算法排序算法实验报告

2019211283 304班 陈童

10/25/2021

1.实验题目

对 n 个整数使用归并排序/快速排序进行升列排序.使用mergesort.in / quicksort.in 作为输入文件，将排序后的序列输出到 mergesort.out / quicksort.out 中。 在输入文件中，先输入正整数N表示待输入的序列长度，再输入整个序列。

2. 实验过程

2.1 归并排序（Mergesort）(完成了迭代和递归两种实现)

2.1.1 算法思路

使用递归，不断将未排序的序列拆分成两个小的未排序的序列，分别调用归并排序。待子序列被整理有序后，调用merge（）函数将两个有序序列合并为一个有序序列，并返回这个有序序列。

伪代码表示：

```
Mergesort(A,i,j){
if(i<j)
{
mid = (i+j)/2
Mergesort(A,i,mid)
Mergesort(A,mid+1,j)
Merge(A,B,i,mid,mid+1,j)
Copy(B,A)
return A
}
}
```

2.1.2 复杂度分析

时间复杂度:T(n)=2T(n/2)+cn , n>1
T(1)=0

T(1)=0

可算出时间复杂度T(n)=O(nlogn)

空间复杂度: S(n)=2S(n/2)+cn,n>1

S(1)=cn

可推出空间复杂度S(n)为O(nlogn)

2.2 快速排序

2.2.1 算法思路

快速排序是一种in-place的排序算法。与归并排序类似，快速排序不断将序列拆分成小的未排序序列，通过将每次指定的元素移动到相应位置，保证该元素左边的元素都比他小，右边的元素都比他大。
伪代码表示

```
quicksort(A,i,j){
    if(i<j){
        q=partition(A,i,j)
        quicksort(A,i,q-1)
        quicksort(A,q+1,j)
        return A
    }
}
```

2.2 复杂度分析

与归并排序相似，快速排序的时间复杂度为O(nlogn)
但因为没有额外空间申请的开销，空间复杂度为O(n)

3. 程序运行结果

3.1 归并排序

在命令行中输入以下指令以运行程序

```
g++ merge_sort.cpp -o mergesort && ./mergesort
```

mergesort.in

```
5
9 11 5 22 12
```

mergesort.out

```
5 9 11 12 22
```

C Compiler ...

```
merge_sort.cpp    quick_sort.cpp
❯ g++ merge_sort.cpp -o mergesort && ./mergesort
Time used 0.032 miliseconds
```

当随机生成数据到N=100时

mergesort.in

```
100
57 -1 -177 -92 180 22 -206 128 173 -41
190 -85 242 -208 237 -247 77 -21 90 -138
53 -81 -41 -93 -190 183 -151 28 66 85
-153 76 -238 17 60 -117 229 -101 -171 71
217 -78 143 86 235 -5 -22 -159 -56 107
-249 -97 -42 194 -82 240 -126 -54 -220 153
-28 -84 -201 -226 51 103 227 158 -22 183
48 231 -115 -237 115 64 -187 -214 175 -81
-135 -156 -121 -249 -233 -55 -145 154 201 48
-62 -127 -245 132 2 -184 -34 87 188 -106
```

gesort.out

```
-249 -249 -247 -245 -238 -237 -233 -226 -220 -214
-208 -206 -201 -190 -187 -184 -177 -171 -159 -156
-153 -151 -145 -138 -135 -127 -126 -121 -117 -115
-106 -101 -97 -93 -92 -85 -84 -82 -81 -81
-78 -62 -56 -55 -54 -42 -41 -41 -34 -28
-22 -22 -21 -5 -1 2 17 22 28 48
48 51 53 57 60 64 66 71 76 77
85 86 87 90 103 107 115 128 132 143
153 154 158 173 175 180 183 183 188 190
194 201 217 227 229 231 235 237 240 242
```

```
❯ g++ gen_test_case.cpp -o gen && ./gen && g++ merge_sort.c
pp -o mergesort && ./mergesort
Time used 0.22 miliseconds
```

