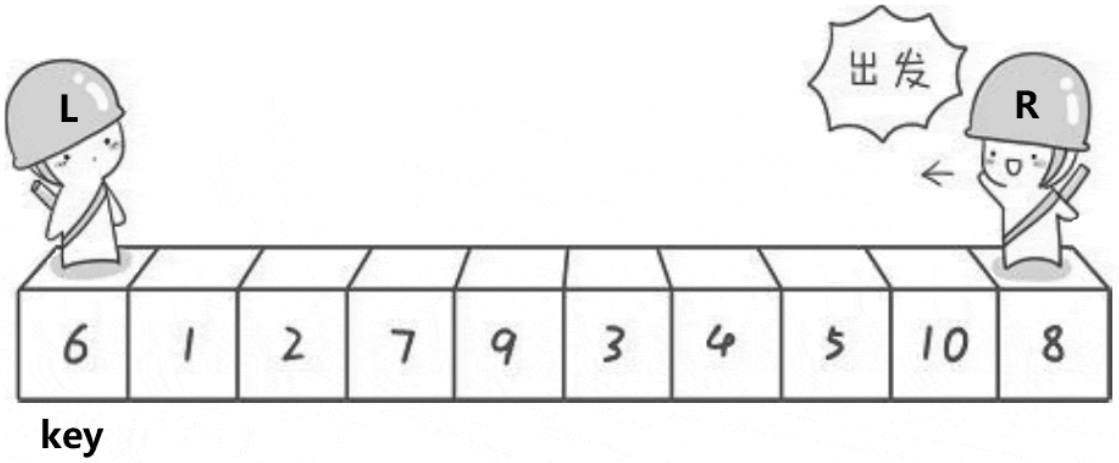
```
int num = { 2, 56, 23, 85, 61, 45, 32, 14, 75, 64, 24, 26, 28, 51 };
public static int[] SelectionSort(int[] arr)
{
    int n = arr.Length;
    for (int i = 0; i < n; i++)
    {
        int min_idx = i;
        for (int j = i+1; j < n; j++)
        {
            if (arr[j] < arr[min_idx])
            {
                min_idx = j;
            }
        }
        int temp = arr[min_idx];
        arr[min_idx] = arr[i];
        arr[i] = temp;
    }
    return arr;
}
```

@五分钟学算法之插入排序

插入排序

```
int num = { 2, 56, 23, 85, 61, 45, 32, 14, 75, 64, 24, 26, 28, 51 };
public static int[] InsertionSort(int[] arr)
{
    int n = arr.Length;
    for (int i = 1; i < n; i++)
    {
        for (int j = i; j > 0 && arr[j] < arr[j - 1]; j--)
        {
            Swap(arr, j, j - 1);
        }
    }
    return arr;
}

private static void Swap(int[] arr, int i, int j)
{
    int tmp = arr[i];
    arr[i] = arr[j];
    arr[j] = tmp;
}

```

@五分钟学算法之快速排序



快速排序

```
先从数列中取出一个元素作为基准数
扫描数列，将比基准数小的元素全部放到它的左边，大于或等于基准数的元素全部放到它的右边，得到左右两个区间
在对左右区间重复第二步，直到元素少于两个

void main()
{
int arr[9]={5,4,7,2,9,1,77}
quick-sort(arr,1,7);
}
//分区
```

```
void quick-sort(int arr[],int left,int right)
{
 if(left<right)
 {
  int mid=get-mid(arr, left, right);
  quick-sort(arr, left, mid-1);
  quick-sort(arr,mid+1 , right);
 }
}
//排序
void get-mid(int arr[],int left, int right)
{
 int pivot=arr[left];
 while(left<right)
 {
  while(arr[right]>pivot)  right--;
  arr[left]=arr[right];
  while(arr[left]<pivot)  left++;
  arr[right]=arr[left];
 }
 arr[left]=pivot;//pivot就是第一轮排序后的中间数字，left就是这个数的下标，然后这个下标就作为中间数进行第二轮排序
 return left;
}
```

```
using System;

public class QuickSort {
    public static void Sort(int[] arr, int left, int right) {
        if (left < right) {
            int pivot = Partition(arr, left, right);
            if (pivot > 1) {
                Sort(arr, left, pivot - 1);
            }
            if (pivot + 1 < right) {
                Sort(arr, pivot + 1, right);
            }
        }
    }

    private static int Partition(int[] arr, int left, int right) {
        int pivot = arr[left];
        while (true)
        {
```

```
        while (arr[left] < pivot)
        {
            left++;
        }
        while (arr[right] > pivot)
        {
            right--;
        }
        if (left < right) {
            int temp = arr[left];
            arr[left] = arr[right];
            arr[right] = temp;
        }
        else
        {
            return right;
        }
    }
  }
}

public class Program {
  public static void Main() {
    int[] arr = { 4, 2, 6, 1, 5, 3 };
    QuickSort.Sort(arr, 0, arr.Length - 1);

    foreach (int i in arr) {
      Console.Write(i + " ");
    }
  }
}


