Chapter 2: Algorithm Analysis

* Algorithm Analysis:

taking a CPU = 1 GHz = 10^9 Hz

```
for (i=0 ; i<n ; i++)        --- n
    for (j=0 ; j<n ; j++)    --- n
        for (K=0 ; K<n ; K++)  --- n
            // code
```

taking $n = 10^6$, we have n^3 iterations , and:

$$\text{Time} = \frac{(10^6)^3}{10^9} = \frac{10^{18}}{10^9} = 10^9 \text{ sec} \approx 31 \text{ years}$$

if we have 2 loops only, then:

$$\text{Time} = \frac{(10^6)^2}{10^9} = 10^3 \text{ sec} \approx 16 \text{ min}$$

but if we have one loop only, then:

$$\text{Time} = \frac{10^6}{10^9} = 10^{-3} \text{ sec}$$

So, we should study algorithms for huge data only

note: Algorithms can be studied only in loops (for loop, while loop, do while loop and recursions).

* Mathmatical Definitions:

1) $T(n) = O(f(n))$, if there are constants C and n_0, such that $T(n) \leq C.(f(n))$ when $n \geq n_0$.

2) $T(n) = \Omega(g(n))$, if there are constants C and n_0, such that $T(n) \geq C.(g(n))$ when $n \geq n_0$.

3) $T(n) = \Theta(h(n))$, if and only if, $T(n) = O(h(n))$ and $T(n) = \Omega(h(n))$.

Note that $1000n < n^2$, since that algorithms should be studied only for huge data.

$C << n$, so we don't need to consider the constant.

Taking $T_1(n) = O(f(n))$, $T_2(n) = O(g(n))$, then:

1) $T_1(n) + T_2(n) = \max(O(f(n)), O(g(n)))$

2) $T_1(n) * T_2(n) = O(f(n) * g(n))$

(T_1 & T_2 are consecutive in point (1), while they are overlapping in point (2)).

,

ex 1: $f(n) = 7n^3 + 15n^2 + 3n + 9 \leq 7n^3 + 15n^3 + 3n^3 + 9n^3$

$\Rightarrow T(n) = O(n^3)$

$[T_1$

$[T_2$

ex 2: $f(n) = n^2$, $g(n) = n$

$n^2 [\; [n$

$T(n) = O(f(n) * g(n))$
$= O(n^3)$

ex3:
```
for (i=0 ; i<n ; i++)          --- n
   for (j=0 ; j < n*n ; j++)   --- n²
       // code
```

$T(n) = O(n^3)$

ex4:
```
for ( i=0 ; i<n ; i++)          --- n
   for (j=i ; j < i*i ; j++)    --- n²-n
   // code
```

the max value of i is n , so we have n iterat[ions] in the first loop and max value of $n^2 - n$ for the number of iterations in the second loop.

$f(n) = n$, $g(n) = n^2 - n$

$\Rightarrow T(n) = O(f(n) * g(n))$
$= O(n^3)$

n^2

ex 5:
```
for (i=0 ; i<n ; i++)                    --- n.
    for ( j=i ; j < i*i ; j++)           --- n^2-n
        for ( K=j ; K < j*j ; K++)       --- n^4-n^2
            // code
```

n^2-n n^2 $n^2 * n^2$ n^4

```
for (t=0; t < n*n ; t++)     --- n^2
    // code
```

$n(n^2-n)(n^4-n^2)$

$(n^3 - n^2)$

n^7

$$T(n) = n(n^2-n)(n^4-n^2) + n^2$$

$$\Rightarrow T(n) = \boxed{O(n^7)}$$

ex 6:

$$fact(n) = \begin{cases} 1 & , n=0 \\ n * fact(n-1) & , n>0 \end{cases}$$

```
long fact (int n){
    if (n==0)
        return 1;
    else
        return n* fact(n-1);
}
```

