**算法分析与设计**

**作业3**

王汝芸

201711010202

计工本1702

2019年9月17日

1. **给出基于比较的排序算法的复杂度表。**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 选择法 | 冒泡法 | 插入排序 | 快速排序 | 合并排序 | 堆排序 |
| 最快情况复杂度 | O() | O() | O() | O() | O(nlogn) | O(nlogn) |
| 平均情况复杂度 | O() | O() | O() | O(nlogn) | O(nlogn) | O(nlogn) |
| 最好情况复杂度 | O() | O(n) | O(n) | O(nlogn) | O(nlogn) | O(nlogn) |
| 空间复杂度 | O(1) | O(1) | O(1) | O(n)  O(logn) | O(n) | O(1) |

1. **给出插入排序、堆排序、归并排序和快速排序算法的思想描述、伪代码及复杂度。**

* **插入排序**

1. 基本思路

对于给定的数字序列从前往后操作。先考虑前两个数字，如果第2个数字小于第1个数字，则将其与第1个数字进行交换，这样这两个数字就成为有序的了。接下来暂存第3个数字，并将其与第2个数字相比较如果第2个数字小于第3个数字，则第3个数字就是前三个数字中最大的，只要将其放回原位就可以了。如果第2个数字大于第3个数字，则将其移到3号位置上。再看第1个数字，如果它比第3个数字小，则将第3个数字放到2号位置上，这时3个数字就排好了。如果第1个数字也比第3个数字大，则将其移到2号位置上，此后再将第3个数字放到1号位置上，则该种情况的3个数字也就排好了。

这种方法每次考虑一个新加进来的数字时，充分地利用此前已经排序的结果，从而达到提高排序效率的目的。

1. 伪代码

01 **for** i **=** 2 to n

02 x **=** a**[**i**];**p **=** 1**;**

03 **for** j **=** i**-**1 down to 1

04 **if** a**[**j**]>**x then

05 a**[**j**+**1**]**←a**[**j**]**

06 **else**

07 p **=** j**+**1**;break;**

08 end **if**

09 end **for**

10 a**[**p**]**←x

11 end **for**

1. 复杂度

最坏情况复杂度：O()

平均情况复杂度：O()

最好情况复杂度：O(n)

空间复杂度：O(1)

* **堆排序**

1. 基本思路

先用MaxHeapify算法将输入的数组创建为一个堆。

交换堆中的第一个元素和最后一个元素，则新的最后一个元素即是已经排好序的元素。

将堆的大小减1，并对第1个元素执行SiftDown操作。

继续这一过程，当堆的大小减到1时，即完成了对数组的排序。

1. 伪代码

01 MaxHeapify**(**A**)**

02 **for** i**=**n downto 2

03 A**[**1**]** ↔ A**[**i**]**

04 SiftDown**(**A**[**1…i**-**1**],**1**)**

05 end **for**

1. 复杂度

最坏情况复杂度：O(nlogn)

平均情况复杂度：O(nlogn)

最好情况复杂度：O(nlogn)

空间复杂度：O(1)

* **归并排序**

1. 基本思路

给定两个相邻的分别已经排好序的数组申请临时空间，使其大小为两个已经排序数组大小之和，该空间用来存放合并后的序列。

设定两个指针，最初位置分别为两个已排序数组的起始位置比较两个指针所指向的元素，选择相对小的元素放入到合并空间，并移动指针到下一位置重复步骤4直到某一指针到达序列尾将有剩余元素的数组中的元素复制到合并数组末尾。

1. 伪代码

01 i0 **=** low**,**i1 **=** mid **+**1

02 **for** j **=** low to high

03 **if** i0 **<** mid **and(**A**[**i0**] <=** A**[**i1**]** **or** i1 **>=** high**)**

04 B**[**j**] =** A**[**i0**]**

05 i0 **=** i0 **+** 1

06 **else**

07 B**[**j**] =** A**[**i1**]**

08 i1 **=** i1 **+** 1

09 end **if**

10 end **for**

1. 复杂度

最坏情况复杂度：O(nlogn)

平均情况复杂度：O(nlogn)

最好情况复杂度：O(nlogn)

空间复杂度：O(n)

