Data Structure, Week 1 – Solution

Performance Analysis and Measurement

1. 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]);
		}
	}
}
```

풀이 : 함수는 짧지만 재귀 함수이기 때문에 생각보다 시간 복잡도를 구하는데 어려움이 많다. 배열 a의 크기가 n이라고 가정하고, 배열 a 전체에 대해 Permutations 을 수행한다고 하자. 이 경우 $k = n - 1$일 때 걸리는 시간은 $\Theta(n)$이다.

$k < n - 1$일 때는 else 문이 수행된다. 이 때 두 번째 for 문은 $n - k$번 수행된다. 이 반복문을 한 번 실행할 때마다 $\Theta(t_{Permutations}(k + 1, n - 1))$의 시간이 걸린다. 그래서 $k < n - 1$일 때 $t_{Permutations}(k, n - 1) = \Theta((n - k)t_{Permutations}(k + 1, n - 1))$이 된다. 이 순환을 풀면 $n \geq 1$에 대해 $t_{Permutations}(0, n - 1) = \Theta(n(n!))$라는 결론을 얻는다.

2. 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;
        }
}
```

풀이 : 코드는 길어보이지만, 시간 복잡도를 구하는 일은 크게 어렵지 않다. 핵심은 반복문이 얼마나 반복되는가인데, 첫 번째 for 문은 $n+1$번 수행되므로 $\Theta(n)$이다. while 문은 한 번 수행될 때마다 key 값이 1 씩 증가하므로 n^2-1번 수행된다. 따라서 $\Theta(n^2)$이다.

두 번째 for 문도 $n+1$번 수행되므로 $\Theta(n)$이다. 따라서 Magic 함수의 시간 복잡도는 $\Theta(n^2)$이다.

