实验报告

实验名称（测量 FFT 程序执行时间）

班级智能 1602 学号 201608010708 姓名田宗杭

实验目标

测量 FFT 程序运行时间，确定其时间复杂度。

实验要求

* 采用 C/C++编写程序
* 根据自己的机器配置选择合适的输入数据大小 n，至少要测试多个不同的 n（参见思考题）
* 对于相同的 n，建议重复测量 30 次取平均值作为测量结果（参见思考题）
* 对测量结果进行分析，确定 FFT 程序的时间复杂度
* 回答思考题，答案加入到实验报告叙述中合适位置

思考题

1. 分析 FFT 程序的时间复杂度，得到执行时间相对于数据规模 n 的具体公式
2. 根据上一点中的分析，至少要测试多少不同的 n 来确定执行时间公式中的未知数？
3. 重复 30 次测量然后取平均有什么统计学的依据？

实验内容

FFT 算法代码

FFT 算法可以参考[这里]
(https://en.wikipedia.org/wiki/Cooley%E2%80%93Tukey_FFT_algorithm)。
```c++
/* fft.cpp
 *
 * This is a KISS implementation of
 * the Cooley-Tukey recursive FFT algorithm.
 * This works, and is visibly clear about what is happening where.
 *
 * To compile this with the GNU/GCC compiler:
 * g++ -o fft fft.cpp -lm
 *
 * To run the compiled version from a *nix command line:
 * ./fft
 *
 */
#include <complex>
#include <cstdio>
```

```
#define M_PI 3.14159265358979323846 // Pi constant with double precision

using namespace std;

// separate even/odd elements to lower/upper halves of array respectively.
// Due to Butterfly combinations, this turns out to be the simplest way
// to get the job done without clobbering the wrong elements.
void separate (complex<double>* a, int n) {
    complex<double>* b = new complex<double>[n/2];  // get temp heap storage
    for(int i=0; i<n/2; i++)    // copy all odd elements to heap storage
        b[i] = a[i*2+1];
    for(int i=0; i<n/2; i++)    // copy all even elements to lower-half of a[]
        a[i] = a[i*2];
    for(int i=0; i<n/2; i++)    // copy all odd (from heap) to upper-half of a[]
        a[i+n/2] = b[i];
    delete[] b;                 // delete heap storage
}

// N must be a power-of-2, or bad things will happen.
// Currently no check for this condition.
//
// N input samples in X[] are FFT'd and results left in X[].
// Because of Nyquist theorem, N samples means
// only first N/2 FFT results in X[] are the answer.
// (upper half of X[] is a reflection with no new information).
void fft2 (complex<double>* X, int N) {
    if(N < 2) {
        // bottom of recursion.
        // Do nothing here, because already X[0] = x[0]
    } else {
        separate(X,N);      // all evens to lower half, all odds to upper half
        fft2(X,     N/2);   // recurse even items
        fft2(X+N/2, N/2);   // recurse odd  items
        // combine results of two half recursions
        for(int k=0; k<N/2; k++) {
            complex<double> e = X[k    ];   // even
            complex<double> o = X[k+N/2];   // odd
                            // w is the "twiddle-factor"
            complex<double> w = exp( complex<double>(0,-2.*M_PI*k/N) );
            X[k    ] = e + w * o;
            X[k+N/2] = e - w * o;
        }
    }
}

// simple test program
```

```
int main () {
    const int nSamples = 64;
    double nSeconds = 1.0;                      // total time for sampling
    double sampleRate = nSamples / nSeconds;    // n Hz = n / second
    double freqResolution = sampleRate / nSamples; // freq step in FFT result
    complex<double> x[nSamples];                // storage for sample data
    complex<double> X[nSamples];                // storage for FFT answer
    const int nFreqs = 5;
    double freq[nFreqs] = { 2, 5, 11, 17, 29 }; // known freqs for testing

    // generate samples for testing
    for(int i=0; i<nSamples; i++) {
        x[i] = complex<double>(0.,0.);
        // sum several known sinusoids into x[]
        for(int j=0; j<nFreqs; j++)
            x[i] += sin( 2*M_PI*freq[j]*i/nSamples );
        X[i] = x[i];        // copy into X[] for FFT work & result
    }
    // compute fft for this data
    fft2(X,nSamples);

    printf("  n\tx[]\tX[]\tf\n");       // header line
    // loop to print values
    for(int i=0; i<nSamples; i++) {
        printf("% 3d\t%+.3f\t%+.3f\t%g\n",
            i, x[i].real(), abs(X[i]), i*freqResolution );
    }
}

// eof
```