当取N=1000时

```
❯ g++ gen_test_case.cpp -o gen && ./gen && g++ merge_sort.c
pp -o mergesort && ./mergesort
Time used 1.917 miliseconds
```

mergesort.in

```
1000
57 -1 -177 -92 180 22 -206 128 173 -41
190 -85 242 -208 237 -247 77 -21 90 -138
53 -81 -41 -93 -190 183 -151 28 66 85
-153 76 -238 17 60 -117 229 -101 -171 71
217 -78 143 86 235 -5 -22 -159 -56 107
-249 -97 -42 194 -82 240 -126 -54 -220 153
-28 -84 -201 -226 51 103 227 158 -22 183
```

```
48 231 -115 -237 115 64 -187 -214 175 -81
-135 -156 -121 -249 -233 -55 -145 154 201 48
-62 -127 -245 132 2 -184 -34 87 188 -106
-249 147 121 78 -113 108 -73 147 44 154
-141 -19 -5 -75 -115 48 -108 149 218 162
10 -193 -156 -242 145 218 -137 -220 -244 212
192 115 -168 102 17 71 -55 -38 -79 151
-160 81 -12 -193 -134 40 -110 -71 85 -244
222 -152 -155 69 204 -27 39 10 155 -124
-127 -244 63 20 -12 -156 -30 94 116 -216
-24 144 113 -12 94 -160 -200 -91 -127 197
135 -33 22 -211 -3 135 246 135 -127 170
-106 218 -15 165 -125 -216 -208 -239 -71 -98
-6 45 68 146 -58 65 241 -217 -81 147
-197 -203 75 -40 -88 -39 58 127 11 -125
85 118 245 26 17 189 124 81 -194 51
122 -190 -156 234 5 39 157 -235 -57 19
-70 205 105 -244 213 -248 -74 -142 -1 81
94 -112 58 247 -99 99 -47 -19 -219 -236
169 25 -241 -70 179 -27 -196 10 -13 -205
67 -225 -50 6 -186 -153 -102 -46 -200 -100
226 162 -196 47 -246 131 -2 -185 128 -51
-41 -121 -197 233 197 -143 23 72 55 -74
-197 -26 -120 116 150 194 120 35 -234 148
-95 -117 -193 -174 -72 16 107 -39 128 -136
69 -13 -117 -159 -30 12 -117 -53 -219 -162
-161 123 127 54 108 -50 4 210 165 197
```

