

(3/500)在线性时间内对二进制数组排序

给定一个二进制数组（数组元素是0和1），请对数组进行排序，要求时间复杂度为 $O(n)$ ，空间复杂度为 $O(1)$ 。输出内容应该是所有的0在前，之后是所有的1。

例如，

输入:{ 1, 0, 1, 0, 1, 0, 0, 1 }

输出:{ 0, 0, 0, 0, 1, 1, 1, 1 }

1. 计算0的个数

一种简单的方法是计算数组中0的个数（假设为 k ），然后把数组的前 k 个元素置为0，其他的置为1。同样道理，也可以计算1的个数（假设为 k ），然后把前面的 k 个元素置为1，其他的置为0。二者选一即可。实现方式比较简单，代码就省略了。

2. 交换元素

和计算0的个数相比，我们可以换种方法。如果当前元素是0，我们可以把0放到下一个可用的位置。在所有的元素都处理了之后，把其它所有的元素都置为1。

C++语言实现：

```
#include <bits/stdc++.h>
using namespace std;

// Function to sort binary array in linear time
int Sort(int A[], int n)
{
    // k stores index of next available position
    int k = 0;

    // do for each element
    for (int i = 0; i < n; i++)
    {
        // if current element is zero, put 0 at next free
        // position in the array
        if (A[i] == 0)
            A[k++] = 0;
    }
}
```

```

// fill all remaining indexes by 1
for (int i = k; i < n; i++)
    A[k++] = 1;
}

// main function
int main()
{
    int A[] = { 0, 0, 1, 0,1, 1, 0, 1, 0, 0 };
    int n = sizeof(A)/sizeof(A[0]);

    Sort(A, n);

    // print the rearranged array
    for (int i = 0 ; i < n; i++)
        cout << A[i] << " ";

    return 0;
}

```

java语言实现：

```

class SortBinaryArray
{
    // Function to sort binary array in linear time
    public static void Sort(int A[], int n)
    {
        // k stores index of next available position
        int k = 0;

        // do for each element
        for (int i = 0; i < n; i++)
        {
            // if current element is zero, put 0 at next free
            // position in the array
            if (A[i] == 0)
                A[k++] = 0;
        }

        // fill all remaining indexes by 1
    }
}

```

```

        for (int i = k; i < n; i++)
            A[k++] = 1;
    }

    // main function
    public static void main (String[] args)
    {
        int A[] = { 0, 0, 1, 0, 1, 1, 0, 1, 0, 0 };
        int n = A.length;

        Sort(A, n);

        // print the rearranged array
        for (int i = 0 ; i < n; i++)
            System.out.print(A[i] + " ");
    }
}

```

输出: 0 0 0 0 0 0 1 1 1 1

以上两种方案的时间复杂度都是 $O(n)$ ，空间复杂度都是 $o(1)$ 。

3. 快速排序

除了以上两种方法，我们还可以借用快速排序中分而治之的思想。这种方法的关键是使用1作为主元素（pivot element），然后创建一个分区过程，最后的结果就是排好序的。

C++语言实现：

```

#include <bits/stdc++.h>
using namespace std;

// Function to sort binary array in linear time
int Partition(int A[], int n)
{
    int pivot = 1;
    int j = 0;

```

```

// each time we encounter a 0, j is incremented and
// 0 is placed before the pivot
for (int i = 0; i < n; i++)
{
    if (A[i] < pivot)
    {
        swap(A[i], A[j]);
        j++;
    }
}

```

```

// main function
int main()
{
    int A[] = { 1, 0, 0, 0, 1, 0, 1, 1 };
    int n = sizeof(A)/sizeof(A[0]);

    Partition(A, n);

    // print the rearranged array
    for (int i = 0 ; i < n; i++)
        cout << A[i] << " ";

    return 0;
}

```

java语言实现：

```

class SortBinaryArray
{
    // Function to sort binary array in linear time
    public static void Sort(int A[], int n)
    {
        int pivot = 1;
        int j = 0;

        // each time we encounter a 0, j is incremented and
        // 0 is placed before the pivot
        for (int i = 0; i < n; i++)

```

```

    {
        if (A[i] < pivot)
        {
            // swap (A[i], A[j])

            int temp = A[i];
            A[i] = A[j];
            A[j] = temp;

            j++;
        }
    }
}

// main function
public static void main (String[] args)
{
    int A[] = { 0, 0, 1, 0, 1, 1, 0, 1, 0, 0 };
    int n = A.length;

    Sort(A, n);

    // print the rearranged array
    for (int i = 0 ; i < n; i++)
        System.out.print(A[i] + " ");
}
}

```

输出: 0 0 0 0 0 0 1 1 1 1

尽管以上各种方法时间复杂度是 $O(n)$ ，空间复杂度是 $O(1)$ ，明显第三种方法的时间更少。

4. 还能想到另外一种解决方案，新建两个索引分别指向首尾（head 和 end），向中间聚拢，当head指向的数字不为0，并且end指向的数字不为1时，交换它们的位置，直到head大于或等于end。

我还能想到另外一种解决方案，新建两个索引分别指向首尾（head 和 end），向中间聚拢，当head指向的数字不为0，并且end指向的数字不为1时，交换它们的位置，直到head大于或等于end。

闲话少说，上代码。偷一下懒，我就只写Java代码了。

```

public class SortBinaryArray2 {

    // Function to sort binary array in linear time
    public static void Sort(int A[], int n)
    {
        □int head = 0;
        □int end = A.length - 1;

        □while( head < end )
        □{
            □□while( A[head] == 0 && head < A.length -1 )
            □□{
                □□□head ++;
            □□}
            □□while( A[end] == 1 && end > 0 )
            □□{
                □□□end --;
            □□}
            □□if( head >= end )
            □□{
                □□□break;
            □□}
            □□int temp = A[head];
            □□A[head] = A[end];
            □□A[end] = temp;
            □}
        }

    □public static void main(String[] args) {
        int A[] = { 0, 0, 1, 0, 1, 1, 0, 1, 0, 0 };
        int n = A.length;

        Sort(A, n);

        // print the rearranged array
        for (int i = 0 ; i < n; i++)
        {
            □ System.out.print(A[i] + " ");
        }
    □}
}

```

```
}
```

这种方法是不是更省时？ ^_^

练习：

1. 在原方案的基础上修改方案，把1放在前面。

2.

给定一个数组，在线性时间内（时间复杂度为 $O(n)$ ）排序，使所有的偶数排在奇数之前。