华东师范大学数据科学与工程学院上机实践报告

课程名称：算法设计与分析	年级：21级	上机实践成绩：
指导教师：金澈清	**姓名**：包亦晟	
上机实践名称：顺序统计量	**学号**：10215501451	**上机实践日期**：2023.3.30
上机实践编号：No.5	**组号**：1-451	

一、目的

1．熟悉算法设计的基本思想

2．掌握随机选择算法（rand select）的方法

3．掌握选择算法（SELECT）的方法

二、实验内容

1. 编写随机选择算法和SELECT算法；
2. 随机生成1e2、1e3、1e4、1e5、1e6个数，使用随机选择算法和SELECT算法找到第0.5N大的数输出，并画图描述不同情况下的运行时间差异；
3. 随机生成1e6个数，使用随机选择算法和SELECT算法找到第0.2N、0.4N、0.6N、0.8N大的数输出，并画图描述不同情况下的运行时间差异；
4. 递增生成1e2、1e3、1e4、1e5、1e6个数，使用随机选择算法和SELECT算法找到第0.5N大的数输出，并画图描述不同情况下的运行时间差异；
5. 随机生成1e2、1e3、1e4、1e5、1e6个数，使用merge sort找到第0.5N大的数输出，并画图描述不同情况下的运行时间差异；
6. 对比随机选择算法和SELECT算法以及merge sort。

三、使用环境

推荐使用C/C++集成编译环境。

四、实验过程

随机选择算法

随机选择算法是以快速排序为基础的一种算法。它与快速排序一样，也需要对输入数组进行递归划分。但是，也有一点与快速排序不同，它仅仅处理划分的其中一边，而快速排序是需要处理划分的两边的。这一点也导致了两者平均情况时间复杂度的不同：快速排序的平均情况时间复杂度为$\Theta(nlgn)$，而随机选择的平均情况时间复杂度为$\Theta(n)$。

快速选择算法需要使用到之前所介绍过的randomized-partition，因此随机选择是一个随机算法，它需要用到随机数生成器。

随机选择的基本步骤如下：

1. 随机产生一个主元
2. 利用partition将主元放到它应该在的位置，使得主元左侧的元素都小于主元，主元右侧的元素都大于主元
3. 比较k与主元左侧元素数量（包括主元）
    - 相等 直接返回
    - 大于 那么到主元右侧（不包括主元）继续进行递归 要注意修改k的值
    - 小于 那么到主元左侧（不包括主元）继续进行递归

以下是算法导论中所给出的伪代码：

```
RANDOMIZED-SELECT(A, p, r, i)
1  if p=r
2     then return A[p]
3  q ← RANDOMIZED-PARTITION(A, p, r)
4  k ← q−p+1
5  if i=k              ▷ the pivot value is the answer
6     then return A[q]
7  elseif i<k
8     then return RANDOMIZED-SELECT(A, p, q−1, i)
9  else return RANDOMIZED-SELECT(A, q+1, r, i−k)
```

错误

我在写randomized-partition的时候出现了一些问题，导致我调试了好久，我这里暂且记录一下。以下是我最开始的错误版本：

```
int randomPartition(int l, int r) {
    default_random_engine e;
    uniform_int_distribution<int> u(l, r);
    e.seed(time(0));
    int t = u(e);
    swap(a[l], a[t]);
    int i = l - 1, j = r + 1, x = a[l];
    while (i < j) {
        do i++; while (a[i] < x);
        do j--; while (a[j] > x);
        if (i < j) swap(a[i], a[j]);
    }
    return j;
}
```

第一个问题是：**i不应该从l - 1开始，而应该从l开始**。因为到了定义i的时候，我已经将随机选取的主元换到了a[l]，也就是说此时的a[l]，在循环中应该是不参与比较的。而我循环中的逻辑是do-while，是先++后再进行while判断。所以最开始定义i的时候应该定义在l处。

第二个问题是：**在while循环结束后，应该还要将a[l]和a[j]进行调换**，不然我们的主元就依然还会停留在l处，不会到达它应该在的位置。之所以会产生这种错误，是因为在原本的partition中我的i是从l - 1开始的，也就是说a[l]是参与比较的。但是在我最后的那份代码逻辑中，a[l]并没有参与比较，所以在循环结束后还需要换一次位置。

