Data Structure Assignment #1

Performance Analysis and Measurement

1. 시간 복잡도
   a. Sum 함수의 명령 수행 횟수는 총 몇 번인가?

```
float Sum(float* a, const int n)
{
	float s = 0;
	for (int i = 0; i < n; i++)
		s += a[i];
	return s;
}
```

   b. Rsum 함수의 명령 수행 횟수는 총 몇 번인가?

```
float Rsum(float* a, const int n)
{
	if (n <= 0) return 0;
	else return (Rsum(a, n-1) + a[n-1]);
}
```

   c. Add 의 명령 수행 횟수는 총 몇 번인가?

```
void Add(int** a, int** b, int** c, int m, int n)
{
	for (int i = 0; i < m; i++)
		for (int j = 0; j < n; j++)
			c[i][j] = a[i][j] + b[i][j];
}
```

   d. Fibonacci 함수의 명령 수행 횟수는 총 몇 번인가?

```
void Fibonacci(int n)
{	// Compute the Fibonacci number Fn
	if (n <= 1) cout << n << endl;
	else
	{	// Compute Fn
		int fn;
		int fnm2 = 0, fnm1 = 1;
		for (int i = 2; i <= n; i++)
		{
			fn = fnm1 + fnm2;
			fnm2 = fnm1;
			fnm1 = fn;
		}	// End of for
		cout << fn << endl;
	}	// End of else
}
```

2. 점근적 표기법 (O, Ω, Θ)
   a. Big-oh 표기법을 정의하라.
   b. Omega 표기법을 정의하라.
   c. Theta 표기법을 정의하라.
   d. Permutations 함수의 시간 복잡도는?

```
void Permutations(char* a, const int k, const int m)
{
	// Generate all the permutations of a[k], ..., a[m].
	if (k == m)   // Output permutation
	{
		for (int i = 0; i <= m; i++)
			cout << a[i] << " ";
		cout << endl;
	}
	else // a[k:m] has more than one permutation. Generate these recursively.
	{
		for (int i = k; i <= m; i++)
		{
			swap(a[k], a[i]);
			Permutations(a, k + 1, m);
			swap(a[k], a[i]);
		}
	}
}
```

   e. BinarySearch 함수의 시간 복잡도는?

```
int BinarySearch(int* a, const int x, const int n)
{	// Search the sorted array a[0], ... , a[n-1] for x.
	int left = 0, right = n - 1;
	while (left <= right)
	{	// There are more elements
		int middle = (left + right) / 2;
		if (x < a[middle]) right = middle - 1;
		else if (x > a[middle]) left = middle + 1;
		else return middle;
	} // End of while
	return -1; // Not found
}
```

f. Magic 함수의 시간 복잡도는?
(여기서 말하는 Magic Square 란 모든 행, 열, 대각선의 합이 같은 $n \times n$ 행렬을 말함)
(H. Coxeter 는 n이 홀수일 때 Magic Square 를 만드는 간단한 방법을 제시함)

```
void Magic(const int n)
{       // Create a magic square of size n, n is odd.
        const int MaxSize = 51;     // Maximum square size
        int square[MaxSize][MaxSize], k, l;

        // Check correctness of n
        if ((n > MaxSize) || (n < 1))
                throw "Error! n out of range";
        else if (!(n % 2)) throw "Error! n is even";

        // n is odd. Coxeter's rule can be used
        for (int i = 0; i < n; i++) // Initialize square to 0
                fill(square[i], square[i] + n, 0); // STL algorithm
        square[0][(n - 1) / 2] = 1; // Middle of first row

        // i and j are current position
        int key = 2, i = 0, j = (n - 1) / 2;
        while (key <= n * n)
        {
                // Move up and left
                if (i - 1 < 0) k = n - 1;
                else k = i = 1;
                if (j - 1 < 0) l = n - 1;
                else l = j - 1;
                if (square[k][l]) i = (i + 1) % n; // Square occupied, mvoe down
                else
                {       // square[k][l] is unoccupied
                        i = k;
                        j = l;
                }
                square[i][j] = key;
                key++;
        }       // End of while

        // Output the magic square
        cout << "Magic square of size " << n << endl;
        for (i = 0; i < n; i++)
        {
                copy(square[i], square[i] + n, ostream_iterator<int>(cout, " "));
                cout << endl;
        }
}
```

3. 추가 문제

a. D 함수의 명령 수행 횟수는 총 몇 번인가?

```
void D(int* x, int n)
{
	int i = 1;
	do
	{
		x[i] += 2;
		i += 2;
	} while (i <= n);
	i = 1;
	while (i <= (n / 2))
	{
		x[i] += x[i + 1];
		i++;
	}
}
```

b. Transpose 함수의 명령 수행 횟수는 총 몇 번인가?

```
void Transpose(int** a, int n)
{
	for (int i = 0; i < n - 1; i++)
		for (int j = i + 1; j < n; j++)
			swap(a[i][j], a[j][i]);
}
```

c. Multiply 함수의 명령 수행 횟수는 총 몇 번인가?

```
void Multiply(int** a, int** b, int** c, int n)
{
	for (int i = 0; i < n; i++)
		for (int j = 0; j < n; j++)
		{
			c[i][j] = 0;
			for (int k = 0; k < n; k++)
				c[i][j] += a[i][k] * b[k][j];
		}
}
```

d. SelectionSort 함수의 시간 복잡도는?

```
void SelectionSort(int* a, const int n)
{	// Sort the n integers a[0] to a[n-1] into nondecreasing order.
	for (int i = 0; i < n; i++)
	{
		int j = i;
		// Find smallest integer in a[i] to a[n - 1]
		for (int k = i + 1; k < n; k++)
			if (a[k] < a[j])
				j = k;
		swap(a[i], a[j]);
	}
}
```

e. 복소수 행렬 X는 두 행렬 (A, B) 쌍으로 구성되어 있다. 이 때 A와 B는 실수를 포함한다. 두 복소수 행렬 (A, B), (C, D)의 곱셈을 계산하는 프로그램을 작성하라. 이 때 $(A, B) * (C, D) = (A + iB) * (C + iD) = (AC - BD) + i(AD + BC)$다. 행렬의 크기가 모두 $n \times n$일 때 덧셈 및 곱셈 수행 횟수는 어떻게 되는가?

f. 2-f 의 Magic 함수는 n의 크기에 상관없이 51×51 크기의 배열을 사용한다. $n < 51$인 경우에는 여분의 공간이 남게 되고 $n > 51$인 경우에는 예외가 발생하게 된다. 동적 메모리 할당을 사용하면 이런 문제를 해결할 수 있다. Magic 함수를 수정하고 코드를 테스트하라.