```

归并排序

```
int main(int argc,char const *argv[])
{
 int arr[]={9,5,2,7,12,4,3,1,11};
 int n=9;
 merge-sort(arr,9);
}
void merge-sort(int arr[],int n)
{
 //分配一个临时数组
 int *tempArr=(int*)malloc(n*sizeof(int));
 if(tempArr)//辅助数组分配成功
 {
```

```
    mesort(arr,tempArr,0,n-1);
    free(tempArr);//释放空间
  }
}

//归并
void mesort(int arr[],int tempArr[],int left,int right)
{
  //如果只有一个元素，不需要划分
  if(left<right)
  {
    int mid=left+(left-right)/2;//找中间点
    mesort(arr,tempArr,left,mid);//递归划分左半区
    mesort(arr,tempArr,mid+1,right);  //递归划分右半区
    merge(arr,tempArr,left,mid,right);//合并已经排序的部分
  }
}


void merge(int arr[], int tempArr[],int left,int mid,int right)
{
  int l-pos=left;//标记左半边第一个未排序的元素
  int r-pos=mid+1;//标记右半边第一个未排序的元素
  int pos=left;//临时数组下标
  //合并
  while(l-pos<left && r-pos<=right )
  {
    if(arr[l-pos]<arr[r-pos])//左半区的元素小，传入临时数组
      tempArr[pos++]=arr[l-pos++];
    else
      tempArr[poss++]=arr[r-pos++];
  }
   //合并左边剩余区域
  while(l-pos<=mid)
  {
    tempArr[pos++]=arr[l-pos++];
  }
  //合并右边剩余区域
   while(r-pos<=right)
  {
    tempArr[pos++]=arr[r-pos++];
  }

  //把临时数组中合并后的元素复制到原数组。
  while(left<=right)
  {
    arr[left]=tempArr[left];
      left++;
```

```
 }
}
```

用c#实现-个排序方法，能对任意单一 类型数组排序比如Intgerl],String[,Long[]或者其他任意类型的数组

```
public static void BubbleSort<T>(T[] arr) where T : IComparable<T>
{
   int n = arr.Length;
   for (int i = 0; i < n - 1; i++)
   {
      for (int j = 0; j < n - i - 1; j++)
      {
         if (arr[j].CompareTo(arr[j + 1]) > 0)
         {
            T temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
         }
      }
   }
}
```

我们使用泛型方法，在方法定义时通过where限制T必须实现IComparable<T>接口，即T类型必须支持比较操作。接下来，在冒泡排序算法中，我们比较数组中相邻的元素，如果它们的顺序不正确，就交换它们的位置。注意，我们直接使用T类型的CompareTo方法进行比较操作。这是因为我们通过where限制了T类型必须实现IComparable<T>接口，所有T类型的实例都可以使用CompareTo方法进行比较。因此，我们可以对任意支持比较操作的类型进行排序。