在实验报告中，我也就不附上最后正确的代码了。

虽然随机选择算法在平均情况下的时间复杂度是$\Theta(n)$的，但是在最坏情况下该算法是会退化到$\Theta(n^2)$的，因为假如你运气特别特别不好，那么在每次划分的时候都有可能按照剩下的元素中最大的一个进行划分，而划分操作是需要$\Theta(n)$时间复杂度的。当然，因为随机选择是一个随机化的算法，所以是不可能针对这种算法设计出一种特定的输入使得最坏情况出现的。只可能是因为运气太背了。该算法在平均情况下的性能是比较好的。

对于平均情况时间复杂度可以直接看《算法导论》，我也就不加以赘述了。

SELECT算法

这个算法是一个即使在最坏情况下时间复杂度是$O(n)$的选择算法。与randomized-select类似，SELECT算法也是需要通过对于输入数组进行递归划分来完成任务的。而不同的点在于，SELECT算法要保证对于数组的划分是一个比较好的划分。它并不是一个随机算法，所用到的是来自快速排序的确定性划分partition。但也不是一模一样地挪用，SELECT算法中的partition需要多加入一个参数，即将要被划分的主元元素。

该算法的基本思路因为算法导论上已经写的非常清楚了，我怕自己概括不当就直接附上算法导论中的内容：

1）将输入数组的 n 个元素划分为 $\lfloor n/5 \rfloor$ 组，每组 5 个元素，且至多只有一个组由剩下的 $n \bmod 5$ 个元素组成。

2）寻找 $\lceil n/5 \rceil$ 个组中每一组的中位数。首先对每组中的元素（至多为 5 个）进行插入排序，然后从排序过的序列中选出中位数。

3）对第 2 步中找出的 $\lceil n/5 \rceil$ 个中位数，递归调用 SELECT 以找出其中位数 x。（如果有偶数个中位数，根据约定，x 是下中位数。）

4）利用修改过的 PARTITION 过程，按中位数的中位数 x 对输入数组进行划分。让 k 比划分低区的元素数目多 1，所以 x 是第 k 小的元素，并且有 $n-k$ 个元素在划分的高区。

5）如果 $i=k$，则返回 x。否则，如果 $i<k$，则在低区递归调用 SELECT 以找出第 i 小的元素，如果 $i>k$，则在高区找第 $(i-k)$ 个最小元素。

基本思路就是这样，转换成代码的过程也不是怎么复杂，但依然有很多的注意点：

首先是partition，这里的partition不同于randomized-partition，又回归了快速排序中的确定性划分。下列为了方便说明，我这里给出正确的代码：

```
int partition(int l, int r, int p) {
    int i = l - 1, j = r + 1;
    while (i < j) {
        do i++; while (a[i] < p);
        do j--; while (a[j] > p);
        if (i < j) swap(a[i], a[j]);
    }
    //swap(a[j], a[p]);
    return j;
}
```

第一个不同点是多增加了一个**参数p**。要注意，这里的**p并非是主元的下标，而是主元的值本身**。而为什么可以这样呢？为什么可以直接传入主元的值，而不是主元的下标呢？答案就在第二、三个不同点。

第二个不同点是**i重新初始化为l - 1**。因为不再需要随机化，主元也不是固定选区的，a[l]也是要在循环中参与比较了，所以i要重新从l - 1出发。

第三个不同点是**不再需要在循环结束后进行swap**。因为i已经重新从l - 1出发，所以在循环的过程中主元就会到它应该在的位置，即该位置左边都小于等于主元，该位置右边都大于等于主元。