FFT 程序时间复杂度分析

通过分析 FFT 算法代码，可以得到该 FFT 算法的时间复杂度具体公式为：

将 x(n)分解为偶数与奇数的两个序列之和，即
x1(n)和 x2(n)的长度都是 N / 2，x1(n)是偶数序列，x2(n)是奇数序列，则
其中 X1(k)和 X2(k)分别为 x1(n)和 x2(n)的 N / 2 点 DFT。由于 X1(k)和 X2(k)均以 N / 2 为周期，且 WN k+N/2=-WN k
依此类推，经过 m-1 次分解，最后将 N 点 DFT 分解为 N / 2 个两点 DFT。

a*n*logn+(b/3)*n+2^(1/2)*c*logn+d

其中*n*为数据大小，未知数有：

1. *a*
2. *b*
3. *c*
4. *d*

测试

测试平台

在如下机器上进行了测试：

部件	配置	备注
CPU	core i7-6700U	
内存	DDR3 8GB	
操作系统	Ubuntu 18.04 LTS	中文版

测试记录

FFT 程序的测试输入文件请见[这里](./test.input)。

```
double freq[nFreqs] = { 2, 5, 11, 17, 29 }; // known freqs for testing
```

FFT 程序运行过程的截图如下：

```
tian@ubuntu:~$ cd Desktop
tian@ubuntu:~/Desktop$ g++ -o fft fft.cpp -lm
tian@ubuntu:~/Desktop$ ./fft
root@ubuntu:~/Desktop# perf stat ./fft
```

FFT 程序的输出

![图 1 测试执行时间](./perf_ls.png)

N=1

```
  n      x[]      X[]      f
  0     +0.000   +0.000   0
it took 0.000006 seconds
```

N=2

```
  n      x[]      X[]      f
  0     +0.000   +0.000   0
  1     +0.000   +0.000   1
it took 0.000020 seconds
```

N=4

```
  n      x[]      X[]      f
  0     +0.000   +0.000   0
  1     +2.000   +4.000   1
  2     +0.000   +0.000   2
  3     -2.000   +4.000   3
it took 0.000029 seconds
```

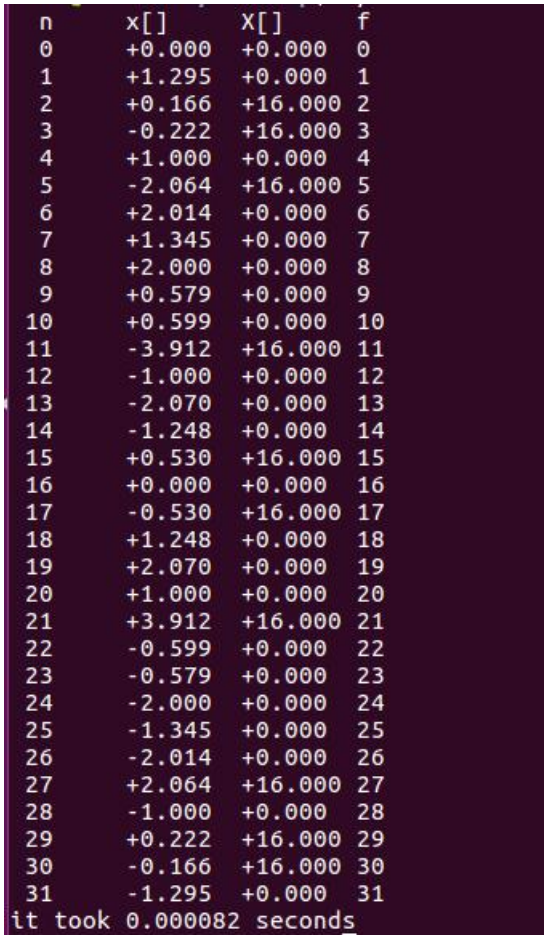
N=8

```
  n      x[]      X[]      f
  0      +0.000   +0.000   0
  1      +1.000   +4.000   1
  2      +2.000   +4.000   2
  3      -1.000   +4.000   3
  4      +0.000   +0.000   4
  5      +1.000   +4.000   5
  6      -2.000   +4.000   6
  7      -1.000   +4.000   7
it took 0.000052 seconds
```