@五分钟学算法之堆排序

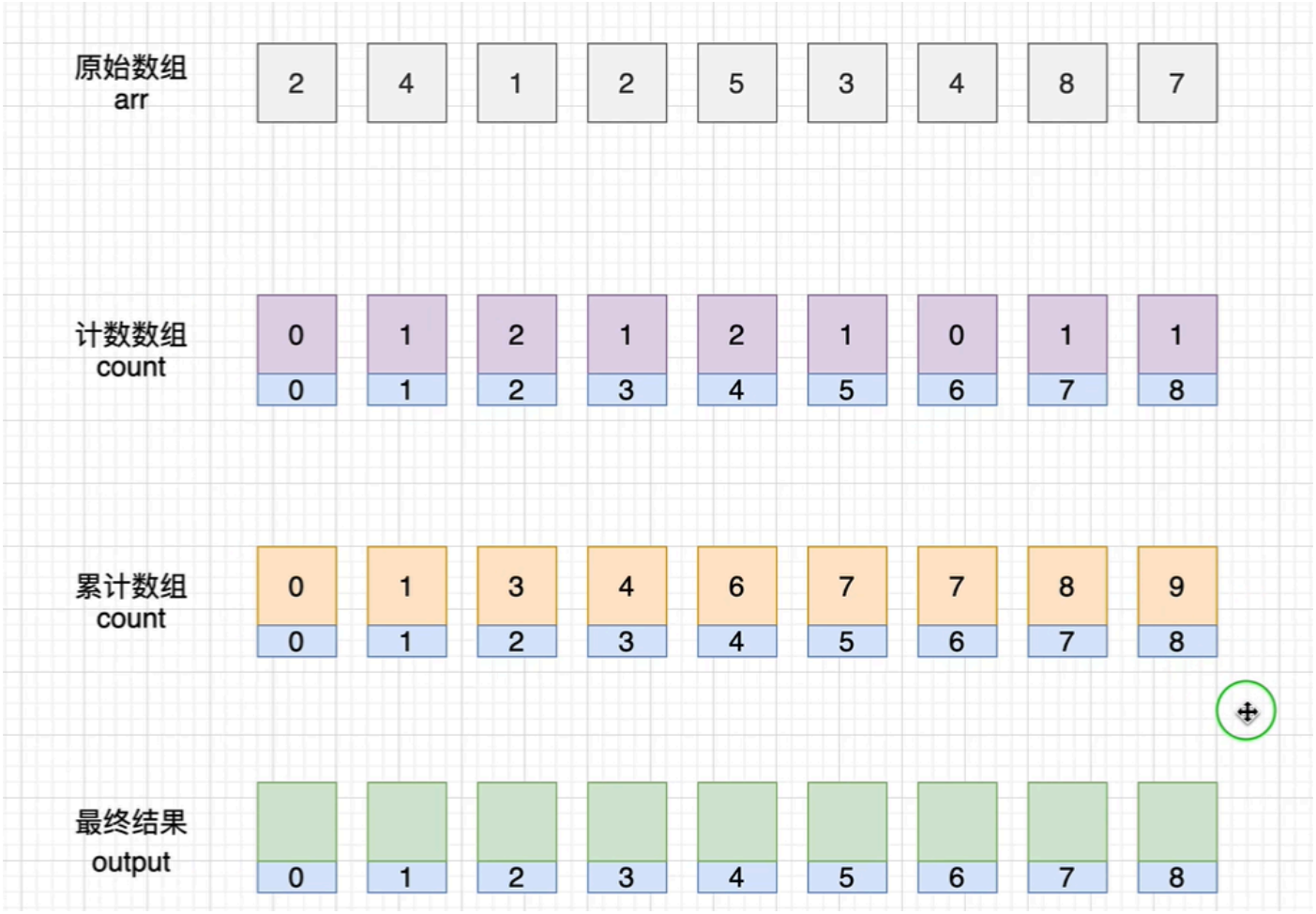
堆排序

```
1，建堆
2，维护堆
3，排序

//堆的维护，建立大顶堆
void heapify(int arr[],int n,int i)
{
  int larget=i//假设最大的数字下标为i;
  int lson=i*2+1;//左孩子下标
  int rson=i*2+2;//右孩子下标
  if(lson<n && arr[larget]<arr[lson])
    larget=lson;//把左子树下标传给larget;
  if(rson<n  && arr[larget] < arr[rson])
    larget=rson;//把右孩子下标传给larget;
  if（larget ！ = i)
  {
    swap(&arr[larget],&arr[i]);
    heapify(arr,n,larget);//递归，继续判断
  }
}

void heap-sort(int arr[],int n)
{
  int i;
  //建堆
  for(i=n/2-1; i>0; i--)//i的父节点的下标为(n-1)/2,这里n为最后一个数的下标，就是arr.length()-1数组长度-1
  {
    heapify(arr,n,i);//建堆之后进行堆的维护
  }
  //排序
  for(i=n-1; i>0;i--)
  {
    swap(&arr[i],arr[0]);
    heapify(arr,i,0);
  }
}
```

计数排序

```
void int count-sort(int arr[],int len)
{
  if(len<1)  return;
  //找到数组中的最大值
  int max=arr[0];
  for(int i=0; i<len; i++ )
```