以上两点也解答了第一个不同点中的问题，因为主元的下标不再被需要了。

SELECT算法的时间复杂度我在报告中也不给出了，可以自行去《算法导论》中看。

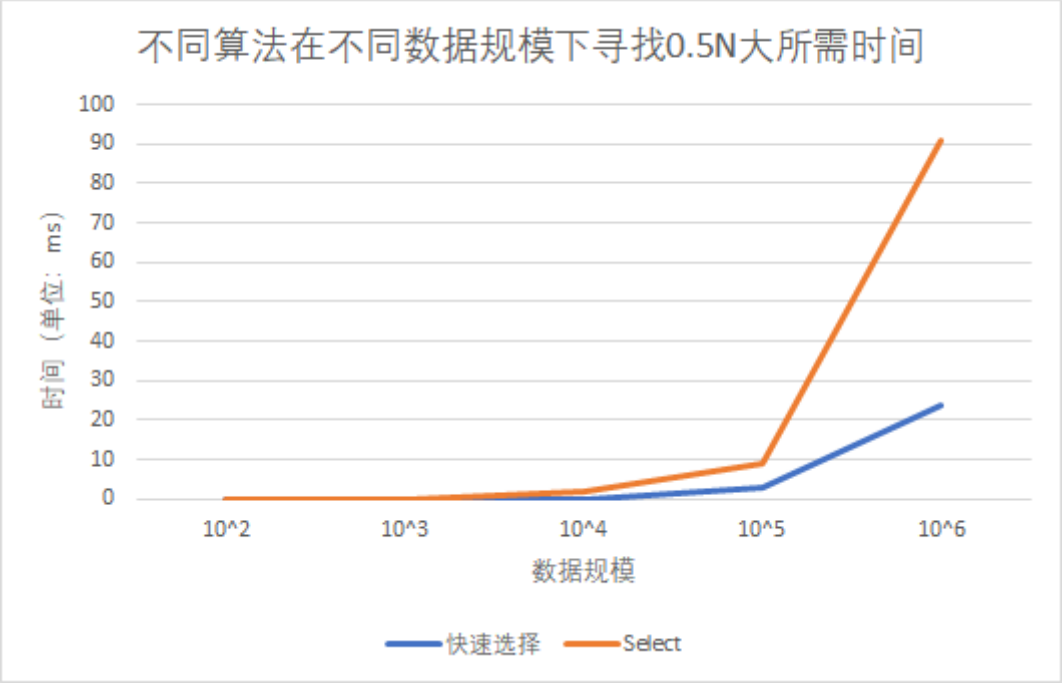
归并排序

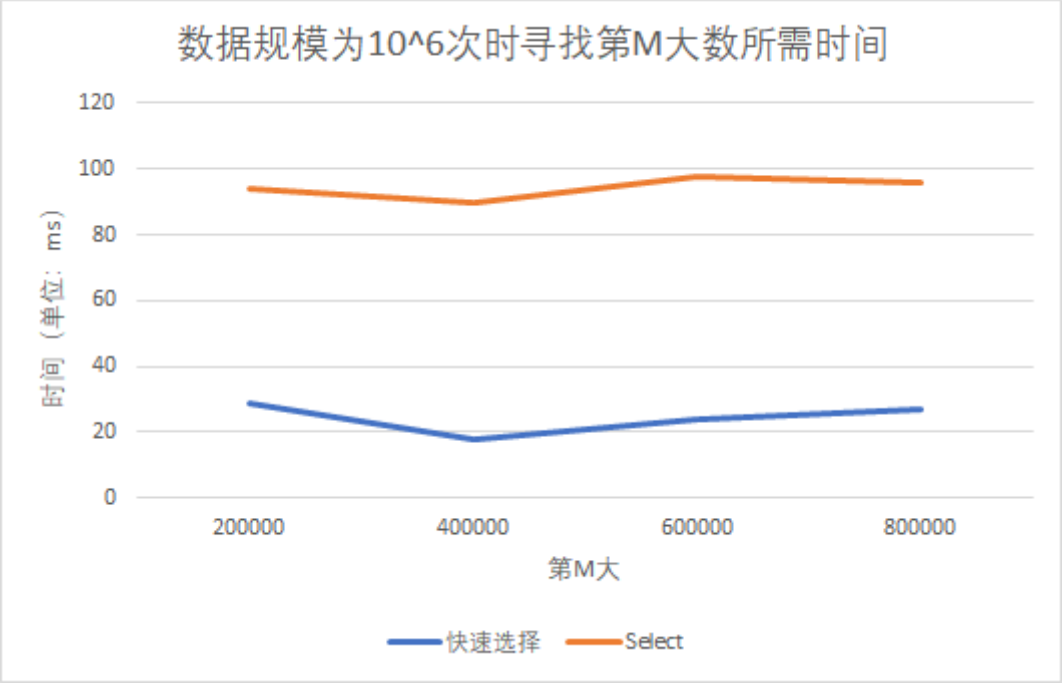
其实归并排序没什么好写的，因为之前的实验中已经写过了。但在本次实验中，我打算直接将归并排序的返回值设置为我们要求的第k大的数。然后，噩梦就发生了。一直得不到结果，怎么调试都不行。后来我恍然大悟，归并过程中，只有第一次的那个函数的返回值是有意义的，其他的即使返回了也没起到任何作用，但是这些其他的函数也是要返回a[r - kth + 1]的，这里就存在数组越界问题，我们需要用条件判断来规避。我最后的处理是：