N=16

```
  n      x[]      X[]      f
  0      +0.000   +0.000   0
  1      +0.166   +8.000   1
  2      +1.000   +8.000   2
  3      +2.014   +8.000   3
  4      +2.000   +0.000   4
  5      +0.599   +0.000   5
  6      -1.000   +0.000   6
  7      -1.248   +0.000   7
  8      +0.000   +0.000   8
  9      +1.248   +0.000   9
 10      +1.000   +0.000   10
 11      -0.599   +0.000   11
 12      -2.000   +0.000   12
 13      -2.014   +8.000   13
 14      -1.000   +8.000   14
 15      -0.166   +8.000   15
it took 0.000048 seconds
```

N=32



```
 n      x[]     X[]     f
 0     +0.000  +0.000  0
 1     +1.295  +0.000  1
 2     +0.166  +16.000 2
 3     -0.222  +16.000 3
 4     +1.000  +0.000  4
 5     -2.064  +16.000 5
 6     +2.014  +0.000  6
 7     +1.345  +0.000  7
 8     +2.000  +0.000  8
 9     +0.579  +0.000  9
10     +0.599  +0.000  10
11     -3.912  +16.000 11
12     -1.000  +0.000  12
13     -2.070  +0.000  13
14     -1.248  +0.000  14
15     +0.530  +16.000 15
16     +0.000  +0.000  16
17     -0.530  +16.000 17
18     +1.248  +0.000  18
19     +2.070  +0.000  19
20     +1.000  +0.000  20
21     +3.912  +16.000 21
22     -0.599  +0.000  22
23     -0.579  +0.000  23
24     -2.000  +0.000  24
25     -1.345  +0.000  25
26     -2.014  +0.000  26
27     +2.064  +16.000 27
28     -1.000  +0.000  28
29     +0.222  +16.000 29
30     -0.166  +16.000 30
31     -1.295  +0.000  31
it took 0.000082 seconds
```

N=64

n	x[]	X[]	f
0	+0.000	+0.000	0
1	+2.834	+0.000	1
2	+1.295	+32.000	2
3	+1.269	+0.000	3
4	+0.166	+0.000	4
5	+2.570	+32.000	5
6	-0.222	+0.000	6
7	+1.756	+0.000	7
8	+1.000	+0.000	8
9	+0.839	+0.000	9
10	-2.064	+0.000	10
11	-1.145	+32.000	11
12	+2.014	+0.000	12
13	+1.305	+0.000	13
14	+1.345	+0.000	14
15	-0.449	+0.000	15
16	+2.000	+0.000	16
17	-0.840	+32.000	17
18	+0.579	+0.000	18
19	+0.194	+0.000	19
20	+0.599	+0.000	20
21	-2.808	+0.000	21
22	-3.912	+0.000	22
23	-1.122	+0.000	23
24	-1.000	+0.000	24
25	-0.205	+0.000	25
26	-2.070	+0.000	26
27	+0.907	+0.000	27
28	-1.248	+0.000	28
29	+0.158	+32.000	29
30	+0.530	+0.000	30
31	+2.444	+0.000	31
32	+0.000	+0.000	32
33	-2.444	+0.000	33
34	-0.530	+0.000	34
35	-0.158	+32.000	35
36	+1.248	+0.000	36
37	-0.907	+0.000	37
38	+2.070	+0.000	38
39	+0.205	+0.000	39
40	+1.000	+0.000	40
41	+1.122	+0.000	41
42	+3.912	+0.000	42
43	+2.808	+0.000	43
44	-0.599	+0.000	44
45	-0.194	+0.000	45

```
 46      -0.579  +0.000  46
 47      +0.840  +32.000 47
 48      -2.000  +0.000  48
 49      +0.449  +0.000  49
 50      -1.345  +0.000  50
 51      -1.305  +0.000  51
 52      -2.014  +0.000  52
 53      +1.145  +32.000 53
 54      +2.064  +0.000  54
 55      -0.839  +0.000  55
 56      -1.000  +0.000  56
 57      -1.756  +0.000  57
 58      +0.222  +0.000  58
 59      -2.570  +32.000 59
 60      -0.166  +0.000  60
 61      -1.269  +0.000  61
 62      -1.295  +32.000 62
 63      -2.834  +0.000  63
it took 0.000122 seconds
```

```
 Performance counter stats for './fft':

          1.884555      task-clock (msec)         #    0.821 CPUs utilized
                 0      context-switches          #    0.000 K/sec
                 0      cpu-migrations            #    0.000 K/sec
                77      page-faults               #    0.041 M/sec
   <not supported>      cycles
   <not supported>      instructions
   <not supported>      branches
   <not supported>      branch-misses

       0.002296505 seconds time elapsed
```

