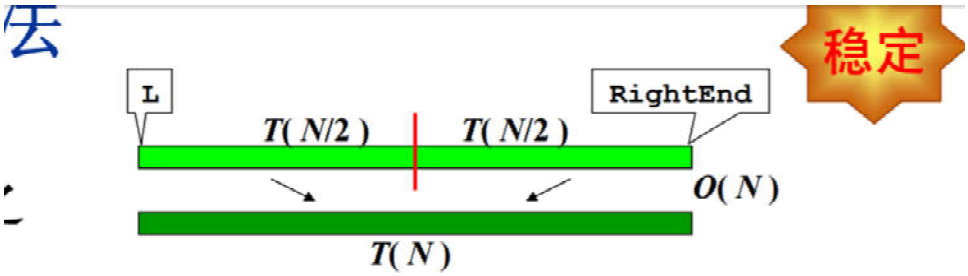
归并算法

2019年6月30日　　15:16

稳定的算法

分而治之



递归

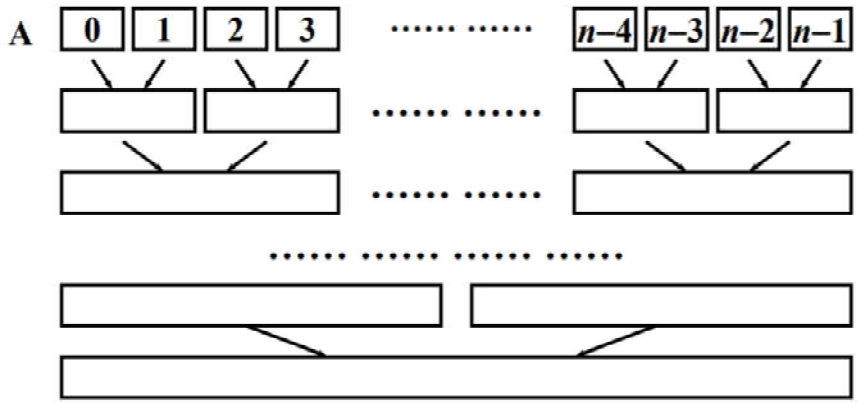
代码：

```
#include<iostream>
using namespace std;
//这是递归的归并排序
/* L = 左边起始位置, R = 右边起始位置, RightEnd = 右边终点位置*/
void Merge(int* a,int *tempa, int L, int R, int RightEnd)
{
	/* 将有序的A[L]~A[R-1]和A[R]~A[RightEnd]归并成一个有序序列 */
	int LeftEnd, NumElements, Tmp;
	int i;

	LeftEnd = R - 1; /* 左边终点位置 */
	Tmp = L;	/* 有序序列的起始位置 */
	NumElements = RightEnd - L + 1;

	while (L <= LeftEnd && R <= RightEnd) {
		if (a[L] <= a[R])
			tempa[Tmp++] = a[L++]; /* 将左边元素复制到tempa */
		else
			tempa[Tmp++] = a[R++]; /* 将右边元素复制到tempa */
	}

	while (L <= LeftEnd)
		tempa[Tmp++] = a[L++]; /* 直接复制左边剩下的 */
	while (R <= RightEnd)
		tempa[Tmp++] = a[R++]; /* 直接复制右边剩下的 */

	for (i = 0; i < NumElements; i++, RightEnd--)
		a[RightEnd] = tempa[RightEnd]; /* 将有序的tempa[]复制回A[] */
}

void Msort(int *a,int *tempa,int L,int RightEnd)
{
	//核心
	int center;
	if (L < RightEnd)
	{
```

```
        center = (L + RightEnd) / 2;
        Msort(a, tempa, L, center);          /* 递归解决左边 */
        Msort(a, tempa, center + 1, RightEnd);   /* 递归解决右边 */
        Merge(a, tempa, L, center + 1, RightEnd);  /* 合并两段有序序列 */
    }

}

void MergeSort(int *a,int n)
{
    //申请和a长度一样的临时数组
    int *tempa=(int *)malloc(n*sizeof(int));
    if (tempa != NULL)
    {
        Msort(a, tempa, 0, n - 1);
        free(tempa);
    }
    else cout << "空间不足";
}



int main()
{
    int n;
    cin >> n;
    int* a = new int[n];
    for (int i = 0; i < n; i++)
    {
        cin >> a[i];
    }
    MergeSort(a, n);

    for (int i = 0; i < n; i++)
    {
        cout << a[i]<<" ";
    }
    delete[] a;

}
```

非递归（循环）

非递归算法



额外空间复杂度是???　　$O(N)$

代码：

```
#include<iostream>
#define ElementType int
using namespace std;

/* 归并排序 - 循环实现 */

void Merge(ElementType A[], ElementType TmpA[], int L, int R, int RightEnd)
{ /* 将有序的A[L]~A[R-1]和A[R]~A[RightEnd]归并成一个有序序列 */
	int LeftEnd, NumElements, Tmp;
	int i;

	LeftEnd = R - 1; /* 左边终点位置 */
	Tmp = L;	/* 有序序列的起始位置 */
	NumElements = RightEnd - L + 1;

	while (L <= LeftEnd && R <= RightEnd) {
		if (A[L] <= A[R])
			TmpA[Tmp++] = A[L++]; /* 将左边元素复制到TmpA */
		else
			TmpA[Tmp++] = A[R++]; /* 将右边元素复制到TmpA */
	}

	while (L <= LeftEnd)
		TmpA[Tmp++] = A[L++]; /* 直接复制左边剩下的 */
	while (R <= RightEnd)
		TmpA[Tmp++] = A[R++]; /* 直接复制右边剩下的 */

	for (i = 0; i < NumElements; i++, RightEnd--)
		A[RightEnd] = TmpA[RightEnd]; /* 将有序的TmpA[]复制回A[] */
}
/* length = 当前有序子列的长度*/
void Merge_pass(ElementType A[], ElementType TmpA[], int N, int length)
{ /* 两两归并相邻有序子列 */
	int i, j;

	for (i = 0; i <= N - 2 * length; i += 2 * length)
```

```
            Merge(A, TmpA, i, i + length, i + 2 * length - 1);
        if (i + length < N) /* 归并最后2个子列*/
            Merge(A, TmpA, i, i + length, N - 1);
        else /* 最后只剩1个子列*/
            for (j = i; j < N; j++) TmpA[j] = A[j];
}

void Merge_Sort(ElementType A[], int N)
{
        int length;
        ElementType* TmpA;

        length = 1; /* 初始化子序列长度*/
        TmpA =(int*) malloc(N * sizeof(ElementType));
        if (TmpA != NULL) {
            while (length < N) {
                Merge_pass(A, TmpA, N, length);
                length *= 2;
                Merge_pass(TmpA, A, N, length);
                length *= 2;
            }
            free(TmpA);
        }
        else printf("空间不足");
}

int main()
{
        int n;
        cin >> n;
        int* a = new int[n];
        for (int i = 0; i < n; i++)
        {
            cin >> a[i];
        }
        Merge_Sort(a, n);

        for (int i = 0; i < n; i++)
        {
            cout << a[i]<<" ";
        }
        delete[] a;

}
```