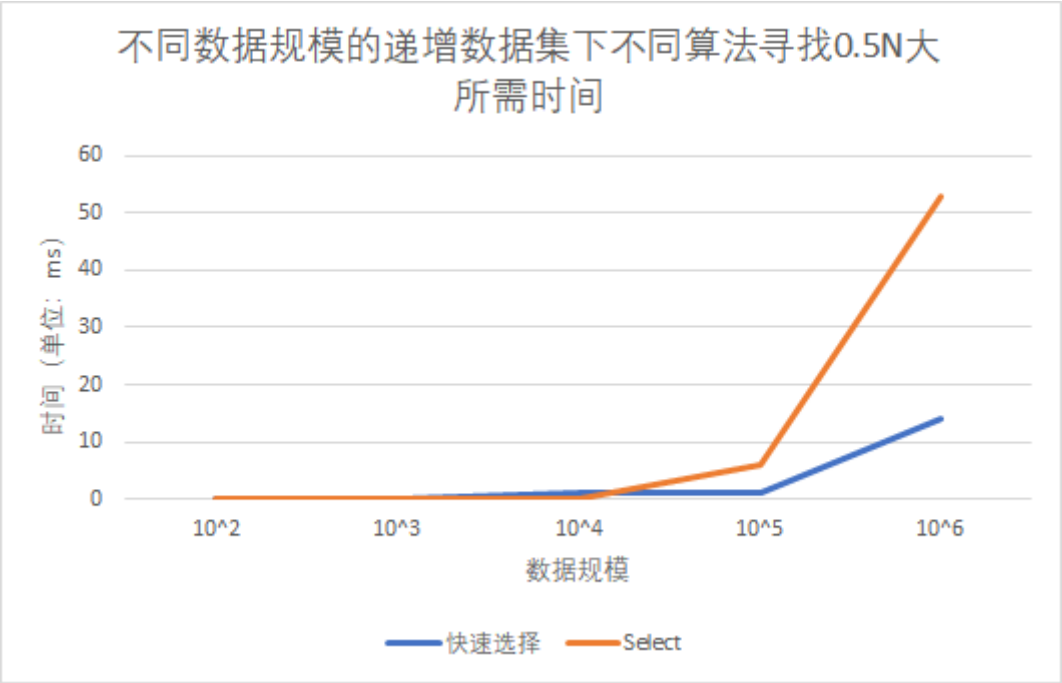
```
return r - kth + 1 >= 0 ? a[r - kth + 1] : 0;
```