ERMINAL PROBLEMS OUTPUT

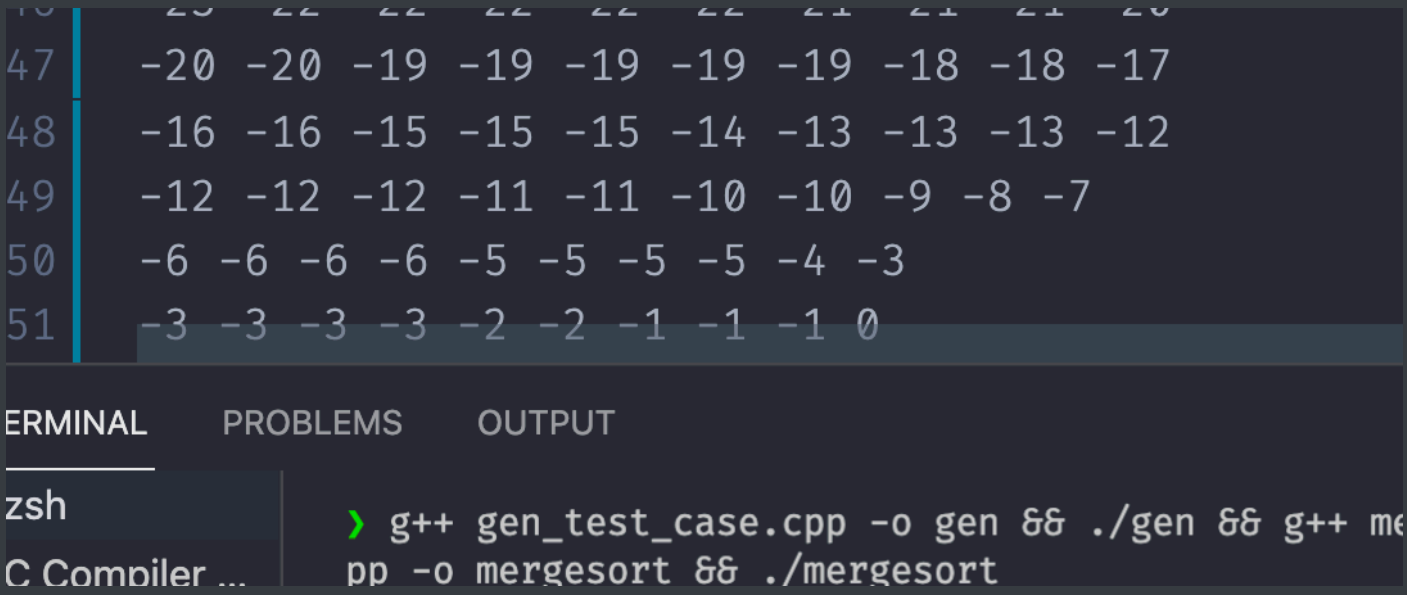
zsh

C Compiler ...

```
❯ g++ gen_test_case.cpp -o gen && ./gen && g++ me
pp -o mergesort && ./mergesort
```

mergesort.out

```
-180 -180 -179 -177 -176 -175 -174 -174 -171 -17
-171 -170 -170 -170 -169 -169 -168 -167 -166 -16
-164 -163 -163 -162 -161 -161 -161 -161 -160 -16
-160 -159 -159 -159 -158 -158 -157 -157 -156 -15
-156 -156 -156 -155 -154 -154 -153 -153 -153 -15
-152 -151 -150 -149 -149 -148 -147 -147 -147 -14
-145 -144 -143 -143 -143 -142 -142 -142 -141 -14
-140 -139 -138 -138 -138 -138 -137 -137 -137 -13
-136 -135 -135 -134 -134 -134 -133 -130 -129 -12
-129 -128 -127 -127 -127 -127 -127 -127 -126 -12
-125 -125 -125 -125 -124 -123 -123 -123 -122 -12
-121 -121 -120 -120 -119 -118 -117 -117 -117 -11
-115 -115 -113 -113 -113 -112 -112 -111 -111 -11
-110 -109 -109 -109 -108 -107 -107 -106 -106 -10
-102 -101 -101 -100 -100 -100 -100 -100 -99 -99
-99 -98 -98 -98 -98 -97 -96 -95 -94 -93
-93 -92 -92 -92 -91 -89 -89 -89 -88 -88
-87 -87 -86 -85 -85 -84 -84 -84 -84 -83
-83 -82 -81 -81 -81 -81 -81 -81 -80 -79
-79 -79 -79 -79 -78 -77 -75 -75 -75 -74
-74 -74 -74 -73 -73 -72 -71 -71 -71 -71
-70 -70 -67 -66 -66 -62 -61 -61 -60 -58
-58 -57 -57 -56 -55 -55 -55 -54 -54 -54
-53 -52 -52 -52 -52 -51 -51 -50 -50 -50
-47 -47 -47 -46 -45 -43 -42 -42 -41 -41
-41 -40 -40 -40 -40 -40 -39 -39 -39 -39
-39 -38 -36 -36 -35 -34 -34 -33 -33 -33
-32 -32 -31 -31 -31 -30 -30 -30 -30 -30
-30 -29 -28 -27 -27 -27 -26 -24 -24 -23
-23 -22 -22 -22 -22 -22 -21 -21 -21 -20
```