3. 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]);
        }
}
```

풀이 : PPT 만 제대로 봤어도 답을 쉽게 구할 수 있는 문제다. 사실 안봐도 너무 쉬운 문제다. 이중 for 문이 있는데 첫 번째 for 문은 $n+1$번 수행되고 두 번째 for 문은 $n-i$번 수행된다. 따라서 SelectionSort 의 시간 복잡도는 $\Theta\big((n+1)(n-i)\big) = \Theta(n^2)$가 된다.

4. 다음 순환식의 시간 복잡도를 구하고, Master Theorem 을 통해 맞는지 확인해 보라.

- The Form : $T(n) = \begin{cases} c, \ if \ n < d \\ aT(n/b) + f(n), \ if \ n \geq d \end{cases}$
- The Master Theorem : ($\varepsilon > 0$ is any constant)
1. If $f(n)$ is $O(n^{log_b a - \varepsilon})$, then $T(n)$ is $\Theta(n^{log_b a})$.
2. If $f(n)$ is $\Theta(n^{log_b a}(\log n)^k)$, then $T(n)$ is $\Theta(n^{log_b a}(\log n)^{k+1})$.
3. If $f(n)$ is $\Omega(n^{log_b a + \varepsilon})$, then $T(n)$ is $\Theta(f(n))$, provided $af(n/b) \leq \delta f(n)$ for some $\delta < 1$.

a. $T(n) = T\left(\frac{n}{2}\right) + \frac{1}{2}n^2 + n$
b. $T(n) = 2T\left(\frac{n}{4}\right) + \sqrt{n} + 42$
c. $T(n) = 3T\left(\frac{n}{2}\right) + \frac{3}{4}n + 1$
d. $T(n) = 2T\left(\frac{n}{2}\right) + n\log n$

풀이 : 1 주차 과제 중에서 가장 어려운 문제다. Master Theorem 에 대한 이해도 필요한 과제다. 마스터 정리란 재귀 관계식으로 표현한 알고리즘의 동작 시간을 점근적으로 간단하게 계산하는 방법이다. 일단 각 문제를 풀어본 뒤에 마스터 정리를 어떻게 적용할 수 있는지 살펴보도록 하자.

a.

$$T(n) = T\left(\frac{n}{2}\right) + \frac{1}{2}n^2 + n$$
$$= T\left(\frac{n}{4}\right) + \frac{1}{2}\left(\frac{n}{2}\right)^2 + \frac{n}{2} + \frac{1}{2}n^2 + n = T\left(\frac{n}{2^2}\right) + \frac{1}{2}n^2\left(1 + \frac{1}{2^2}\right) + n\left(1 + \frac{1}{2}\right)$$
$$= \cdots = T\left(\frac{n}{2^k}\right) + \frac{1}{2}n^2\left(1 + \frac{1}{2^2} + \cdots + \frac{1}{2^{2(k-1)}}\right) + n\left(1 + \frac{1}{2} + \cdots + \frac{1}{2^{k-1}}\right)$$

이 때, $n = 2^k$라고 가정하자. 그러면 T(1) = 1 이므로

$$= T\left(\frac{2^k}{2^k}\right) + \frac{1}{2}n^2\left(1 + \frac{1}{2^2} + \cdots + \frac{1}{2^{2(k-1)}}\right) + n\left(1 + \frac{1}{2} + \cdots + \frac{1}{2^{k-1}}\right)$$
$$= T(1) + \frac{1}{2}n^2\frac{1 - \left(\frac{1}{4}\right)^k}{1 - \frac{1}{4}} + n\frac{1 - \left(\frac{1}{2}\right)^k}{1 - \frac{1}{2}}$$
$$= O(1) + \frac{2}{3}n^2\left(1 - \left(\frac{1}{4}\right)^k\right) + 2n\left(1 - \left(\frac{1}{2}\right)^k\right)$$

$n = 2^k$이므로, $k = \log_2 n$이다. 따라서,

$$= O(1) + \frac{2}{3}n^2\left(1 - \left(\frac{1}{4}\right)^{\log_2 n}\right) + 2n\left(1 - \left(\frac{1}{2}\right)^{\log_2 n}\right)$$
$$= O(1) + \frac{2}{3}n^2\left(1 - (2^{-2})^{\log_2 n}\right) + 2n\left(1 - (2^{-1})^{\log_2 n}\right)$$
$$= O(1) + \frac{2}{3}n^2\left(1 - n^{-2\log_2 2}\right) + 2n\left(1 - n^{-\log_2 2}\right)$$

$$= O(1) + \frac{2}{3}n^2\left(1 - \frac{1}{n^2}\right) + 2n\left(1 - \frac{1}{n}\right) = O(1) + O(n^2) + O(n) = O(n^2)$$

이제 마스터 정리를 이용해서 구한 시간 복잡도가 맞는지 확인해 보자.

$aT(n/b) + f(n)$에서 $a = 1$, $b = 2$, $f(n) = \frac{1}{2}n^2 + n = O(n^2) = \Omega\left(n^{\log_2 1+\varepsilon}\right)$이므로

시간 복잡도는 $\Theta\left(f(n)\right) = \Theta(n^2)$이 된다.

b.

$$T(n) = 2T\left(\frac{n}{4}\right) + \sqrt{n} + 42$$
$$= 2\left(2T\left(\frac{n}{16}\right) + \sqrt{\frac{n}{4}} + 42\right) + \sqrt{n} + 42 = 4T\left(\frac{n}{16}\right) + 2\cdot\frac{\sqrt{n}}{2} + 2\cdot 42 + \sqrt{n} + 42$$
$$= 2^2T\left(\frac{n}{4^2}\right) + 2\sqrt{n} + 42(1+2)$$
$$= \cdots = 2^kT\left(\frac{n}{4^k}\right) + k\sqrt{n} + 42(1 + 2 + \cdots + 2^{k-1})$$

이 때, $n = 4^k$라고 가정하자. 그러면 T(1) = 1 이므로

$$= 2^kT\left(\frac{4^k}{4^k}\right) + k\sqrt{n} + 42\left(1 + 2 + \cdots + 2^{k-1}\right)$$
$$= 2^kT(1) + k\sqrt{n} + 42\left(\frac{1 - \left(\frac{1}{2}\right)^k}{1 - \frac{1}{2}}\right) = 2^kO(1) + k\sqrt{n} + 84\left(1 - \left(\frac{1}{2}\right)^k\right)$$

