数值计算与分析实验 1.2 实验报告

一、实验目的

通过本次实验加深对牛顿迭代法的理解，了解经典迭代算法。

二、实验步骤

本实验题要求用 C/C++语言或者汇编语言

1. 用牛顿迭代法实现 unsigned my_ isqrt (unsigned c) ;牛顿法的初始值必须严格满足公式 $2^{s-1} < \lfloor \sqrt{c} \rfloor \le 2^{s}$. 用二分查找法来确定初始值对区间 $[1,\ 2^{32})$ 中的所有整数，分别用 sqrt(double), my_ isqrt, isqrt2, isqrt3, isqrt4 计算他们的平方根.统计下面表中所需要的信息。其中 sqrt(double)是系统提供的平方根函数。

2. 根据此概率分布构造并程序实现最优查找二叉树.

在求迭代的初始值 $x_0 = 2^s$, $s = 0, 1, \cdots 16$. 如果初始值取这些值得概率分布如下

s	0	1	2	3	4	5	6	7	8
概率	0.0	$\frac{1}{128}$	$\frac{1}{128}$	$\frac{1}{64}$	$\frac{1}{32}$	$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{4}$	$\frac{1}{2}$
s	9	10	11	12	13	14	15	16	
概率	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

并且由此实现新的求整数平方根算法 my isqrt_ op.对区间 $[1,\ (2^8+\ 1)^2)$，$[(2^8+\ 1)^2,\ 2^{32})$ 的所有整数，用各种方法求其平方根.统计下面信息.

算法	是否有误差	平均用时	平均迭代次数
sqrt(double)			不适用
my_isqrt_op			
my_isqrt			
isqrt2			
isqrt3			
isqrt4			

三、实验环境

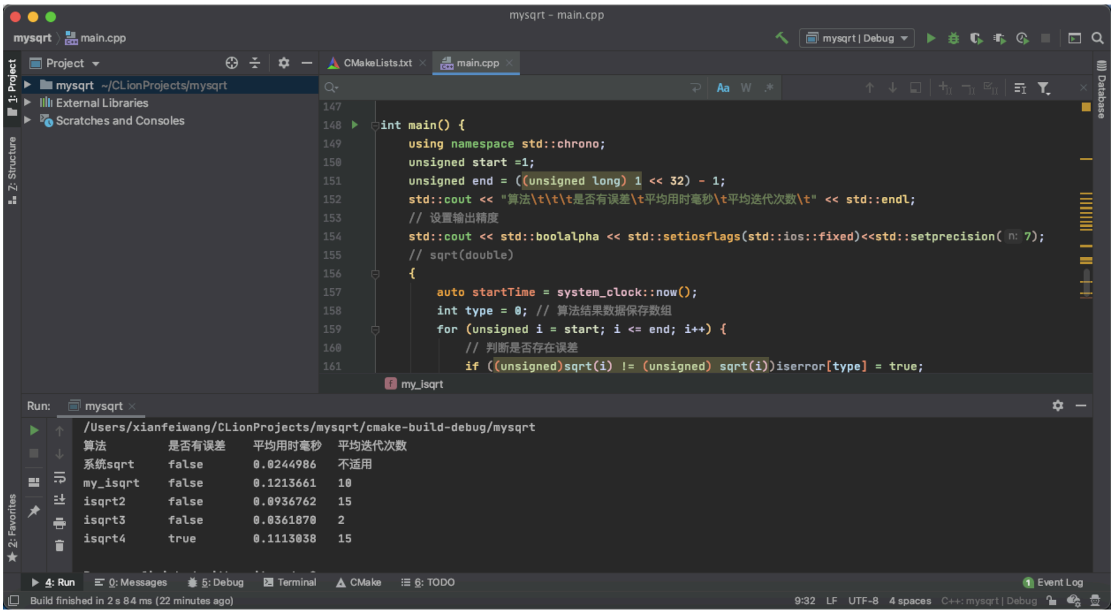
C++语言，macOS 11.0 Clion2020 Clang-LLVM 编译器

四、实验结果

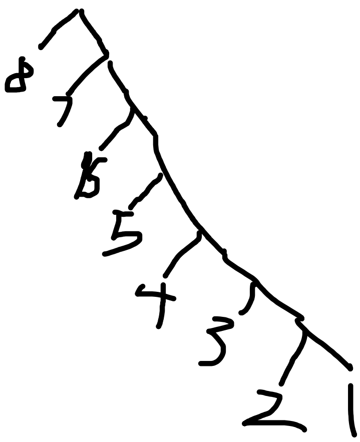
（1）自己编写了 unsigned my_isqrt(unsigned c)函数（完整运行代码见文末附录 1），按照文中