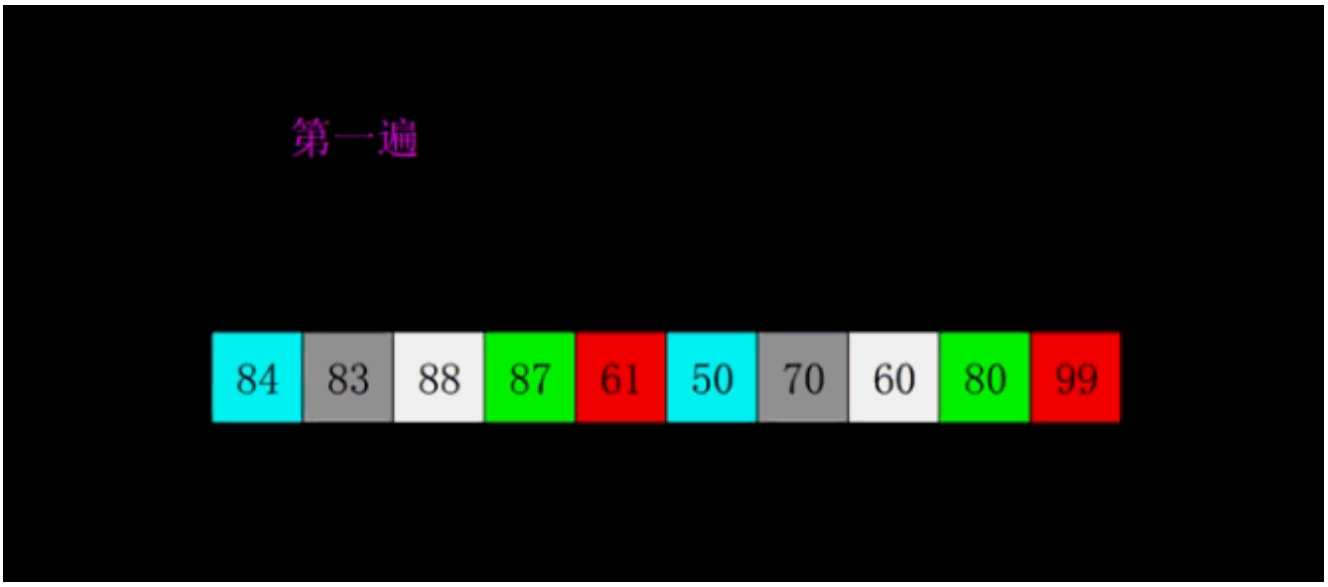
```
{
  if(arr[i]>max)
    max=arr[i];
}

//分配一个数组地址，用来计数,数组的长度就是最大值
int *count = (int*) malloc(sizeof(int*)*(max-1));
memset( count, 0, sizeof(int*)*(max-1));//让数组内容都为0;


//计数
for(int i=0;i<len; i++)
{
  count[arr[i]]++;//相同的数字累加
}

//统计计数的累计值
for(int i =0; i< max-1; i++)
{
  count[i] += count[i-1];
}
//分配一个数组地址，用来保存结果
int *output=(int*)malloc(sizeof(int)*len);

//将元素放到正确的位置上
for(i=0;i<len;i++)
{
  output[ count[arr[i]-1] ]=arr[i];
  //当i=0的时候arr[0]是2,count中比2小的数字有三个，而output的下标是从0开始的，所以count[arr[i]-1];
  count[arr[i]]--;
}
//将结果返回给原数组
for(i=0; i<len; i++)
{
  arr[i]=output[i];
}
}

```



桶排序

```
using System;

class Program {
    static void BucketSort(int[] arr, int bucketSize = 5)
    {
        if (arr.Length == 0) return;

        // 找到最大值和最小值
        int minValue = arr[0], maxValue = arr[0];
        for (int i = 1; i < arr.Length; i++) {
            if (arr[i] < minValue) {
                minValue = arr[i];
            } else if (arr[i] > maxValue) {
                maxValue = arr[i];
            }
        }

        // 桶的数量
        int bucketCount = (maxValue - minValue) / bucketSize + 1;
        int[][] buckets = new int[bucketCount][];

        // 将数据放入桶中
        for (int i = 0; i < arr.Length; i++) {
```

```
24          int bucketIndex = (arr[i] - minValue) / bucketSize;
25          if (buckets[bucketIndex] == null) {
26              buckets[bucketIndex] = new int[0];
27          }
28          buckets[bucketIndex] = InsertionSort(buckets[bucketIndex], arr[i]);
29      }
30
31      // 合并桶中的数据
32      int index = 0;
33      for (int i = 0; i < buckets.Length; i++) {
34          if (buckets[i] == null) continue;
35          for (int j = 0; j < buckets[i].Length; j++) {
36              arr[index++] = buckets[i][j];
37          }
38      }
39  }
40
41  static int[] InsertionSort(int[] arr, int key) {
42      int i;
43      for (i = arr.Length - 1; i >= 0 && arr[i] > key; i--) {
44          arr[i + 1] = arr[i];
45      }
46      arr[i + 1] = key;
47
48      return arr;
49  }
50
51  static void Main(string[] args) {
52      int[] arr = new int[] { 4, 3, 2, 1, 0 };
53      BucketSort(arr);
54      Console.WriteLine(string.Join(",", arr));
55  }
56 }
```



希尔排序

```
void shellSort(int[] arr)
{
    int n = arr.Length;
    // 初始步长设为数组长度的一半
      for (int gap = n / 2; gap > 0; gap /= 2)
      {
         for (int i = gap; i < n; i++)
         {
            for (int j = i; j >= gap && arr[j] < arr[j - gap]; j -= gap)
            {
               var temp = arr[j];
               arr[j] = arr[j - gap];
               arr[j - gap] = temp;
            }

         }
      }

  }
```