数据结构实验报告

姓名： 杨家玺 学号：U201717007 班级：软工 1703 班

实验二 求幂算法、求数组主元问题的求解

一、实验描述

1.求整数幂: 用基于 2 和 3 的方式分别写出算法来求 power(n, x)，分析两种算法的复杂程度，设计试验来验证你的想法。
2.求主元:设计并实现使用分治求数组的主元的算法。如果不用分治，通 过比较和计数，复杂程度是多少?

二、实验之前

1. 程序计时方法：在算法执行前于执行结束后分别执行 clock()，获取算法的运行 ticks,不除以 CLOCK_PER_SECOND,因为个人认为使用 ticks 会更加准确。
2. 设计了三个宏指令以简化程序

```
#define _TIME_INIT clock_t tic = 0;
#define _TIME_START tic = clock();
#define _TIME_END printf("%lu", clock() - tic);
```

3. 计时原则：同一规模多次试验取平均，规模渐进增长
4. 算法复杂度绘图：绘制规模-ticks 图，绘制 f(规模)-ticks 图，f 为关系函数。
5. 使用 Python 程序辅助统计并进行画图。

三、实验设计

1. 求整数幂：可用递归或迭代的方式实现
   设计函数 power(a, n)来计算 a^b，分别使用 2 和 3 为底数
   使用 2 为底数：T(N)=T(N/2)+C，故 T(N)=O(log2N)
   使用 3 为底数：T(N)=T(N/3)+C，故 T(N)=O(log3N)
2. 求主元：可采用分治或迭代法
   分治法：T(N)=T(N/2)+O(N)，故 T(N)=O(N)
   迭代法：在线算法，T(N)=O(N)

四、实验方案

1. 求整数幂
   （1）设置 a 为定值，输入 n
   （2）分别以 2 和 3 为底数计算 a^n
   （3）重复计算并记录运行时间
2. 求主元
   a. 分治法
      （1）使用书中给出的算法实现

（2）随机生成规模为 n 的数组，记录运行时间

b. 在线算法

（1）使用在线算法实现求主元问题

（2）随机生成规模为 n 的数组，记录运行时间

五、算法实现

1. 求整数幂

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include "utils.c"

long double power_2_rec(long double x, int n)
{
    if (0 == n)
        return 1;
    if (1 == n)
        return x;
    if (n & 1)
        return power_2_rec(x * x, n >> 1) * x;
    else
        return power_2_rec(x * x, n >> 1);
}

long double power_3_rec(long double x, int n)
{
    if (n == 0)
        return 1;
    if (n == 1)
        return x;
    if (n == 2)
        return x * x;
    if (n % 3 == 0)
        return power_3_rec(x * x * x, n / 3);
    if (n % 3 == 1)
        return power_3_rec(x * x * x, n / 3) * x;
    else
        return power_3_rec(x * x * x, n / 3) * x * x;
}

long double power_2(int x, int n)
{
    long double res = 1;
    long double base = x;
    while (n)
    {
        if (n & 1)
        {
            res *= base;
        }
        n >>= 1;
        base *= base;
    }
    return res;
}

long double power_3(int x, int n)
{
    long double res = 1;
    long double base = x;
    while (n)
    {
        if (n % 3 == 1)
        {
            res *= base;
        }
        if (n % 3 == 2)
        {
            res *= base * base;
        }
        n /= 3;
        base *= base * base;
    }
    return res;
}

int main(int argc, char **argv)
{
    _TIME_INIT
    int a = 2;
    int b = atoi(argv[1]);
    char mtd = argv[2][0];
```

