实验报告

一、实验名称（测量 FFT 程序执行时间）

智能 1602 201608010627 任小禹

二、实验目标

测量 FFT 程序运行时间，确定其时间复杂度。

三、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

四、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？

五、实验内容

FFT 算法代码

//fft.cpp（数据大小 64，循环 30 次）

```
/* 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>
#include<ctime>
```

```
#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 () {
	clock_t start,end;
	double time;
	start=clock();
	int number=30;
	while(number--){
	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 );
	}
	}
	end=clock();
	time=double(end-start)/CLOCKS_PER_SEC;
	printf("  此程序运行的时间为:%f s",time/30 );
}

// eof
```

FFT 程序时间复杂度分析

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

其中 n 为数据大小，未知数有：

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

六、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 i5-7200U	
内存	DDR3 4GB	
操作系统	Ubuntu 14.04 LTS	虚拟机

测试记录

数据大小为 16：

```
pig@ubuntu:~/Desktop$ g++ fft.cpp -o fft
pig@ubuntu:~/Desktop$ ./fft
  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
 runningname:0.000263 s
```

循环 30 次

```
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
 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
runningname:0.000135 s
```

数据大小为 32：

```
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
runningname:0.000302 s
```

循环 30 次

```
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
runningname:0.000277 s
```

数据大小为 64

```
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
runningname:0.003539 s
```

循环 30 次

```
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
runningname:0.000826 s
```

七、思考题

1. 因为时间复杂度地公式中有四个未知数，所以至少需要 4 个不同的 n。
2. FFT 效率很高，所以单次执行很短，所以多次循环重复取平均值可以减少误差，使结果更准确。

八、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)。