《算法设计与分析》课程实验报告

分治法的应用

黎昱彤

学号：2022211414

班级：2022211312

计算机科学与技术

目录

1 实验介绍

1.1 实验原理

分治法的基本思想：

将一个规模为n的问题分解为k个规模较小的子问题，这些子问题相互独立且与原问题相同。递归地解决这些子问题，再将各子问题的解合并得到原问题的解。

1.2 实验任务

问题：

给定一个有n个数的无序的数组，查找其中排序好后的第k大的元素（排序从1开始）。比较不同规模下n=1000, 100000, 1000000到10000000,且k取在有序后的头部、中间和尾部时(比如，间隔为n/8的方式)各自的时间复杂度，比较相关的差异，画出曲线图。

基本工作：

1) 随机选择方式的实现
2) 确定的线性时间的选择算法

通过比较说明相关算法的特点，可能的改进（如果有，再实现加以验证）

可选扩展：

注意到算法执行时会改变元素的位置，试探究执行多次查找后可能的查找复杂度的变化（执行partition后数组相对会“更有序”一些）

1.3 实验环境

Windows11

Visual Studio Code

Python

2 算法设计

2.1 随机选择算法

快速排序算法的性能取决于划分的对称性，通过修改Partition()，采用随机取基准值的方式对数组进行划分，即**随机选择算法**。有可能避免一些极端的划分情况（每次划分时pivot为最小/最大元素）。

伪代码如算法1所示。

Algorithm1 随机选择算法

Function RandomPartition(A, p, r)

  pivot ← random(p, r)

  swap(A[pivot], A[p])

  return Partition(A, p, r)

End Function

Function RandomizedSelect(A, p, r, k)

  If p==r **Then**

    Return A[p]

  End If

  i ← RandomizedPartition(A, p, r)

  j ← i-p+1

  If k <= j **Then**

    Return RandomizedSelect(A, p, i, k)

  Else

    Return RandomizedSelect(A, i+1, r, k-j)

```
    End If
End Function
```

2.2 线性时间选择算法

线性时间选择算法基于快速排序，枢轴元素使用Median of Medians方法，避免了最坏情况（划分不平衡）的发生。

伪代码如算法2所示。

Algorithm2 线性时间选择算法

```
Function Partition(A, p, r, x)
    For j = p To r Do
        If A[j] == x Then
            Swap A[j] and A[r]
            Break
        End If
    End For
    x ← A[r]
    i ← p-1
    For j = p To r - 1 Do
        If A[j] <= x Then
            i ← i+1
            Swap A[i] and A[j]
        End If
    End For
```

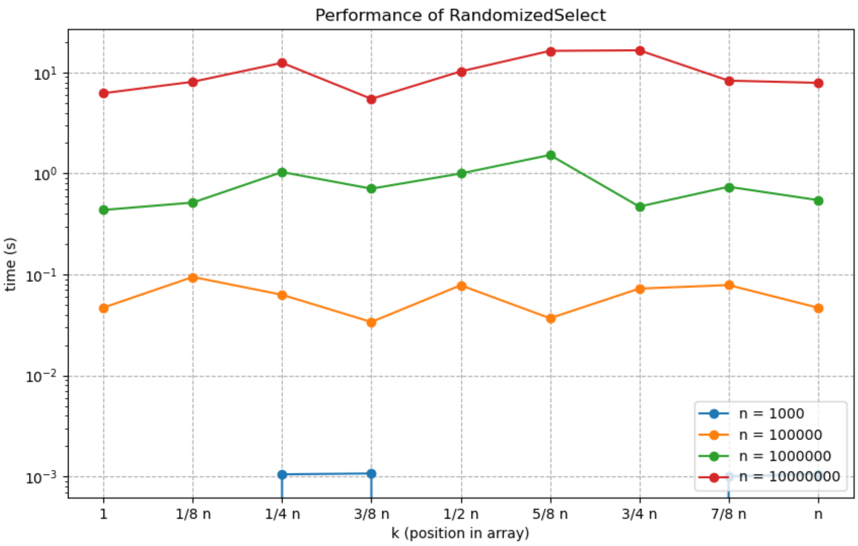
```
    Swap A[i + 1] and A[r]
    Return i + 1
End Function
Function Select(A, p, r, k)
    If  子数组长度小于75 Then
        简单排序数组
        Return A[p + k - 1]
    End If
    For i = 0 To (r-p-4)//5 Do
        每组最多5个数
        median ← 一组的中位数
        Swap A[p + i] and A[median_index]
    End For
    median_of_medians ← 各组中位数的中位数
    i ← Partition(A, p, r, median_of_medians)
    j ← i-p+1
    If k <= j Then
        Return Select(A, p, i, k)
    Else
        Return Select(A, i + 1, r, k - j)
    End If
End Function
```