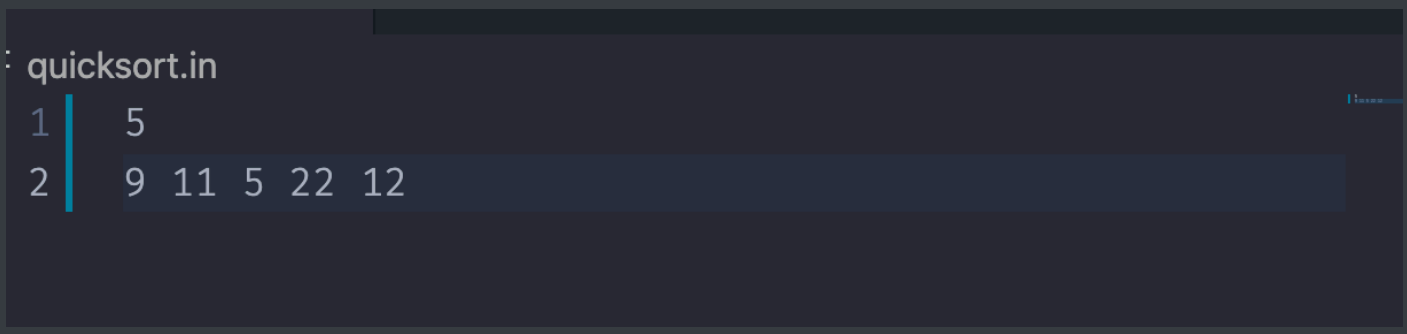


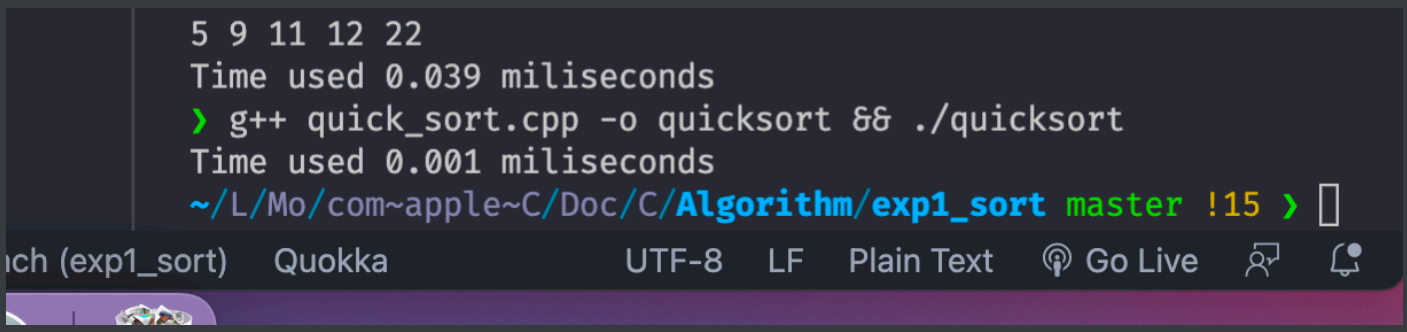
可见，当N=5,100,1000时， 耗时分别为0.032ms,0.22ms和1.9ms，大致符合O(nlogn)的曲线

3.2 快速排序

在命令行中输入以下指令以运行程序

```
g++ quick_sort.cpp -o quicksort && ./quicksort
```





quicksort.out

```
5 9 11 12 22
```

取N=100

```
Time used 0.002 miliseconds
❯ g++ gen_test_case.cpp -o gen && ./gen && ./quicksort
Time used 0.013 miliseconds
```

quicksort.in

```
100
57 -1 -177 -92 180 22 -206 128 173 -41
190 -85 242 -208 237 -247 77 -21 90 -138
53 -81 -41 -93 -190 183 -151 28 66 85
-153 76 -238 17 60 -117 229 -101 -171 71
217 -78 143 86 235 -5 -22 -159 -56 107
-249 -97 -42 194 -82 240 -126 -54 -220 153
-28 -84 -201 -226 51 103 227 158 -22 183
48 231 -115 -237 115 64 -187 -214 175 -81
-135 -156 -121 -249 -233 -55 -145 154 201 48
-62 -127 -245 132 2 -184 -34 87 188 -106

```

quicksort.out

```
-249 -249 -247 -245 -238 -237 -233 -226 -220 -214
-208 -206 -201 -190 -187 -184 -177 -171 -159 -156
-153 -151 -145 -138 -135 -127 -126 -121 -117 -115
-106 -101 -97 -93 -92 -85 -84 -82 -81 -81
-78 -62 -56 -55 -54 -42 -41 -41 -34 -28
-22 -22 -21 -5 -1 2 17 22 28 48
48 51 53 57 60 64 66 71 76 77
85 86 87 90 103 107 115 128 132 143
153 154 158 173 175 180 183 183 188 190
194 201 217 227 229 231 235 237 240 242
```

取N=1000

```
❯ g++ gen_test_case.cpp -o gen && ./gen && g++ quick_sort.cpp -o quicksort&& ./quicksort
Time used 0.146 miliseconds
```

```
1000
57 -1 -177 -92 180 22 -206 128 173 -41
190 -85 242 -208 237 -247 77 -21 90 -138
53 -81 -41 -93 -190 183 -151 28 66 85
-153 76 -238 17 60 -117 229 -101 -171 71
217 -78 143 86 235 -5 -22 -159 -56 107
-24 [illegible] 126 -54 -220 153
-28 [illegible] 27 158 -22 183
48 231 -115 -237 115 64 -187 -214 175 -81
-135 -156 -121 -249 -233 -55 -145 154 201 48
-62 -127 -245 132 2 -184 -34 87 188 -106
-249 147 121 78 -113 108 -73 147 44 154
```