$n = 4^k$이므로, $k = \log_4 n$이다. 따라서,

$$= 2^{\log_4 n}O(1) + \log_4 n \cdot \sqrt{n} + 84\left(1 - \left(\frac{1}{2}\right)^{\log_4 n}\right)$$
$$= n^{log_4 2}O(1) + \log_4 n \cdot \sqrt{n} + 84\left(1 - 2^{-\log_4 n}\right)$$
$$= n^{\frac{1}{2}}O(1) + \log_4 n \cdot \sqrt{n} + 84\left(1 - n^{-log_4 2}\right)$$
$$= \sqrt{n}O(1) + \log_4 n \cdot \sqrt{n} + 84\left(1 - \frac{1}{\sqrt{n}}\right) = O\left(\sqrt{n}\right) + O\left(\sqrt{n}\log n\right) + O(1) = O(\sqrt{n}\log n)$$

이제 마스터 정리를 이용해서 구한 시간 복잡도가 맞는지 확인해 보자.

$aT(n/b) + f(n)$에서 $a = 2$, $b = 4$, $f(n) = \sqrt{n} + 42 = O\left(\sqrt{n}\right) = \Theta\left(n^{\log_4 2}(\log n)^0\right)$이므로

시간 복잡도는 $\Theta(n^{log_b a}(\log n)^{k+1}) = \Theta(\sqrt{n}\log n)$이 된다.

c.

$$T(n) = 3T\left(\frac{n}{2}\right) + \frac{3}{4}n + 1$$
$$= 3\left(3T\left(\frac{n}{4}\right) + \frac{3}{4}\left(\frac{n}{2}\right) + 1\right) + \frac{3}{4}n + 1 = 9T\left(\frac{n}{4}\right) + \frac{3^2}{4}\left(\frac{n}{2}\right) + 3 + \frac{3}{4}n + 1$$
$$= 3^2T\left(\frac{n}{2^2}\right) + \frac{3}{4}n\left(1 + \frac{3}{2}\right) + (1+3)$$
$$= \cdots = 3^kT\left(\frac{n}{2^k}\right) + \frac{3}{4}\left(1 + \frac{3}{2} + \cdots + \left(\frac{3}{2}\right)^{k-1}\right) + \left(1 + 3 + \cdots + 3^{k-1}\right)$$

이 때, $n = 2^k$라고 가정하자. 그러면 T(1) = 1 이므로

$$= 3^k T\left(\frac{2^k}{2^k}\right) + \frac{3}{4}\left(1 + \frac{3}{2} + \cdots + \left(\frac{3}{2}\right)^{k-1}\right) + \left(1 + 3 + \cdots + 3^{k-1}\right)$$

$$= 3^k T(1) + \frac{3}{4}\frac{\left(\frac{3}{2}\right)^k - 1}{\frac{3}{2} - 1} + \frac{3^k - 1}{3 - 1} = 3^k O(1) + \frac{3}{2}\left(\left(\frac{3}{2}\right)^k - 1\right) + \frac{1}{2}\left(3^k - 1\right)$$

$n = 2^k$이므로, $k = \log_2 n$이다. 따라서,

$$= 3^{\log_2 n} O(1) + \frac{3}{2}\left(\left(\frac{3}{2}\right)^{\log_2 n} - 1\right) + \frac{1}{2}\left(3^{\log_2 n} - 1\right)$$

$$= n^{\log_2 3} O(1) + \frac{3}{2}\left(n^{\log_2 \frac{3}{2}} - 1\right) + \frac{1}{2}\left(n^{\log_2 3} - 1\right)$$

$$= O\left(n^{\log_2 3}\right) + O\left(n^{\log_2 \frac{3}{2}}\right) + O\left(n^{\log_2 3}\right) = O\left(n^{\log_2 3}\right)$$

이제 마스터 정리를 이용해서 구한 시간 복잡도가 맞는지 확인해 보자.

$aT(n/b) + f(n)$에서 $a = 3$, $b = 2$, $f(n) = \frac{3}{4}n + 1 = O(n) = O\left(n^{\log_2 3 - \varepsilon}\right)$이므로

시간 복잡도는 $\Theta(n^{\log_b a}) = \Theta(n^{\log_2 3})$이 된다.

$$T(n) = 2T\left(\frac{n}{2}\right) + n \log n$$

$$= 2\left(2T\left(\frac{n}{4}\right) + \frac{n}{2}\log\frac{n}{2}\right) + n\log n = 4T\left(\frac{n}{4}\right) + n\log\frac{n}{2} + n\log n$$

$$= 2^2 T\left(\frac{n}{2^2}\right) + n\left(\log n + \log\frac{n}{2}\right) = \cdots = 2^k T\left(\frac{n}{2^k}\right) + n\left(\log n + \log\frac{n}{2} + \cdots + \log\frac{n}{2^{k-1}}\right)$$

이 때, $n = 2^k$라고 가정하자. 그러면 T(1) = 1 이므로

$$= 2^k T\left(\frac{2^k}{2^k}\right) + n\left(\log n + \log\frac{n}{2} + \cdots + \log\frac{n}{2^{k-1}}\right)$$

$$= nT(1) + n\log\left(n \cdot \frac{n}{2} \cdot \ldots \cdot \frac{n}{2^{k-1}}\right) = nO(1) + n\log\frac{n^k}{2^{1+\cdots+(k-1)}}$$

$$= nO(1) + n\log\frac{n^k}{2^{\frac{k(k-1)}{2}}} = nO(1) + n\log\frac{2^{k^2}}{2^{\frac{k^2-k}{2}}} = nO(1) + n\log 2^{k^2 - \frac{k^2-k}{2}}$$

$$= nO(1) + n\log 2^{\frac{k^2+k}{2}} = nO(1) + n\frac{k^2+k}{2}\log 2$$

$n = 2^k$이므로, $k = \log_2 n$이다. 따라서,

$$= nO(1) + n\frac{(\log_2 n)^2 + \log_2 n}{2}\log 2 = O(n) + O(n(\log n)^2) = O(n(\log n)^2)$$

이제 마스터 정리를 이용해서 구한 시간 복잡도가 맞는지 확인해 보자.

$aT(n/b) + f(n)$에서 $a = 2$, $b = 2$, $f(n) = n\log n = O(n\log n) = O\left(n^{\log_2 2}(\log n)^1\right)$이므로

시간 복잡도는 $\Theta(n^{\log_b a}(\log n)^{k+1}) = \Theta(n(\log n)^2)$이 된다.