Made By:
Kenan Gazwan

Analysis of Recursive Algorithms

ICS202-Summary

King Fahd University of Petroleum and Minerals

Telegram: @KenanGazwan

(Analysis of Recursive Algorithms)

✓ Recurrence Relation

A recurrence relation, T(n), is a recursive function of integer variable n.

Like all recursive functions, it has both: recursive case and base case.

Ex:

$$T(n) = \begin{cases} a \quad , & if\ n = 1 \\ 2T(n) + bn + c, & if\ n > 1 \end{cases}$$

(Base case of the recurrence relation)

(Recurrent or recursive case)

✓ Recurrence relations are useful for expressing the running times (i.e., the number of basic operations executed) of recursive algorithms.

To find the running time of recursive algorithms, we have two steps:

1) Forming Recurrence Relation
2) Solving Recurrence Relation

(1) Forming Recurrence Relation

Example (1) : this recurrence relation describes the number of **comparisons** carried out by the following method.

```
public void fun(int n){
        if(n > 0) {
                System.out.println(n);
                fun(n – 1);
        }
}
```

- **When n = 0**, the base case is reached.
  The number of comparisons is 1, and hence, $T(0) = 1$.
- **When n > 0**, the number of comparisons is $1 + T(n - 1)$.

Therefore, the recurrence relation is:

$$T(n) = \begin{cases} 1\ , & if\ n = 0 \\ T(n-1) + 1, & if\ n > 0 \end{cases}$$

Example (2) : this recurrence relation describes the number of **print statements** carried out by the following method.

```
public void fun(int n){
        if(n > 0) {
                System.out.println(n);
                fun(n – 1);
        }
}
```

- **When n = 0**, the base case is reached.
  The number of print statements is 0, and hence, $T(0) = 0$
- **When n > 0**, the number of print statements is $1 + T(n - 1)$.

Therefore, the recurrence relation is:

$$T(n) = \begin{cases} 0\ , & if\ n = 0 \\ T(n-1) + 1, & if\ n > 0 \end{cases}$$

Example (3) : this recurrence relation describes the number of **additions** carried out by the following method.

```
public int fun(int n){
 if(n = 1)
  return 2;

 else
  return 3* fun(n/2) + fun(n/2) + 5;
}
```

- **When n = 1**, the base case is reached.
  The number of additions is 0, and hence, $T(1) = 0$
- **When n > 1**, the number of additions is $2 + 2T(\frac{n}{2})$

Therefore, the recurrence relation is:

$$T(n) = \begin{cases} 0\,, & if\ n = 1 \\ 2T\left(\frac{n}{2}\right) + 2\ , & if\ n > 1 \end{cases}$$

(2) Solving Recurrence Relation:

The Steps to Solve:

1) **Expand the recurrence.**
2) **Express the expansion as a summation by plugging the recurrence back into itself until you see a pattern.**
3) **Evaluate the summation.**

Example : Form and solve the recurrence relation describing the number of **multiplications** carried out by the factorial method:

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

Forming

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

Solving

```
T(n) = T(n-1) + 1
T(n) = [T(n-2) + 1] + 1 => T(n-2) + 2
T(n) = [T(n-3) + 1] + 2 => T(n-3) + 3
...
T(n) = T(n-i) + i

From the recurrence relation, the function
stops when             → T(n) = 0,
which means            → n - i = 0
                       → n = i
substitute each i by n → T(n) = T(n-n) + n
                       → T(n) = T(0) + n
                       → T(n) = 0 + n
therefore              → T(n) = O(n)
```

This Example Is to Explain the Idea in general. There are Still Some Special Cases.