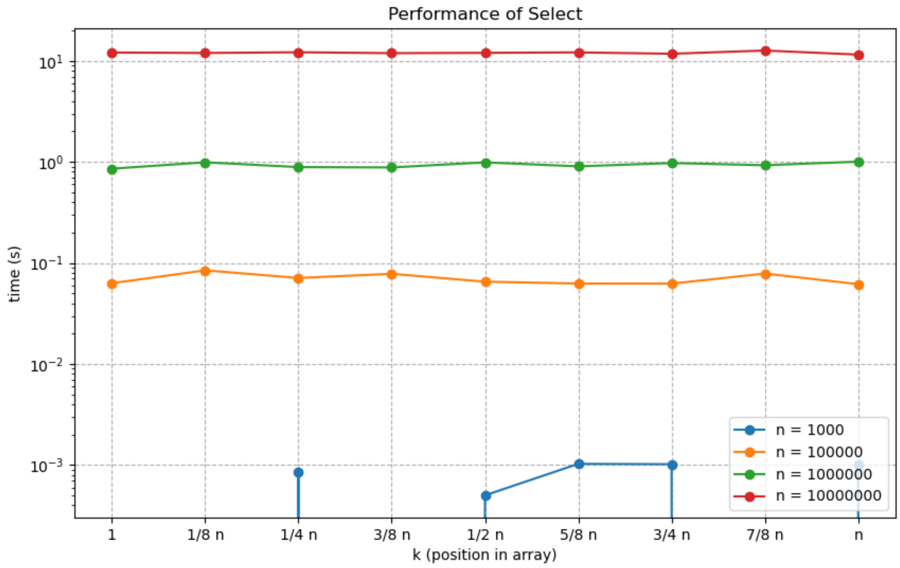
3 实验结果及分析

3.1 实验结果

```
n = 1000, k = 1, RandomizedSelect, time = 0.0000s
n = 1000, k = 1, Select, time = 0.0000s
n = 1000, k = 1/8 n, RandomizedSelect, time = 0.0000s
n = 1000, k = 1/8 n, Select, time = 0.0000s
n = 1000, k = 1/4 n, RandomizedSelect, time = 0.0010s
n = 1000, k = 1/4 n, Select, time = 0.0009s
n = 1000, k = 3/8 n, RandomizedSelect, time = 0.0011s
n = 1000, k = 3/8 n, Select, time = 0.0000s
n = 1000, k = 1/2 n, RandomizedSelect, time = 0.0000s
n = 1000, k = 1/2 n, Select, time = 0.0005s
n = 1000, k = 5/8 n, RandomizedSelect, time = 0.0000s
n = 1000, k = 5/8 n, Select, time = 0.0010s
n = 1000, k = 3/4 n, RandomizedSelect, time = 0.0000s
n = 1000, k = 3/4 n, Select, time = 0.0010s
n = 1000, k = 7/8 n, RandomizedSelect, time = 0.0010s
n = 1000, k = 7/8 n, Select, time = 0.0000s
n = 1000, k = n, RandomizedSelect, time = 0.0011s
n = 1000, k = n, Select, time = 0.0010s
n = 100000, k = 1, RandomizedSelect, time = 0.0470s
n = 100000, k = 1, Select, time = 0.0630s
n = 100000, k = 1/8 n, RandomizedSelect, time = 0.0944s
n = 100000, k = 1/8 n, Select, time = 0.0845s
n = 100000, k = 1/4 n, RandomizedSelect, time = 0.0629s
n = 100000, k = 1/4 n, Select, time = 0.0711s
n = 100000, k = 3/8 n, RandomizedSelect, time = 0.0338s
n = 100000, k = 3/8 n, Select, time = 0.0781s
n = 100000, k = 1/2 n, RandomizedSelect, time = 0.0781s
n = 100000, k = 1/2 n, Select, time = 0.0654s
n = 100000, k = 5/8 n, RandomizedSelect, time = 0.0369s
n = 100000, k = 5/8 n, Select, time = 0.0626s
n = 100000, k = 3/4 n, RandomizedSelect, time = 0.0724s
n = 100000, k = 3/4 n, Select, time = 0.0625s
n = 100000, k = 7/8 n, RandomizedSelect, time = 0.0785s
n = 100000, k = 7/8 n, Select, time = 0.0786s
n = 100000, k = n, RandomizedSelect, time = 0.0469s
n = 100000, k = n, Select, time = 0.0617s
```

```
n = 1000000, k = 1, RandomizedSelect, time = 0.4354s
n = 1000000, k = 1, Select, time = 0.8576s
n = 1000000, k = 1/8 n, RandomizedSelect, time = 0.5150s
n = 1000000, k = 1/8 n, Select, time = 0.9905s
n = 1000000, k = 1/4 n, RandomizedSelect, time = 1.0327s
n = 1000000, k = 1/4 n, Select, time = 0.8896s
n = 1000000, k = 3/8 n, RandomizedSelect, time = 0.7076s
n = 1000000, k = 3/8 n, Select, time = 0.8819s
n = 1000000, k = 1/2 n, RandomizedSelect, time = 0.9995s
n = 1000000, k = 1/2 n, Select, time = 0.9901s
n = 1000000, k = 5/8 n, RandomizedSelect, time = 1.5245s
n = 1000000, k = 5/8 n, Select, time = 0.9038s
n = 1000000, k = 3/4 n, RandomizedSelect, time = 0.4702s
n = 1000000, k = 3/4 n, Select, time = 0.9745s
n = 1000000, k = 7/8 n, RandomizedSelect, time = 0.7387s
n = 1000000, k = 7/8 n, Select, time = 0.9269s
n = 1000000, k = n, RandomizedSelect, time = 0.5453s
n = 1000000, k = n, Select, time = 1.0064s
n = 10000000, k = 1, RandomizedSelect, time = 6.2346s
n = 10000000, k = 1, Select, time = 12.1013s
n = 10000000, k = 1/8 n, RandomizedSelect, time = 8.0968s
n = 10000000, k = 1/8 n, Select, time = 11.9680s
n = 10000000, k = 1/4 n, RandomizedSelect, time = 12.4787s
n = 10000000, k = 1/4 n, Select, time = 12.1694s
n = 10000000, k = 3/8 n, RandomizedSelect, time = 5.4851s
n = 10000000, k = 3/8 n, Select, time = 11.9193s
n = 10000000, k = 1/2 n, RandomizedSelect, time = 10.2526s
n = 10000000, k = 1/2 n, Select, time = 12.0069s
n = 10000000, k = 5/8 n, RandomizedSelect, time = 16.3927s
n = 10000000, k = 5/8 n, Select, time = 12.1321s
n = 10000000, k = 3/4 n, RandomizedSelect, time = 16.5944s
n = 10000000, k = 3/4 n, Select, time = 11.7587s
n = 10000000, k = 7/8 n, RandomizedSelect, time = 8.3107s
n = 10000000, k = 7/8 n, Select, time = 12.6591s
n = 10000000, k = n, RandomizedSelect, time = 7.9056s
n = 10000000, k = n, Select, time = 11.5355s
```