```
    _TIME_START
    for (int i = 0; i < 100000; i++)
    {
        switch (mtd)
        {
        case '1':
            power_2(a, b);
            break;
        case '2':
            power_3(a, b);
            break;
        case '3':
            power_2_rec(a, b);
            break;
        case '4':
            power_3_rec(a, b);
            break;
        }
    }
    _TIME_END
    return 0;
}
```

2. 求数组主元

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include "utils.c"

int *rand_arr(int size, int l, int r)    // 产生随机数组
{
    srand(time(NULL));
    int *a = malloc(sizeof(int) * size);
    for (int i = 0; i < size; i++)
        a[i] = rand() % (l - r + 1) + l;
    return a;
}

void out_arr(int *a, int size)           // 输出数组
{
    printf("size : %d\n", size);
    for (int i = 0; i < size; i++)
        printf("%d ", a[i]);
    printf("\n");
}

int isPriority(int *a, int size, int cdt)     // 检查 cdt 是否为数组 a 的主元
{
    int count = 1;
    for (int i = 0; i < size - 1; i++)
        if (a[i] == cdt)
            count++;
    if (count > size / 2)
        return 1;
    else
        return 0;
}

int checkLast(int *a, int size)          // 检查 a 中的最后一个元素是不是主元
{
    return isPriority(a, size, a[size - 1]);
}

int findPriority_1(int *a, int size)     // 使用分治法寻找数组主元
{
    if (size == 0)
        return -1;
    if (size == 1)
        return a[0];
    int tmp[size];
    int t_size = 0;
    for (int i = 0; i < size; i += 2)
        if (a[i] == a[i + 1])
            tmp[t_size++] = a[i];
    int res = findPriority_1(tmp, t_size);
    if (size % 2 && res == -1 && checkLast(a, size))
        return a[size - 1];
    else if (isPriority(a, size, res))
        return res;
    return -1;
}

int findPriority_2(int *a, int size)     // 使用在线算法寻找数组主元
{
    int current = -1;
    int cnt = 0;
    for (int i = 0; i < size; i++)
```

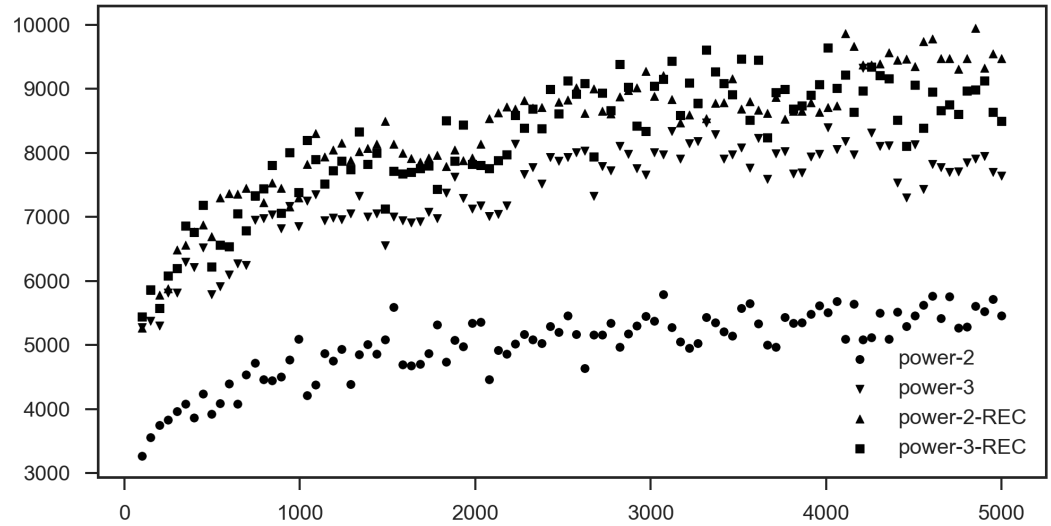
```
    {
        if (cnt == 0)
        {
            current = a[i];
            cnt++;
        }
        else
            cnt += (a[i] == current) ? 1 : -1;
    }
    if (isPriority(a, size, current))
        return current;
    return -1;
}

int main()
{
    int n = 10;
    int *a = rand_arr(n, 1, 10);
    // int a[10] = {1, 2, 2, 2, 3, 2, 1, 1, 2, 2};
    // out_arr(a, n);
    printf("ans = %d\n", findPriority_1(a, n));
    printf("ans = %d\n", findPriority_2(a, n));
    return 0;
}
```

五、实验结果与分析

1. 求幂算法

问题规模 n	Power-2	Power-3	Power-2-REC	Power-3-REC
100	3265.75	5277.6	5270.2	5443.2
595	4393.65	6096.1	7369.4	6533.95
1090	4377.4	7348.4	8301.4	7901
1585	4690.9	6942.05	7999.95	7672.95
2080	4460.5	7006.8	8540.7	7759.8
2575	5163.15	8003.6	9020.9	8919.8
3070	5791.6	7973.2	9213.4	9151.25
3565	5648.95	7763.2	8801.7	8512
4060	5684.3	8053.95	8738.45	9008.05
4555	5624.6	7434.45	9744.75	8383.4



可以看出，三个算法的增长趋势均符合对数的增长规律，但表中读出的信息显示以 2 为底数的求幂算法的执行速度最快，但分析显示理论上应该是以 3 为底数的求幂算法最快。个人猜想是因为在以 2 为底数的算法中使用了位运算与逻