所提到的测试方法及另外几种算法的时间进行比较，结果如下：

算法	是否有误差	平均用时毫秒	平均迭代次数
系统 sqrt	false	0.0244986	不适用
my_isqrt	false	0.1213661	10
isqrt2	false	0.0936762	15
isqrt3	false	0.0361870	2
isqrt4	true	0.1113038	15



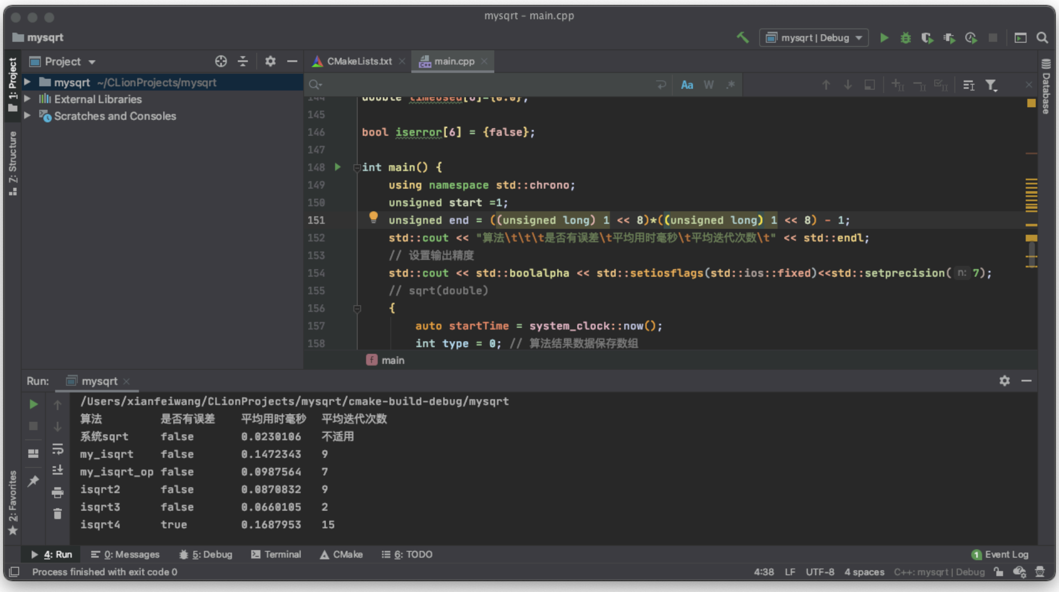
（2）编写 unsigned my_isqrt_op(unsigned c)算法。其中二叉树查找 s 部分如图所示



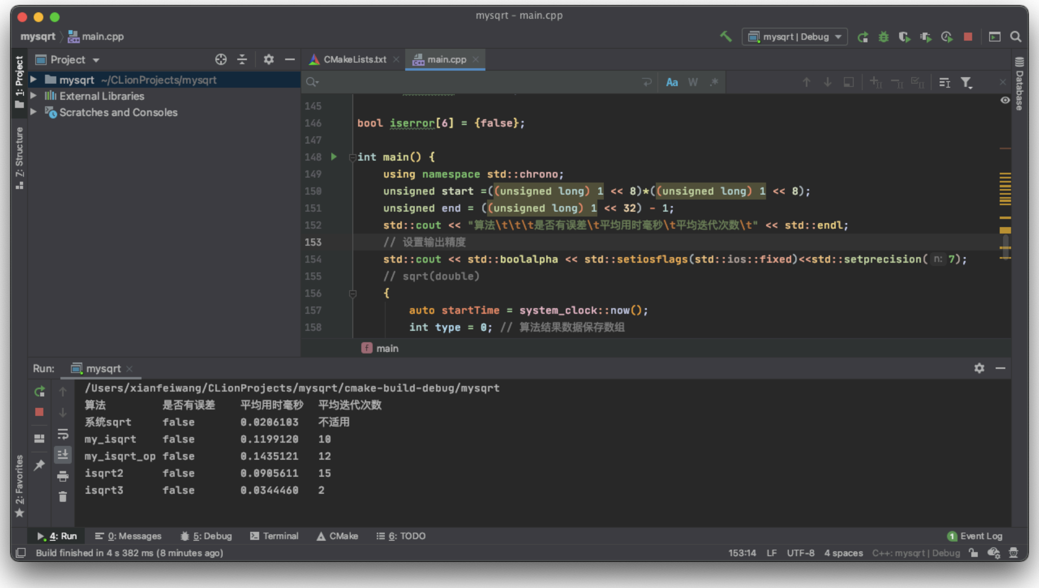
不过不知道如何具体实现二叉树，感觉应该是按照从 8->1 依次查找。所以就按照了从 8 到 1 指数依次进行查找。（完整运行代码见文末附录 2），对于