注：算法是按第 k 小写的，但是绘图时做了 n-k 的处理，得到的还是第 k 大元素。

3.2 实验分析

1. 随机选择算法 (RandomizedSelect)

   平均时间复杂度：O(n)

   最坏时间复杂度：$O(n^2)$

   在每一层划分时会递归到 k 的一侧，所以 k 的位置不会明显影响算法的整体性能。

2. 线性时间选择算法 (Select)

   Median of Medians 方法能确保每次划分后的子数组大小不超过原数组大小的 7/10，避免了最坏情况下时间复杂度退化为 $O(n^2)$的情况，所以算法的时间复杂度为 O(n)

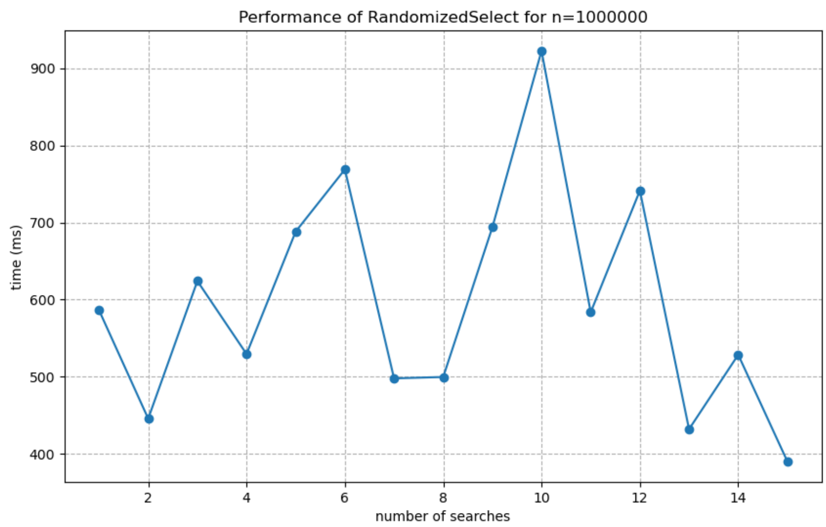
   选择算法的划分策略是固定的，中位数的中位数选取不会受到 k 的显著影响，所以 k 的位置不会明显影响算法的整体性能。

