实验报告

实验名称（测量 FFT 程序执行时间）
班级 物联 1601　　学号 201508010308　　姓名 陈建宇

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

```
#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 programint 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：

$$a*n*logn+\frac{b}{3}*n+\sqrt{2}*c*logn+d$$

其中 n 为数据大小，未知数有：

a b c 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	I5-4200H	
内存	4GB DDR4	
操作系统	Ubuntu 16.04 LTS	中文版

测试记录

截图太多不放了。

FFT 程序的输出——（seconds time elapsed）

16 条数据时 运行时间约为 0.0005 秒

32 条数据时 运行时间约为 0.001 秒

64 条数据时 运行时间约为 0.002 秒

（为减少误差皆为循环 30 次数据）

思考题解答

1.测试次数

在这里我们的复杂度具体公式设立了 4 个未知数，为了求解 4 个未知数的值，我们在这里需要至少 4 个 n 才能求解

2.统计学依据

在这里我们用到了累积法测量的思想： 因为利用 FFT 提高代码效率，使得单次运行时间很短，所以做 30 次取平均以减小误差。

分析和结论

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(nlogn)。单一运行的时候与多次测量取平均值所得数据有较大不同，因此实验过程中需要注意多次测

量取平均值来减小误差，令结果更加精准。