$[1, (2^8+1)^2)$ 区间进行统计，所提到的测试方法及另外几种算法的时间进行比较，结果如下：

算法	是否有误差	平均用时毫秒	平均迭代次数
系统 sqrt	false	0.0196078	不适用
my_isqrt	false	0.1529412	9
my_isqrt_op	false	0.1019608	11
isqrt2	false	0.0274510	3
isqrt3	false	0.0274510	1
isqrt4	true	0.0980392	15



对于 $[(2^8+1)^2, 2^{32})$ 区间进行统计，所提到的测试方法及另外几种算法的时间进行比较，结果如下：



五、实验心得

对于 $[1,\ (2^8+1)^2)$ 区间而言，my_isqrt_op 迭代时间和迭代次数均优于 my_isqrt；而对于 $[(2^8+1)^2,\ 2^{32})$ 区间而言，y_isqrt_op 迭代时间和迭代次数均比不上 my_isqrt。

六、代码附录

（1）代码：

```
#include <iostream>
#include <cmath>
#include <chrono>
#include <iomanip>

// 用于保存迭代次数
unsigned long long num [6]={0};

unsigned my_isqrt(unsigned c) {
    // 二分法查找初值 2^(s-1)
    // 其中 2^(s-1) < 根号 c 向下取证 <= 2^s
    unsigned l = 0, r = 32, s = 0;
    while (l <= r) {
        //计算迭代次数
        num[1]++;
        unsigned mid = l + (r - l) / 2;
        if ((unsigned long) 1 << (mid) * (unsigned long) 1
<< (mid) < c) {
```

```
        if ((unsigned long) 1 << (mid + 1) * (unsigned
long) 1 << (mid + 1) > c)
            s = mid + 1;
        l = mid + 1;
    } else {
        if ((unsigned long) 1 << (mid - 1) * (unsigned
long) 1 << (mid - 1) < c)
            s = mid;
        r = mid - 1;
    }
  }
  // 此处初值 x0 应为 2^s
  // 进行牛顿迭代
  double C = c, x0 = (unsigned) 1 << s;
  while (true) {
    //计算迭代次数
    num[1]++;
    double xi = 0.5 * (x0 + C / x0);
    if (fabs(x0 - xi) < 1e-7) break; // 精度控制
    x0 = xi;
  }
  return int(x0);
}


int isqrt2(unsigned x) {
  unsigned a = 1; //上界
  unsigned b = (x >> 5) + 8; //下界
  if (b > 65535) b = 65535; //a <= sqrt(x) <= b
  do {
    //计算迭代次数
    num[3]++;
    int m = (a + b) >> 1;
    if (m * m > x) b = m - 1; else a = m + 1;
  }
  while (b >= a);
  return a - 1;
}

int isqrt3(unsigned x) {
  if (x <= 1) return x;
  int x1 = x - 1;
  int s = 1;
  if (x1 > 65535) {
```

```
        s += 8;
        x1 >>= 16;
    }
    if (x1 > 255) {
        s += 4;
        x1 >>= 8;
    }
    if (x1 > 15) {
        s += 2;
        x1 >>= 4;
    }
    if (x1 > 3) { s += 1; }
    int x0 = 1 << s;
    x1 = (x0 + (x >> s)) >> 1;
    while (x1 < x0) {
        x0 = x1;
        x1 = (x0 + x / x0) >> 1;
        //计算迭代次数
        num[4]++;
    }
    return x0;
}

unsigned int isqrt4(unsigned long M)
{
    unsigned int N, i;
    unsigned long tmp, ttp;
    if (M == 0)
        return 0;
    N = 0;
    tmp = (M >> 30);
    M <<= 2;
    if (tmp > 1)
    {
        N ++;
        tmp -= N;
    }
    for (i=15; i>0; i--)
    {
        //计算迭代次数
        num[5]++;
        N <<= 1;
        tmp <<= 2;
        tmp += (M >> 30);
```

```
        ttp = N;
        ttp = (ttp<<1)+1;
        M <<= 2;
        if (tmp >= ttp)
        {
            tmp -= ttp;
            N ++;
        }
    }
    return N;
}

double timeused[6]={0.0};

bool iserror[6] = {false};

int main() {
    using namespace std::chrono;
    unsigned start =1;
    unsigned end = ((unsigned long) 1 << 32) - 1;
    std::cout << "算法\t\t\t是否有误差\t平均用时毫秒\t平均迭代次数\t" << std::endl;
    // 设置输出精度
    std::cout << std::boolalpha << std::setiosflags(std::ios::fixed)<<std::setprecision(7);