* **快速排序**

1. 基本思路

使用分治策略把待排序数据序列分为两个子序列，步骤为：

1.挑出一个元素，称为“枢纽"（pivot）元素。

2.分区（partition）操作：将所有比枢轴元素小的元素放在基准前面，所有比它大的元素放在其后面（相等的元素可放到任一边）。

3.将小于基准值的子序列和大于基准值的子序列分别用步骤2递归。当序列长度变为0或1时结束递归。

1. 伪代码

01 quicksort**(**A**,**lo**,**hi**)**

02 **if** lo**<**hi

03 p **=** partition**(**A**,**lo**,**hi**)**

04 quicksort**(**A**,**lo**,**p**-**1**)**

05 quicksort**(**A**,**p**+**1**,**hi**)**

06 end **if**

07

08

09 partition**(**A**,**lo**,**hi**)**

10 pivot **=** A**[**hi**]**

11 i **=** lo

12 **for** j **=** lo to hi**-**1

13 **if** A**[**j**] <=** pivot

14 swap A**[**i**]** with A**[**j**]**

15 i **=** i**+**1

16 end **if**

17 swap A**[**i**]** with A**[**hi**]**

18 **return** i

1. 复杂度

最坏情况复杂度：O()

平均情况复杂度：O(nlogn)

最好情况复杂度：O(nlogn)

空间复杂度：O(n)O(logn)

1. **在CAAIS上学习和理解插入排序、堆排序、归并排序和快速排序算法。**
2. **在CAAIS上选择8个以上的数字组，以20%的交互操作插入排序、堆排序、归并排序和快速排序算法，并保存交互结果。**
3. **编程实现插入排序、堆排序、归并排序和快速排序算法。**

* **插入排序**

01 '''

02 插入排序

03 '''

04

05 **import** numpy **as** np

06

07 a **=** np**.**random**.**randint**(**10**,**size**=**10**).**tolist**()**

08 **print(**a**)**

09

10 **for** i **in** **range(**1**,** **len(**a**)):**

11 key **=** a**[**i**]**

12 j **=** i **-** 1

13 **while** j **>=** 0 **and** key **<** a**[**j**]:**

14 a**[**j **+** 1**] =** a**[**j**]**

15 j **-=** 1

16 a**[**j **+** 1**] =** key

17

18 **print(**a**)**

* **堆排序**

01 '''

02 堆排序

03 '''

04

05 **import** numpy **as** np

06 **from** heapq **import** heappush**,** heappop

07

08 a **=** np**.**random**.**randint**(**10**,**size**=**10**).**tolist**()**

09 **print(**a**)**

10

11 # 建堆

12 **def** MaxHeapify**(**a**):**

13 heap **= []**

14 **for** item **in** a**:**

15 heappush**(**heap**,**item**)**

16 heap**.**sort**(**reverse**=True)** # 降序

17 **return** heap

18

19 **def** SiftDown**(**H**,**i**):**

20 i **+=** 1

21 done **=** **False**

22 **if** 2**\***i**>len(**H**):**

23 **return** H

24 **while True:**

25 i **=** 2**\***i

26 **if** i**+**1**<=len(**H**)** **and** H**[**i**+**1**]>**H**[**i**]:**

27 i **=** i**+**1

28 **if** H**[int(**i**/**2**)]<**H**[**i**]:**

29 H**[**i**],**H**[int(**i**/**2**)] =** H**[int(**i**/**2**)],**H**[**i**]**

30 **else:**

31 done **=** **True**

32 **if** 2**\***i**>len(**H**)** **or** done**:**

33 **return** H

34

35

36 **def** HeapSort**(**a**):**

37 a **=** MaxHeapify**(**a**)**

38 **for** i **in** **range(len(**a**)-**1**,-**1**,-**1**):**

39 a**[**0**],**a**[**i**] =** a**[**i**],**a**[**0**]**

40 SiftDown**(([**0**]+**a**[**0**:]),**0**)**

41 a **= [**a**[-**1**]]+**a**[**0**:-**1**]**

42 **return** a

43

44 a **=** HeapSort**(**a**)**

45 **print(**a**)**

* **归并排序**

01 '''

02 归并排序

03 '''

04

05 **import** numpy **as** np

06

07 a **=** np**.**random**.**randint**(**10**,**size**=**10**).**tolist**()**

08 **print(**a**)**

09

10 **def** MergeSort**(**a**):**

11 **if** **len(**a**)<=**1**:**

12 **return** a

13 middle **=** **int(len(**a**)/**2**)**

14 left **=** MergeSort**(**a**[:**middle**])**

15 right **=** MergeSort**(**a**[**middle**:])**

16 **return** Merge**(**left**,**right**)**

17

18 **def** Merge**(**left**,**right**):**

19 c **= []**

20 h **=** j **=** 0

21 **while** j **<** **len(**left**)** **and** h **<** **len(**right**):**

22 **if** left**[**j**] <** right**[**h**]:**

23 c**.**append**(**left**[**j**])**

24 j **+=** 1

25 **else:**

26 c**.**append**(**right**[**h**])**

27 h **+=** 1

28 **if** j**==len(**left**):**

29 c **+=** right**[**h**:]**

30 **else:**

31 c **+=** left**[**j**:]**

32 **return** c

33

34 a **=** MergeSort**(**a**)**

35 **print(**a**)**

* **快速排序**

01 '''