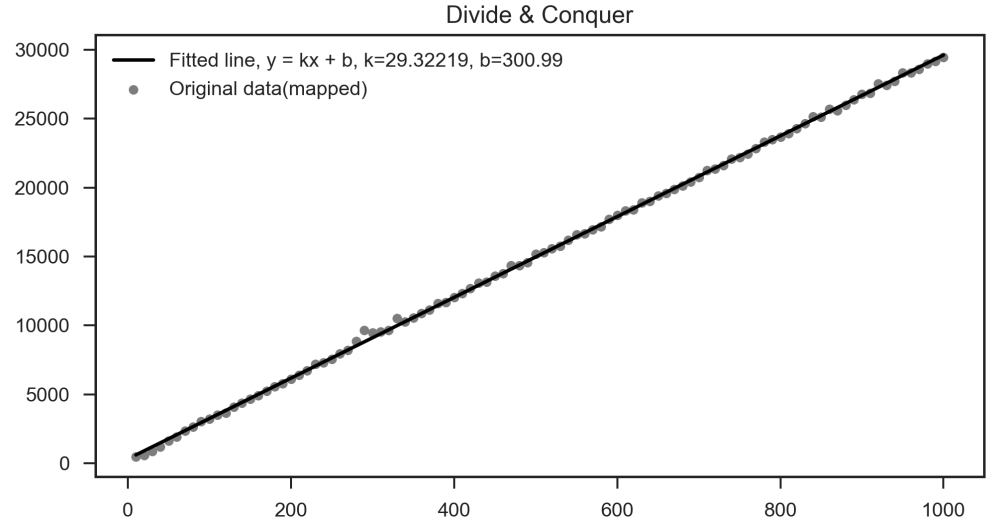
辑运算，这一部分接近硬件底层的运算加快了算法的进程。

总体上看实验效果与分析结果相吻合，但不同算法之间表现出的性能差距还需进一步研究。

2. 求数组主元

（1）分治法

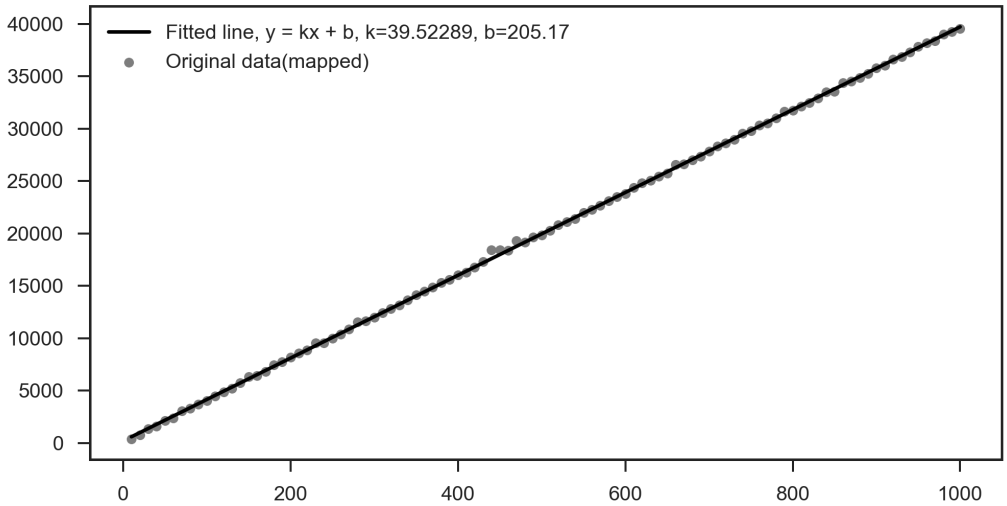
n	10	110	210	310	410	510	610	710	810	910
ticks	463.3	3498.15	6406.4	9547.25	12324.8	15286.35	18333.05	21271.5	23937.65	26842.0



可以看出，该算法拥有线性的时间复杂度，与理论分析完全吻合，T(N)=O(N)

（2）在线算法

n	10	110	210	310	410	510	610	710	810	910
ticks	386.3	4469.85	8577.2	12424.05	16301.9	20285.05	24405.55	28320.95	32160.7	36064.85



可以看出，该算法同样拥有线性的时间复杂度，与理论分析吻合。

（3）两算法比较

在相同的坐标轴尺度下，分治算法的拟合直线斜率小于在线算法，说明其性能较好。