    // sqrt(double)
    {
        auto startTime = system_clock::now();
        int type = 0; // 算法结果数据保存数组
        for (unsigned i = start; i <= end; i++) {
            // 判断是否存在误差
            if ((unsigned)sqrt(i) != (unsigned) sqrt(i))iserror[type] = true;
        }
        auto endTime = system_clock::now();
        auto duration = duration_cast<microseconds>(endTime - startTime);
        timeused[type] = double(duration.count()) / (end - start + 1); // 毫秒
        std::cout << "系统sqrt\t\t" << iserror[type] << "\t\t" << timeused[type] << "\t" << "不适用" << std::endl;
    }

    // my_isqrt
```

```
    {
        auto startTime = system_clock::now();
        int type = 1; // 算法结果数据保存数组
        for (unsigned i = start; i <= end; i++) {
            // 判断是否存在误差
            if (my_isqrt(i) != (unsigned)
sqrt(i))iserror[type] = true;
        }
        auto endTime = system_clock::now();
        auto duration =
duration_cast<microseconds>(endTime - startTime);
        timeused[type] = double(duration.count()) / (end -
start + 1); // 毫秒
        std::cout << "my_isqrt\t" << iserror[type] <<
"\t\t" << timeused[type] << "\t" << num[type]/(end -
start + 1) << std::endl;
    }

    // isqrt2
    {
        auto startTime = system_clock::now();
        int type = 3; // 算法结果数据保存数组
        for (unsigned i = start; i <= end; i++) {
            // 判断是否存在误差
            if (isqrt2(i) != (unsigned)
sqrt(i))iserror[type] = true;
        }
        auto endTime = system_clock::now();
        auto duration =
duration_cast<microseconds>(endTime - startTime);
        timeused[type] = double(duration.count()) / (end -
start + 1); // 毫秒
        std::cout << "isqrt2\t\t" << iserror[type] <<
"\t\t" << timeused[type] << "\t" << num[type]/(end -
start + 1) << std::endl;
    }

    // isqrt3
    {
        auto startTime = system_clock::now();
        int type = 4; // 算法结果数据保存数组
        for (unsigned i = start; i <= end; i++) {
            // 判断是否存在误差
```

```
            if (isqrt3(i) != (unsigned)
sqrt(i))iserror[type] = true;
        }
        auto endTime = system_clock::now();
        auto duration =
duration_cast<microseconds>(endTime - startTime);
        timeused[type] = double(duration.count()) / (end -
start + 1); // 毫秒
        std::cout << "isqrt3\t\t" << iserror[type] <<
"\t\t" << timeused[type] << "\t" << num[type]/(end -
start + 1) << std::endl;
    }

    // isqrt4
    {
        auto startTime = system_clock::now();
        int type = 5; // 算法结果数据保存数组
        for (unsigned i = start; i <= end; i++) {
            // 判断是否存在误差
            if (isqrt4(i) != (unsigned)
sqrt(i))iserror[type] = true;
        }
        auto endTime = system_clock::now();
        auto duration =
duration_cast<microseconds>(endTime - startTime);
        timeused[type] = double(duration.count()) / (end -
start + 1); // 毫秒
        std::cout << "isqrt4\t\t" << iserror[type] <<
"\t\t" << timeused[type] << "\t" << num[type]/(end -
start + 1) << std::endl;
    }

    return 0;
}
```