02 快速排序

03 '''

04

05 **import** numpy **as** np

06 **import** time

07 **from** heapq **import** heappush**,** heappop

08

09 a **=** np**.**random**.**randint**(**60000000**,**size**=**60000000**).**tolist**()**

10 **print(**"快速排序"**)**

11 **print(**"size = 60,000,000"**)**

12 time\_start**=**time**.**time**()**

13

14 **def** Partition**(**arr**,**low**,**high**):**

15 i **= (**low **-** 1**)** # 最小元素索引

16 pivot **=** arr**[**high**]**

17

18 **for** j **in** **range(**low**,** high**):**

19

20 # 当前元素小于或等于 pivot

21 **if** arr**[**j**] <=** pivot**:**

22 i **=** i **+** 1

23 arr**[**i**],** arr**[**j**] =** arr**[**j**],** arr**[**i**]**

24

25 arr**[**i **+** 1**],** arr**[**high**] =** arr**[**high**],** arr**[**i **+** 1**]**

26 **return** **(**i **+** 1**)**

27

28 **def** QuickSort**(**a**,**low**,**high**):**

29 **if** high **>** low**:**

30 k **=** Partition**(**a**,**low**,**high**)**

31 QuickSort**(**a**,**low**,**k**-**1**)**

32 QuickSort**(**a**,**k**+**1**,**high**)**

33

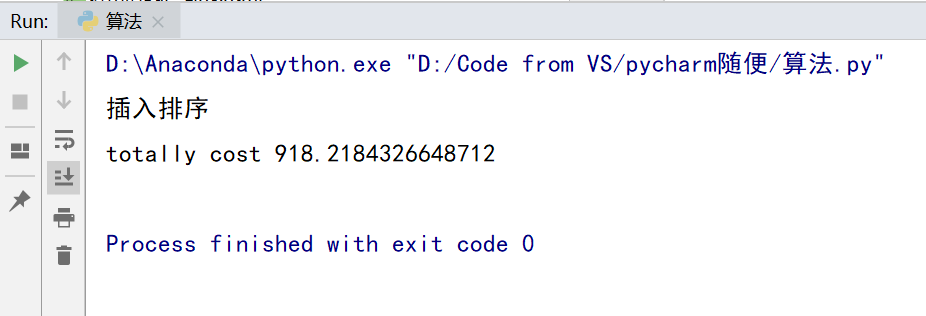
34 QuickSort**(**a**,**0**,len(**a**)-**1**)**

35 time\_end**=**time**.**time**()**

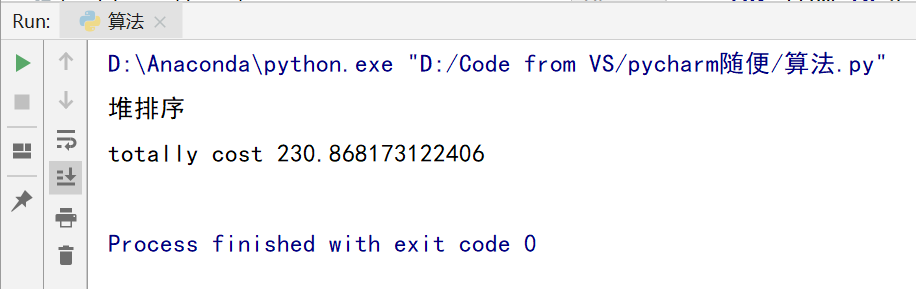
36 **print(**'totally cost'**,**time\_end**-**time\_start**)**

