**南京理工大学计算机科学与工程学院**

**算法设计与分析 报告**

**班 级 9181069502**

**学生姓名 黄海浪**

**学 号 9181040G0818**

**起止时间 2020.03.30-2020.03.31**

**教 师 孙廷凯**

目录

[一.问题描述与实验要求 3](#_Toc36555241)

[二.符号说明与语法约定 3](#_Toc36555242)

[三.算法自然语言描述 3](#_Toc36555243)

[四. 查找算法伪代码描述 4](#_Toc36555244)

[五. 查找算法的代码及结果 5](#_Toc36555245)

[代码： 5](#_Toc36555246)

[结果： 6](#_Toc36555247)

## 问题描述与实验要求

用分治策略设计一个算法，实现n个元素的归并排序，其中n=2^k。

## 符号说明与语法约定

排序的数据来源于系统生成的随机数。

若无特殊说明，“时间复杂度”均指 排序的时间复杂度。

## 三.算法自然语言描述

分治三步法：

划分问题：把序列分成元素尽量相等的两半。

递归求解：把两半元素分别排序。

合并问题：把排序好的序列合并成一个。

复杂度分析：

一般情况下 且 n = 2^k：

T(n) = 2T(n/2)+ n – 1

= 2^2 T(n/2^2) + 2n – 1 – 2

= ...

= n\*logn -1-2-4-...-2^k

= nlogn -n + 1

故排序算法的时间复杂度为θ(nlogn)

图表表示：

双击下图



下图为排序的两个数组归并为一个数组过程



## 四. 查找算法伪代码描述

Algorithm: MERGESORT(A, l, m, h)

输入：数组A[l...h-1]

输出：一个升序排序后的数组A[l...h-1]

1.  if h-l > 1 then

2.  i←l,p←l,q←m

3.  MERGESORT(A,l,(m+l)/2,m)

4.  MERGESORT(A,m,(h+m)/2,h)

5.  while p<m or q<h

6.      if ( A[p] ≤A[q] and p<m )  or q>=h then

7.          B[i]←A[p]

8.          i ← i+1, p ← p+1

9.      else

10.          B[i]←A[q]

11.          i ← i+1, q ← q+1

12.      end if

13. end while

14. A[l...h] ← B[l...h]

15. end if

## 五. 查找算法的代码及结果

### 代码：

#include<iostream>

#include<stdlib.h>

#include<time.h>

#include<math.h>

using namespace std;

typedef long long LL;

const int MAXRAND = 0x0fffffff;     //随机的最大的数

const int \_2K = 20;                 // n = 2^\_2K

void merge\_sort(int \*A,int l,int h,int \*T) {

    if(h-l>1) {                                 // 如果超过 1 个数

        int m = l + (h-l)/2;                    // mid 拆分

        int p = l, i = l, q = m;                // p -> arr[low]   index     q -> arr[mid]

        merge\_sort(A,l,m,T);                    //递归

        merge\_sort(A,m,h,T);

        while( p<m || q<h ) {                   //左边index < mid   ||   右边index < high

            if( (A[p]<A[q] && p<m ) || q>=h )   //左边小 & index < mid 或者 右边 全部处理了

                T[i++] = A[p++];

            else T[i++] = A[q++];

        }

        for(int i = l;i<h; ++i) {

            A[i] = T[i];

        }

    }

}

int main() {

    //  初始化随机数

    srand(time(0));

    //  n = 2^k

    int k = \_2K;

    LL n = (LL)(pow(2,k));

    cout<<"k: "<<k<<endl<<"n: "<<n<<endl<<"maxRand:"<<MAXRAND-1<<endl;

    // 初始化数组

    int \*arr = new int[n] , \*tmp = new int [n];

    for(LL i = 0; i<n; ++i) {

        arr[i] = rand()%MAXRAND;

    }

    //输出

//  cout<<endl<<"未排序:"<<endl;

//  for(LL i = 0; i<n; ++i) {

//      cout<<arr[i]<<" ";

//      if( !(i%20) && i) {

//          cout<<endl;

//      }

//  }

//  cout<<endl<<"--------------------------------"

//      <<endl<<endl<<"排序后:"<<endl;

    // merge\_sort

    LL start = clock();

    merge\_sort(arr,0,n,tmp);