（2）代码

```
#include <iostream>
#include <cmath>
#include <chrono>
#include <iomanip>

// 用于保存迭代次数
unsigned long long num [6]={0};
```

```
unsigned my_isqrt(unsigned c) {
    // 二分法查找初值 2^(s-1)
    // 其中 2^(s-1) < 根号 c 向下取证 <= 2^s
    unsigned l = 0, r = 32, s = 0;
    while (l <= r) {
        //计算迭代次数
        num[1]++;
        unsigned mid = l + (r - l) / 2;
        if ((unsigned long) 1 << (mid) * (unsigned long) 1 << (mid)
< c) {
            if ((unsigned long) 1 << (mid + 1) * (unsigned long) 1
<< (mid + 1) > c)
                s = mid + 1;
            l = mid + 1;
        } else {
            if ((unsigned long) 1 << (mid - 1) * (unsigned long) 1
<< (mid - 1) < c)
                s = mid;
            r = mid - 1;
        }
    }
    // 此处初值 x0 应为 2^s
    // 进行牛顿迭代
    double C = c, x0 = (unsigned) 1 << s;
    while (true) {
        //计算迭代次数
        num[1]++;
        double xi = 0.5 * (x0 + C / x0);
        if (fabs(x0 - xi) < 1e-7) break; // 精度控制
        x0 = xi;
    }
    return int(x0);
}

unsigned my_isqrt_op(unsigned c) {
    // 二叉查找树查找初值 2^(s-1)
    // 其中 2^(s-1) < 根号 c 向下取证 <= 2^s
    // 不知道咋写查找树了  但是感觉就是 8->7->6->...->1 这样查找
    unsigned s = 8;
    while (s>0) {
        //计算迭代次数
        num[2]++;
        if ((unsigned long) 1 << s * (unsigned long) 1 << s < c) {
```

```
        if ((unsigned long) 1 << (s- 1) * (unsigned long) 1 <<
(s - 1) < c) {
            break;
        }
    }
    s--;
}
// 此处初值 x0 应为 2^s
// 进行牛顿迭代
double C = c, x0 = (unsigned) 1 << s;
while (true) {
    //计算迭代次数
    num[2]++;
    double xi = 0.5 * (x0 + C / x0);
    if (fabs(x0 - xi) < 1e-7) break; // 精度控制
    x0 = xi;
}
return int(x0);
}


int isqrt2(unsigned x) {
    unsigned a = 1; //上界
    unsigned b = (x >> 5) + 8; //下界
    if (b > 65535) b = 65535; //a <= sqrt(x) <= b
    do {
        //计算迭代次数
        num[3]++;
        int m = (a + b) >> 1;
        if (m * m > x) b = m - 1; else a = m + 1;
    }
    while (b >= a);
    return a - 1;
}

int isqrt3(unsigned x) {
    if (x <= 1) return x;
    int x1 = x - 1;
    int s = 1;
    if (x1 > 65535) {
        s += 8;
        x1 >>= 16;
    }
    if (x1 > 255) {
```

```
        s += 4;
        x1 >>= 8;
    }
    if (x1 > 15) {
        s += 2;
        x1 >>= 4;
    }
    if (x1 > 3) { s += 1; }
    int x0 = 1 << s;
    x1 = (x0 + (x >> s)) >> 1;
    while (x1 < x0) {
        x0 = x1;
        x1 = (x0 + x / x0) >> 1;
        //计算迭代次数
        num[4]++;
    }
    return x0;
}

unsigned int isqrt4(unsigned long M)
{
    unsigned int N, i;
    unsigned long tmp, ttp;
    if (M == 0)
        return 0;
    N = 0;
    tmp = (M >> 30);
    M <<= 2;
    if (tmp > 1)
    {
        N ++;
        tmp -= N;
    }
    for (i=15; i>0; i--)
    {
        //计算迭代次数
        num[5]++;
        N <<= 1;
        tmp <<= 2;
        tmp += (M >> 30);
        ttp = N;
        ttp = (ttp<<1)+1;
        M <<= 2;
        if (tmp >= ttp)
```

```
        {
            tmp -= ttp;
            N ++;
        }
    }
    return N;
}

double timeused[6]={0.0};

bool iserror[6] = {false};

int main() {
    using namespace std::chrono;
    unsigned start =1;
    unsigned end = ((unsigned long) 1 << 8)* ((unsigned long) 1 <<
8) - 1;
    std::cout << "算法\t\t\t 是否有误差\t 平均用时毫秒\t 平均迭代次数\t"
<< std::endl;
    // 设置输出精度
    std::cout << std::boolalpha <<
std::setiosflags(std::ios::fixed)<<std::setprecision(7);
    // sqrt(double)
    {
        auto startTime = system_clock::now();
        int type = 0; // 算法结果数据保存数组