$$T(n) = \begin{cases} d & ; n=0 \\ T(n-1)+\boxed{c} & ; n>0 \end{cases}$$

$$\left.\begin{array}{l}
T(\cancel{n}) = T(\cancel{n-1}) + c \\
T(\cancel{n-1}) = T(\cancel{n-2}) + c \\
T(\cancel{n-2}) = T(\cancel{n-3}) + c \\
T(\cancel{n-3}) = T(\cancel{n-4}) + c \\
T(\cancel{n-4}) = T(\cancel{n-5}) + c \\
\vdots \qquad\qquad \vdots \\
T(\cancel{1}) = T(0) + c
\end{array}\right] nc \quad +$$

$$T(n) = d + nC$$
$$T(n) = O(n)$$

ex7: Merge Sort:

$$T(n) = \begin{cases} d, & n=1 \\ 2T\left(\frac{n}{2}\right) + n, & n>1 \end{cases}$$

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$
$$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{2^2}\right) + n/2$$
$$T\left(\frac{n}{2^2}\right) = 2T\left(\frac{n}{2^3}\right) + n/2^2$$
$$T\left(\frac{n}{2^3}\right) = 2T\left(\frac{n}{2^4}\right) + n/2^3$$
$$\vdots \qquad\qquad \vdots$$

Now, $T(n) = 2\left[2T\left(\frac{n}{2^2}\right) + n/2\right] + n$

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

Similarly, $T(n) = 2^2 \left[2T\left(\frac{n}{2^3}\right) + \frac{n}{2^2} \right] + 2n$

$$T(n) = 2^3 \, T\left(\frac{n}{2^3}\right) + 3n$$

$$\vdots \qquad \vdots$$

$$T(n) = 2^K \, T\left(\frac{n}{2^K}\right) + Kn$$

Now, let $\frac{n}{2^K} = 1 \Rightarrow n = 2^K \Rightarrow K = \log_2 n$

$$\log n = \begin{cases} \log_t n < 1 & , n < t \\ 1 & , n = t \\ \log_t n > 1 & , n > t \end{cases}$$

So, $T(n) = n\,T(1) + n \log n = nd + n \log n$

and since $n \log n > n$:

$\cancel{O(n)}$ $T(n) = O(n \log n)$.

* Insertion Sort :

```
for ( j=2 ; j <=n ; j++){

        Key = A[j];
        i = j-1;

        while ((i>0) && (A[i] > Key)){
                A[i+1] = A[i];
                i--;
        }

        A[i+1]= Key;
}
```

A: 7, 2, 10, 9, 5, 13

j	i	Key	A (indices 1–6)
2	1	2	[7, , , , ,]
	0		[2, 7, , , ,]
3	2	10	[2, 7, , , ,]
			[2, 7, 10, , ,]
4	~~3~~	9	[2, 7, 10, , ,]
	2		[2, 7, 9, 10, ,]
5	~~4~~	5	[2, 7, 9, 10, ,]
	~~3~~		[2, 7, 9, , 10,]
	~~2~~		[2, 7, , 9, 10,]
	1		[2, 5, 7, 9, 10,]
6	5	13	[2, 5, 7, 9, 10, 13]

```
for (j=2 ; j<=n ; j++){                          // c1

    Key = A[j];                                  // c2
    i = j-1;                                     // c3

    while ((i>0) && (A[i]>Key)){                 // c4
        A[i+1] = A[i];                           // c5
        i--;                                     // c6
    }

    A[i+1] = Key;                                // c7
}
```

$T(c_4) = T_2 + T_3 + T_4 + \dots + T_n$

$= \sum_{2}^{n} T_j$

We have three cases for the time:

1) Best Case 2) Worst Case 3) Average Case

For this example:

1) Best case (Already Sorted):

$T(n) = c_1(n) + c_2(n-1) + c_3(n-1) + c_4(n-1) + c_5(0) + c_6(0) + c_7(n-1)$

$T(n) = cn + d$

$T(n) = O(n)$

2) Worst Case (Sorted in Descending Order):

This will be like the following code:

```
for (i = 2 ; i <= n ; i++){        --- n
    for (j = i ; j > 0 ; j--)      --- n
        // code
}
```