1. **挑战：随机生成10万个数，查看插入排序、堆排序、归并排序和快速排序算法的执行时间。**

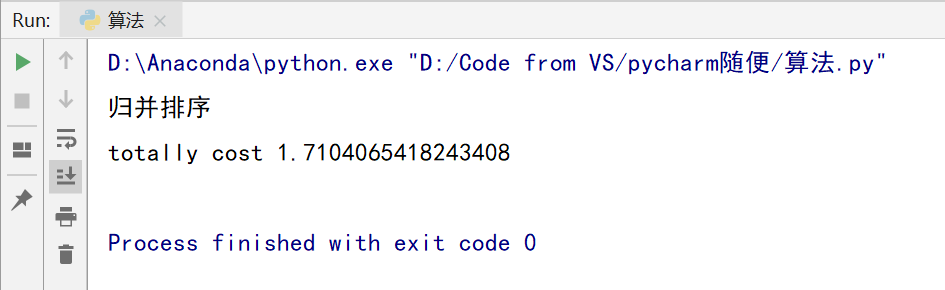
* **插入排序**



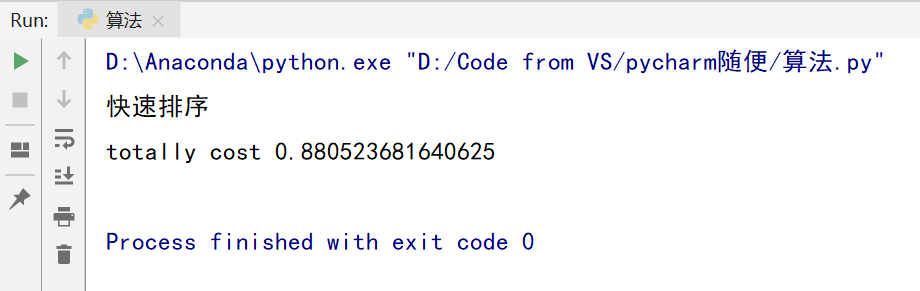
* **堆排序**



* **归并排序**

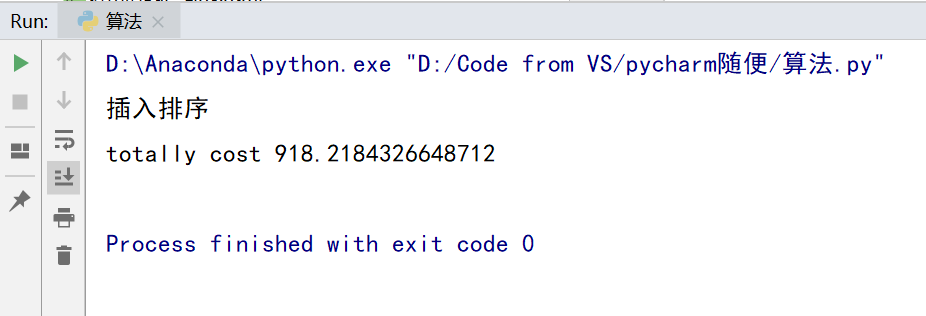


* **快速排序**

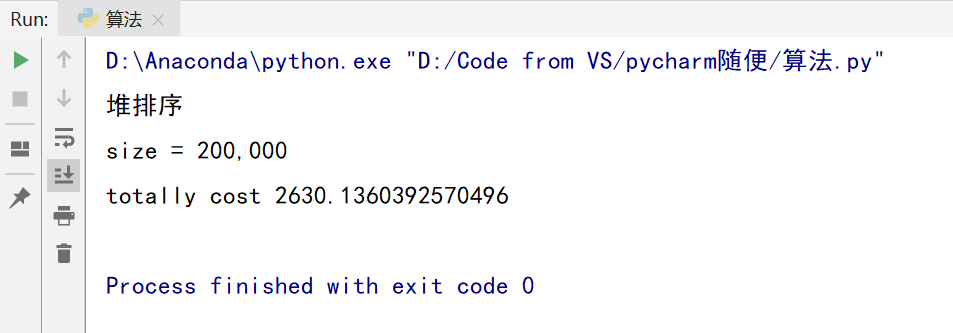


1. **挑战：查看自己机器上10分钟内用插入排序、堆排序、归并排序和快速排序算法分别可以完成的排序数据量。**

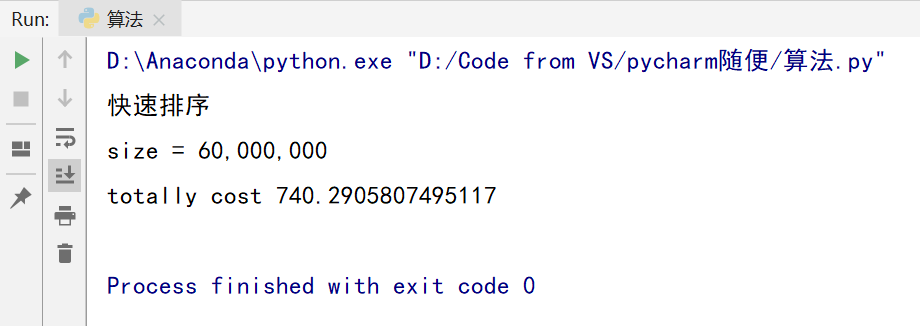
* **插入排序**



* **堆排序**



* **归并排序**
* **快速排序**



1. **挑战：插入排序算法中如果使用二分搜索查找元素的插入位置，是否可以改进算法复杂性。**