    LL end = clock();

    cout<<endl<<"排序所用时间: "<< (end - start)<<"mS"<<endl<<endl;

    //输出

//  for(LL i = 0; i<n; ++i) {

//      cout<<arr[i]<<" ";

//      if( !(i%20) && i) {

//          cout<<endl;

//      }

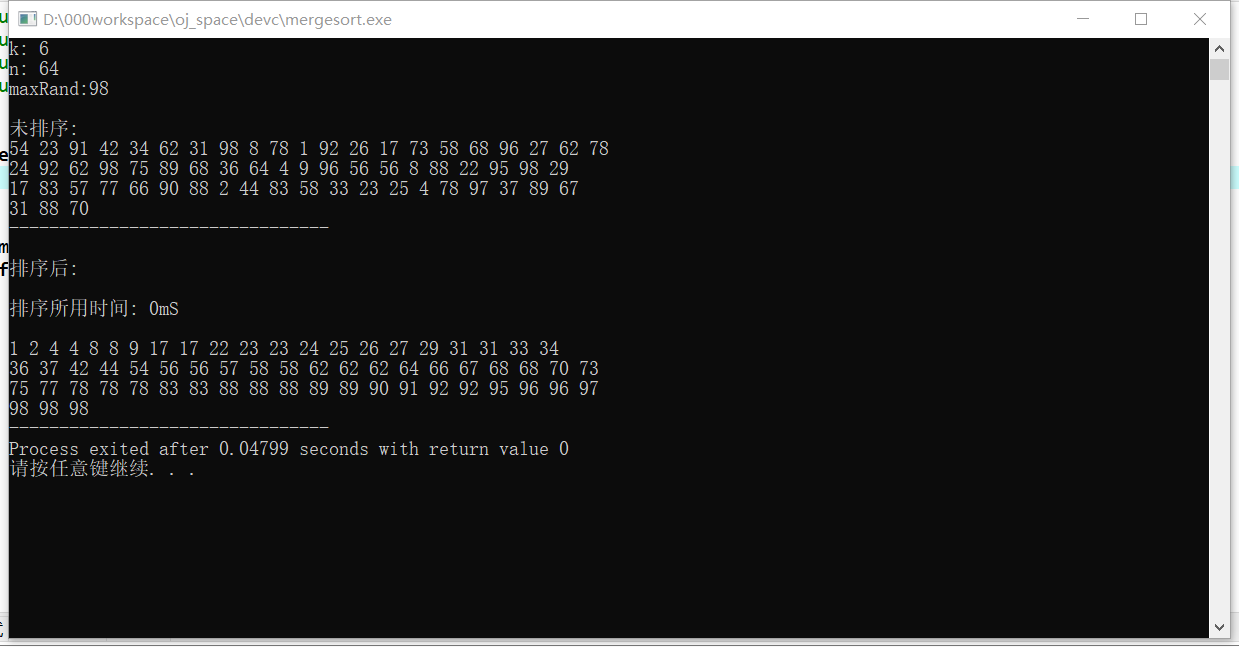
//  }

    return 0;