So, $T(n) = O(n^2)$

3) Average Case (Random Case):

$$T(n) = c_1 (n) + c_2 (n-1) + c_3 (n-1) + c_4 \left(\sum_{j=2}^{n} T_j \right) + c_5 \left(\sum_{j=2}^{n} (T_j - 1) \right) + c_6 \left(\sum_{j=2}^{n} (T_j - 1) \right) + c_7 (n-1)$$

$$= dn^2 + Kn + C$$

$$= O(n^2)$$

Note that: $\sum_{j=2}^{n} T_j = \frac{n(n+1)}{2} - 1$

$$\sum_{j=2}^{n} (T_j - 1) = \frac{n(n-1)}{2} - 1$$

note: the difference between bubble sorting & insertion sort is that:

in bubble sorting:

```
for (i=1; i<n ; i++)      ---n
    for (j=1; j<n ; j++)  ---n
        //code
```

So, $T(n) = O(n^2)$ always in bubble sorting (for best, worst and average case), but in insertion sort, $T(n)$ will differ from a case to another and it will depend on n, so the insertion sort is better than bubble sorting.

ex 8: $T(n) = \begin{cases} d, & n=1 \\ 2T(\frac{n}{2}) + 10, & n>1 \end{cases}$

$T(n) = 2T(n/2) + 10$
$T(n/2) = 2T(n/2^2) + 10$
$T(n/2^2) = 2T(n/2^3) + 10$
$T(n/2^3) = 2T(n/2^4) + 10$
$\vdots$

Now, $T(n) = 2T(n/2) + 10$
$= 2\left[2T(n/2^2) + 10\right] + 10$
$= 2^2\,T(n/2^2) + 2(10) + 10$
$= 2^2\left[2T(n/2^3) + 10\right] + 2(10) + 10$

$$T(n) = 2^3\, T(n/2^3) + 2^2(10) + 2(10) + 10$$

$\vdots \qquad \vdots$

$$T(n) = 2^K\, T(n/2^K) + 2^{K-1}(10) + 2^{K-2}(10) + \dots + 10$$

$$= 2^K\, T(n/2^K) + 10\,(2^{K-1} + 2^{K-2} + \dots + 2^0)$$

$$= 2^K\, T(n/2^K) + 10 \sum_{i=0}^{K-1} 2^i$$

$$= 2^K\, T(n/2^K) + 10\,(2^K - 1) \quad \text{(geometric series)}$$

Now, let $\frac{n}{2^K} = 1 \Longrightarrow n = 2^K$, and:

$$T(n) = n\,T(1) + 10(n-1)$$

$$= nd + 10n - 10$$

$$= O(n)$$

note: $\sum_{i=0}^{K-1} a^i = a^{K-1} + a^{K-2} + a^{K-3} + \dots + a^0$

$$= a^{K-1} + a^{K-2} + \dots + a^0 * \frac{a-1}{a-1}$$

$$= \frac{(a^K + \cancel{a^{K-1}} + \cancel{\dots} + \cancel{a}) - (\cancel{a^{K-1}} + \cancel{a^{K-2}} + \cancel{\dots} + a^0)}{a-1}$$

$$\Longrightarrow \boxed{\sum_{i=0}^{K-1} a^i = \frac{a^K - 1}{a - 1}}$$

and so, $\sum_{i=0}^{K-1} 2^i = \frac{2^K - 1}{2 - 1} = 2^K - 1$

ex9: Find T(n) for the following code:

```
i=n;
while (i>0){
    for (j=0; j<i; j++){
        // code
    }
    i/=2;
}
```

N

$$T(n) = n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \frac{n}{16} + \dots$$

$$= n\left[\frac{1}{1} + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \dots\right]$$

$$= n \sum_{i=0}^{\infty} \left(\frac{1}{2}\right)^i$$

$$= n\left[\frac{1}{1-\frac{1}{2}}\right]$$

$$= 2n$$

$$= O(n)$$

note: $\sum_{i=0}^{\infty} a^i = \frac{1}{1-a}$, $|a| < 1$ (Geometric Series