/Users/tongchen/Library/Mobile Documents/com~apple~CloudDocs/Documents/Code/Algorithm/exp1_sort/quicksort.in • Modified

```
-141 -19 -5 -75 -115 48 -108 149 218 162
10 -193 -156 -242 145 218 -137 -220 -244 212
192 115 -168 102 17 71 -55 -38 -79 151
-160 81 -12 -193 -134 40 -110 -71 85 -244
222 -152 -155 69 204 -27 39 10 155 -124
-127 -244 63 20 -12 -156 -30 94 116 -216
-24 144 113 -12 94 -160 -200 -91 -127 197
135 -33 22 -211 -3 135 246 135 -127 170
-106 218 -15 165 -125 -216 -208 -239 -71 -98
-6 45 68 146 -58 65 241 -217 -81 147
-197 -203 75 -40 -88 -39 58 127 11 -125
85 118 245 26 17 189 124 81 -194 51
122 -190 -156 234 5 39 157 -235 -57 19
-70 205 105 -244 213 -248 -74 -142 -1 81
94 -112 58 247 -99 99 -47 -19 -219 -236
169 25 -241 -70 179 -27 -196 10 -13 -205
```

```
-250 -250 -250 -249 -249 -249 -249 -249 -249 -
-247 -247 -247 -247 -247 -246 -246 -245 -244 -
-244 -244 -244 -243 -242 -242 -242 -241 -241 -
-240 -240 -239 -239 -239 -238 -238 -237 -236 -
-236 -236 -236 -235 -235 -235 -235 -234 -234 -
-233 -233 -232 -232 -231 -231 -231 -230 -230 -
-226 -226 -225 -225 -225 -224 -222 -222 -222 -
-221 -220 -220 -219 -219 -219 -218 -217 -217 -
-217 -216 -216 -216 -216 -215 -215 -214 -214 -
-213 -212 -211 -211 -210 -210 -208 -208 -208 -
-207 -207 -207 -206 -206 -206 -206 -205 -205 -
```

```
-204 -204 -203 -202 -202 -202 -201 -201 -201 -
-200 -200 -199 -199 -198 -197 -197 -197 -197 -
-196 -196 -196 -195 -195 -195 -194 -193 -193 -
-193 -193 -190 -190 -190 -190 -189 -188 -187 -
-187 -186 -186 -186 -185 -185 -184 -184 -183 -
-180 -180 -179 -177 -176 -175 -174 -174 -171 -
-171 -170 -170 -170 -169 -169 -168 -167 -166 -
-164 -163 -163 -162 -161 -161 -161 -161 -160 -
-160 -159 -159 -159 -158 -158 -157 -157 -156 -
-156 -156 -156 -155 -154 -154 -153 -153 -153 -
-152 -151 -150 -149 -149 -148 -147 -147 -147 -
-145 -144 -143 -143 -143 -142 -142 -142 -141 -
-140 -139 -138 -138 -138 -138 -137 -137 -137 -
-136 -135 -135 -134 -134 -134 -133 -130 -129 -
-129 -128 -127 -127 -127 -127 -127 -127 -126 -
-125 -125 -125 -125 -124 -123 -123 -123 -122 -
-121 -121 -120 -120 -119 -118 -117 -117 -117 -
```

可见，当N分别为5，100，1000时，使用时间分别为0.001ms,0.013ms,0.146ms，时间复杂度符合O（nlogn）的特征

4.实验心得和总结

归并排序一开始我用递归实现，后面写实验报告的时候才发现要用迭代实现。所以基本上是两个方法都写了，调试了一遍。两种方式写下来发现从编写程序的角度来看，递归的实现思路更加简单直接，但是会对程序调用栈有比较大的资源占用压力。迭代通过循环帮助减少函数调用和资源占用，但边界条件的设定和调试非常费时费力。两种方法各有各的优点，也有各自的不足。

快速排序的实现相对归并排序而言简单一些，无论是划分（partition）函数的构建还是递归的调用都非常符合直觉。

通过这次实验，我熟练掌握了归并排序的两种实现和快速排序的实现，更好理解了分治法的思想。