}

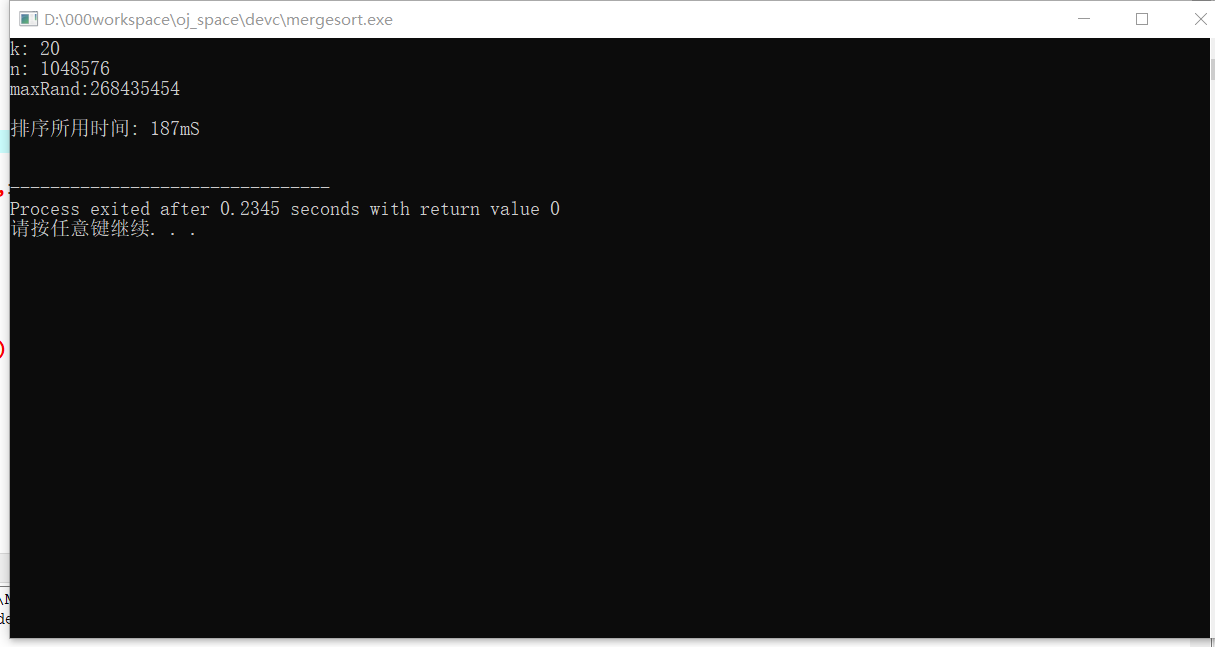
### 结果：

对于小规模数据，排序后效果打印：

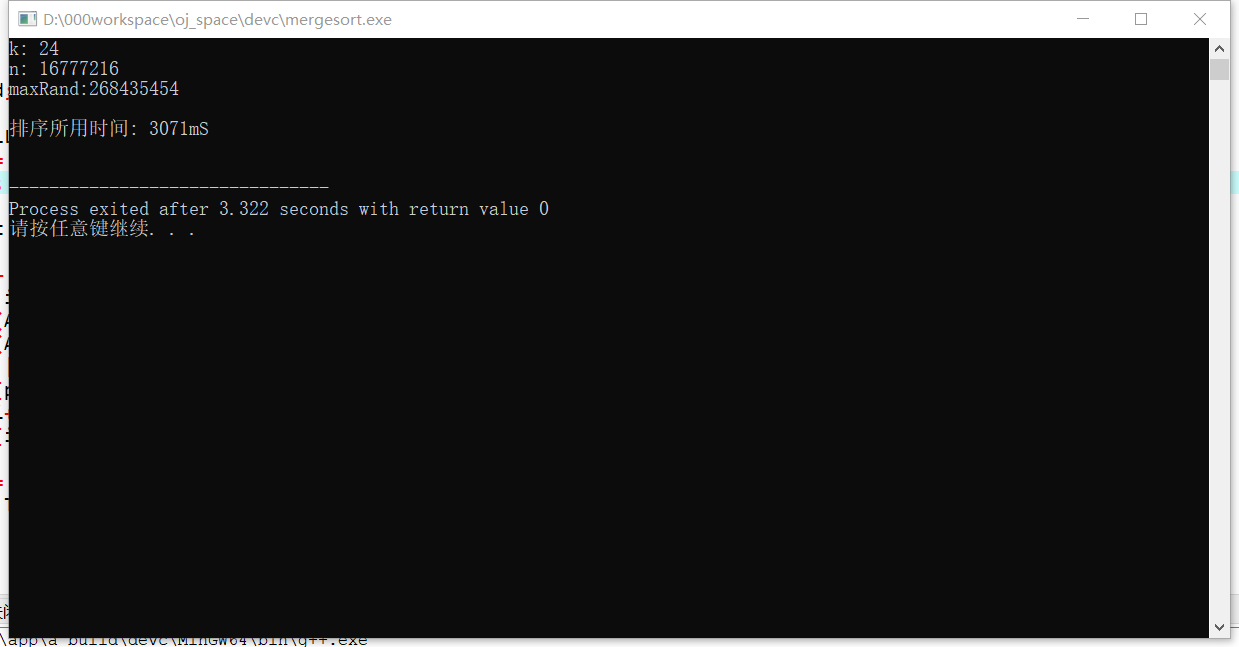


对于大规模数据，所用时间如下：

百万级别所用时间0.2-3s（大概范围）



千万级别所用时间3-25s（大概范围）



亿级所用时间25s-250s（大概范围）