        for (unsigned i = start; i <= end; i++) {
            // 判断是否存在误差
            if ((unsigned)sqrt(i) != (unsigned)
sqrt(i))iserror[type] = true;
        }
        auto endTime = system_clock::now();
        auto duration = duration_cast<microseconds>(endTime -
startTime);
        timeused[type] = double(duration.count()) / (end - start +
1); // 毫秒
        std::cout << "系统 sqrt\t\t" << iserror[type] << "\t\t" <<
timeused[type] << "\t" << "不适用" << std::endl;
    }

    // my_isqrt
    {
        auto startTime = system_clock::now();
        int type = 1; // 算法结果数据保存数组
```

```
        for (unsigned i = start; i <= end; i++) {
            // 判断是否存在误差
            if (my_isqrt(i) != (unsigned) sqrt(i))iserror[type] =
true;
        }
        auto endTime = system_clock::now();
        auto duration = duration_cast<microseconds>(endTime -
startTime);
        timeused[type] = double(duration.count()) / (end - start +
1); // 毫秒
        std::cout << "my_isqrt\t" << iserror[type] << "\t\t" <<
timeused[type] << "\t" << num[type]/(end - start + 1) <<
std::endl;
    }

    // my_isqrt_op
    {
        auto startTime = system_clock::now();
        int type = 2; // 算法结果数据保存数组
        for (unsigned i = start; i <= end; i++) {
            // 判断是否存在误差
            if (my_isqrt_op(i) != (unsigned) sqrt(i))iserror[type]
= true;
        }
        auto endTime = system_clock::now();
        auto duration = duration_cast<microseconds>(endTime -
startTime);
        timeused[type] = double(duration.count()) / (end - start +
1); // 毫秒
        std::cout << "my_isqrt_op\t" << iserror[type] << "\t\t" <<
timeused[type] << "\t" << num[type]/(end - start + 1) <<
std::endl;
    }

    // isqrt2
    {
        auto startTime = system_clock::now();
        int type = 3; // 算法结果数据保存数组
        for (unsigned i = start; i <= end; i++) {
            // 判断是否存在误差
            if (isqrt2(i) != (unsigned) sqrt(i))iserror[type] =
true;
        }
        auto endTime = system_clock::now();
```

```
        auto duration = duration_cast<microseconds>(endTime -
startTime);
        timeused[type] = double(duration.count()) / (end - start +
1); // 毫秒
        std::cout << "isqrt2\t\t" << iserror[type] << "\t\t" <<
timeused[type] << "\t" << num[type]/(end - start + 1) <<
std::endl;
    }

    // isqrt3
    {
        auto startTime = system_clock::now();
        int type = 4; // 算法结果数据保存数组
        for (unsigned i = start; i <= end; i++) {
            // 判断是否存在误差
            if (isqrt3(i) != (unsigned) sqrt(i))iserror[type] =
true;
        }
        auto endTime = system_clock::now();
        auto duration = duration_cast<microseconds>(endTime -
startTime);
        timeused[type] = double(duration.count()) / (end - start +
1); // 毫秒
        std::cout << "isqrt3\t\t" << iserror[type] << "\t\t" <<
timeused[type] << "\t" << num[type]/(end - start + 1) <<
std::endl;
    }

    // isqrt4
    {
        auto startTime = system_clock::now();
        int type = 5; // 算法结果数据保存数组
        for (unsigned i = start; i <= end; i++) {
            // 判断是否存在误差
            if (isqrt4(i) != (unsigned) sqrt(i))iserror[type] =
true;
        }
        auto endTime = system_clock::now();
        auto duration = duration_cast<microseconds>(endTime -
startTime);
        timeused[type] = double(duration.count()) / (end - start +
1); // 毫秒
```

```
        std::cout << "isqrt4\t\t" << iserror[type] << "\t\t" <<
timeused[type] << "\t" << num[type]/(end - start + 1) <<
std::endl;
    }

    return 0;
}
```