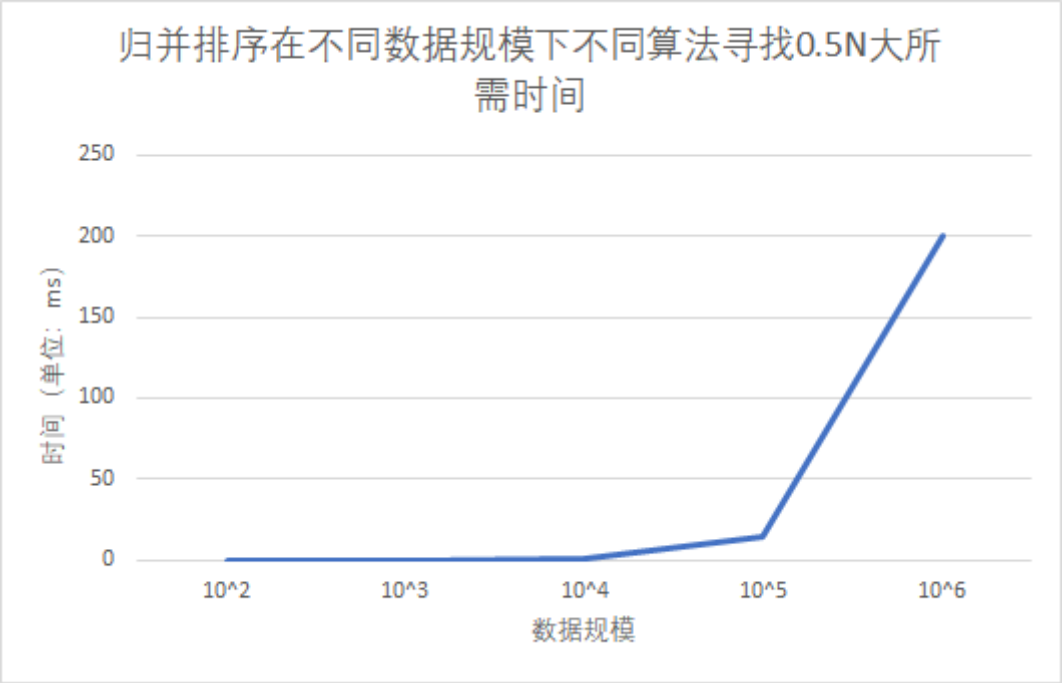
冒号后面的是任何数字都可以。

折线图






不同数据规模的递增数据集下不同算法寻找0.5N大所需时间
60
50
40
30
20
10
0
时间（单位：ms）
10^2
10^3
10^4
10^5
10^6
数据规模
快速选择
Select


归并排序在不同数据规模下不同算法寻找0.5N大所需时间
250
200
150
100
50
0
时间（单位：ms）
10^2
10^3
10^4
10^5
10^6
数据规模

实验结果

```
100 50
when N = 100,in finding the 50 largest element,
randomSelect takes 0s
Select takes 0s
PS D:\Documents\大二下\算法基础\hw5> cd "d:\Documents\大二下\
ator } ; if ($?) {
 .\dataGenerator }
1000 500
when N = 1000,in finding the 500 largest element,
randomSelect takes 0s
Select takes 0s
PS D:\Documents\大二下\算法基础\hw5> cd "d:\Documents\大二下\
ator } ; if ($?) {
 .\dataGenerator }
10000 5000
when N = 10000,in finding the 5000 largest element,
randomSelect takes 0s
Select takes 0.002s
PS D:\Documents\大二下\算法基础\hw5> cd "d:\Documents\大二下\
ator } ; if ($?) {
 .\dataGenerator }
100000 50000cd "d:\Documents\大二下\算法基础\hw5\" ; if ($?) {
Generator }
when N = 100000,in finding the 50000 largest element,
randomSelect takes 0.003s
Select takes 0.009s
```

```
1000000 500000
when N = 1000000,in finding the 500000 largest element,
randomSelect takes 0.024s
Select takes 0.091s
```

```
1000000 200000
when N = 1000000,in finding the 200000 largest element,
randomSelect takes 0.029s
Select takes 0.094s
```

```
1000000 400000
when N = 1000000,in finding the 400000 largest element,
randomSelect takes 0.018s
Select takes 0.09s
PS D:\Documents\大二下\算法基础\hw5> cd "d:\Documents\大二下\
ator } ; if ($?) { .\dataGenerator }
1000000 600000
when N = 1000000,in finding the 600000 largest element,
randomSelect takes 0.024s
Select takes 0.098s
PS D:\Documents\大二下\算法基础\hw5> cd "d:\Documents\大二下\
ator } ; if ($?) { .\dataGenerator }
1000000 800000
when N = 1000000,in finding the 800000 largest element,
randomSelect takes 0.027s
Select takes 0.096s
```

```
100 50
when N = 100,in finding the 50 largest element,
randomSelect takes 0s
Select takes 0s
PS D:\Documents\大二下\算法基础\hw5> cd "d:\Documents\大
ator } ; if ($?) { .\dataGenerator }
1000 500
when N = 1000,in finding the 500 largest element,
randomSelect takes 0s
Select takes 0s
PS D:\Documents\大二下\算法基础\hw5> cd "d:\Documents\大
ator } ; if ($?) { .\dataGenerator }
10000 5000
when N = 10000,in finding the 5000 largest element,
randomSelect takes 0.001s
Select takes 0s
```

```
10000 5000
when N = 10000,in finding the 5000 largest element,
randomSelect takes 0s
Select takes 0s
PS D:\Documents\大二下\算法基础\hw5> cd "d:\Documents\大二下
ator } ; if ($?) { .\dataGenerator }
100000 50000
when N = 100000,in finding the 50000 largest element,
randomSelect takes 0.001s
Select takes 0.006s
PS D:\Documents\大二下\算法基础\hw5> cd "d:\Documents\大二下
ator } ; if ($?) { .\dataGenerator }
1000000 500000
when N = 1000000,in finding the 500000 largest element,
randomSelect takes 0.014s
Select takes 0.053s
```

```
100 50
when N = 100,in finding the 50 largest element,
mergeSort takes 0s
PS D:\Documents\大二下\算法基础\hw5> cd "d:\Documents\大二
ator } ; if ($?) { .\dataGenerator }
1000 500
when N = 1000,in finding the 500 largest element,
mergeSort takes 0s
PS D:\Documents\大二下\算法基础\hw5> cd "d:\Documents\大二
ator } ; if ($?) { .\dataGenerator }
10000 5000
when N = 10000,in finding the 5000 largest element,
mergeSort takes 0.001s
PS D:\Documents\大二下\算法基础\hw5> cd "d:\Documents\大二
ator } ; if ($?) { .\dataGenerator }
100000 50000
when N = 100000,in finding the 50000 largest element,
mergeSort takes 0.014s
PS D:\Documents\大二下\算法基础\hw5> cd "d:\Documents\大二
ator } ; if ($?) { .\dataGenerator }
1000000 500000
when N = 1000000,in finding the 500000 largest element,
mergeSort takes 0.2s
```