上图为当 n=64 时，perf 工具的测量结果显示，由于为运行在虚拟机平台上，所以 perf 工具的测量并不准确，并且后几项缺失。

2. 根据上一点中的分析，至少要测试多少不同的 n 来确定执行时间公式中的未知数？

设立了四种不同的未知数，所以需要列四种不同的方程，也就是需要四种不同的 n

3. 重复 30 次测量然后取平均有什么统计学的依据？

因为测量总是或多或少会存在些误差，不论是人工测量还是机器测量，人工测量会存在诸如读数等方面的误差，机器测量也会存在机械故障或者耗损等方面导致的误差， 所以为了减少因为误差对数据结果造成的影响，采用多次重复测量取平均值来表示测量结果

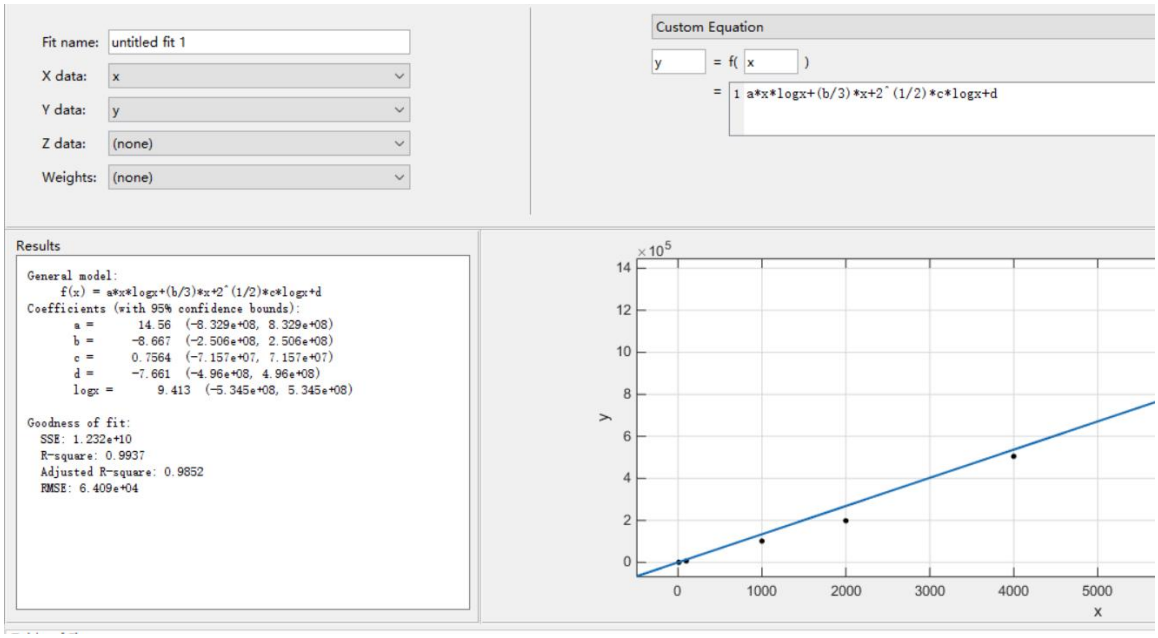
分析和结论

```
>>  x=[1,2,4,8,16,32,64];
>> y=[0.000006,0.000020,0.000029,0.000052,0.000048,0.000082,0.000122]

y =

   1.0e-03 *

    0.0060    0.0200    0.0290    0.0520    0.0480    0.0820    0.1220
```



利用 matlab 的拟合工具可以看到

a= 14.56 * 1.0e-03

b=-8.667* 1.0e-03

c=0.7564* 1.0e-03

d=-7.661* 1.0e-03

从测试记录来看，FFT 程序的执行时间随数据规模增大而增大，其时间复杂度为 O(（14.56* *n*logn+(-8.667/3)*n+2^(1/2)*0.7564*logn-7.661）* 1.0e-03)