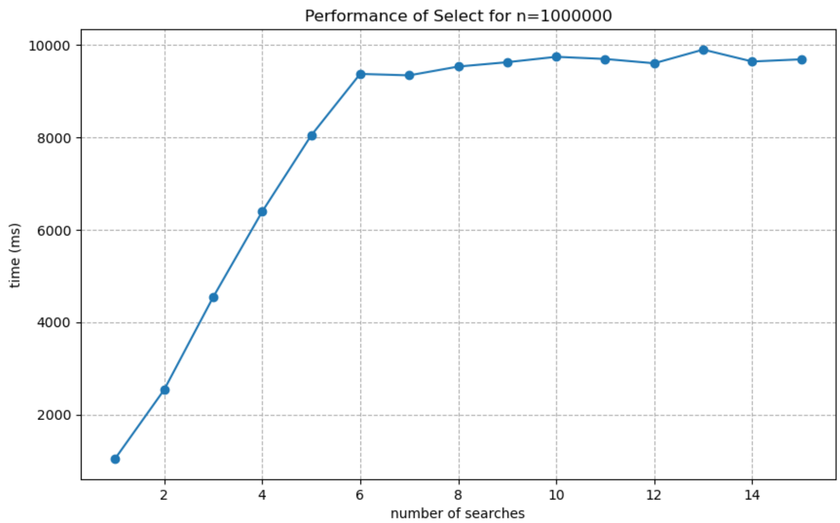
4 实验扩展

注意到算法执行时，Partition会改变元素的位置，将数组按pivot划分，这样逐步执行会让数组更“有序”（尽管非完全排序）。下面将探究执行多次查找后可能的查找复杂度的变化。

取n=1000000，取k=8，重复查找15次第k大元素。对源程序只需改动benchmark和plot，仍记录每次查找所需的时间，观察随着查找次数增加，运行时间的变化趋势。

实验结果如图：





结果表明：对**随机选择算法**，多次查找对执行时间无明显影响，说明算法性能独立于数组的顺序；对**线性时间选择算法**，随着查找次数的增加，执行时间显著上升，说明数组的有序性提升使得算法性能显著下降。

5 实验总结

本次实验验证了分治法在求解无序数组第k大元素问题中的两种算法的特点。随机选择算法对数组有序性不敏感，表现出良好的稳定性和适应性，适合用于多次查找任务。而线性时间选择算法在初始无序数组上表现良好，但当数组逐渐趋于有序时，其性能显著下降。整体而言加深了我对于分治法的理解，锻炼了我的探究与分析能力。

附录

实验关键源码

```
# 随机选择
def RandomizedPartition(A, p, r):
    def Partition(A, p, r):
        x = A[r]
        i = p-1
        for j in range(p, r):
            if A[j] <= x:
                i += 1
                A[i], A[j] = A[j], A[i]
        A[i+1], A[r] = A[r], A[i+1]
        return i+1

    pivot = random.randint(p, r)
    A[r], A[pivot] = A[pivot], A[r]
    return Partition(A, p, r)

def RandomizedSelect(A, p, r, k):
    if p == r:
        return A[p]
    i = RandomizedPartition(A, p, r)
    j = i-p+1
    if k <= j:
        return RandomizedSelect(A, p, i, k)
```

```python
    else:
        return RandomizedSelect(A, i+1, r, k-j)

# 线性时间选择
def Select(A, p, r, k):
    def Partition(A, p, r, x):
        for j in range(p, r+1):
            if A[j] == x:
                A[j], A[r] = A[r], A[j]
                break
        x = A[r]
        i = p-1
        for j in range(p, r):
            if A[j] <= x:
                i += 1
                A[i], A[j] = A[j], A[i]
        A[i+1], A[r] = A[r], A[i+1]
        return i+1

    if (r-p) < 75:
        A[p:r+1] = sorted(A[p:r+1])
        return A[p+k-1]

    for i in range((r-p-4)//5):
        # 取出 A[p+i*5:p+i*5+4]的中位数与 A[p+i]交换
        right = min(p+i*5+4, r)
        median = sorted(A[p+i*5 : right+1])[len(A[p+i*5 : right+1])//2]
        median_index = A.index(median, p+i*5, right+1)
        A[p+i], A[median_index] = A[median_index], A[p+i]
        # 求中位数的中位数
        median_of_medians = Select(A, p, p+(r-p)//5, (r-p)//10)
        i = Partition(A, p, r, median_of_medians)
        j = i-p+1
        if k <= j:
            return Select(A, p, i, k)
        else:
            return Select(A, i+1, r, k-j)
```