五、总结

对上机实践结果进行分析，问题回答，上机的心得体会及改进意见。

随机选择算法的平均时间复杂度为$\Theta(n)$，但是在最坏情况下会退化到$\Theta(n^2)$;Select算法在最坏情况下的时间复杂度也是$O(n)$，看上去是要比随机选择算法更加优越的。但是，我们从之间的实验中也能感受到，光光看渐进时间复杂度是远远不够的，还有很多运行中的实际情况需要进行考虑。在本次实验中，随机选择算法是一个随机算法，直接通过随机数确定主元；而SELECT算法则需要通过插入排序递归取中位数的方式找到最合适的主元，预处理的时间要大于随机选择算法。所以我们可以看到，在实际运行中，SELECT算法是不如随机选择算法的。

从第二张图我们可以看到，当数据规模恒定的时候，无论寻找的是第几大的数，两种算法各自所需要的时间基本是差不多的。这是符合我们对算法的认识的。

而对比第一张图和第二张图，发现在递增数据集上两种算法是要比在随机数据集上的表现要稍微好一些的。我觉得其中一部分的原因是因为省去了partition过程中交换元素的过程。而在SELECT算法中还用到了插入排序，而根据之前的实验我们知道在数组原本就有序的情况下，插入排序的时间复杂度是能到到达$\Theta(n)$的。这应该是能够减少很多时间的。

和我们之前所学过的快速排序和插入排序等比较排序一样，本次实验中的两个算法也是基于比较来确定先后次序的。而经过数学证明，基于比较的排序算法是无法突破$\Theta(nlgn)$的。而之所以本次实验的两个算法的时间复杂度可以突破这个限度，是因为其实这两个算法并没有完整进行排序。

我想这可能就是本次实验安排我们去用归并排序求第k大的数的意义吧。如果想用归并排序求解的话，那就势必需要对整个数组进行排序，然后再取对应位置上的元素。而一旦需要完整的排序，那么时间复杂度就不可能突破$\Theta(nlgn)$。可以从我画的图中看到这一点。