Data Structures
Chapter 2

1. **Recurrence Relations**
   - Recurrence Equations
   - Folding
   - Telescoping
2. Discrete Math
3. Structures

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept., Handong Global University

그러므로 예수께서 자기를 믿은 유대인들에게 이르시되 너희가 내 말에 거하면 참으로 내 제자가 되고 진리를 알지니 진리가 너희를 자유롭게 하리라 (요8:31-32)

Recurrence Relations - 점화식

- **Recurrence relation** is an equation that recursively defines a sequence or multidimensional array of values, once one or more initial terms are given: each further term of the sequence or array is defined as a function of the preceding terms.

- Time Complexity Example

$a_n = a_{n-1} + 1$

$T(n) = T(n-1) + 1$

- Sequence Example

$a_n = a_{n-1} + n$

$T(n) = T(n-1) + n$

Recurrence Relations: Time Complexity Example

- Consider a recursive function called foo() as shown below:

```
void foo(int n) {
  if (n > 0) {
    cout << n << end;
    foo(n - 1);
  }
}
```

- What is the time complexity of the function foo() in terms of n or T(n)?

```
      foo(3)                                  T(3)
 cout 3      foo(2)                           T(2) + 1
      cout 2       foo(1)                     T(1) + 1
             cout 1       foo(0)              T(0)
                          return
```

```
T(0) = 1
T(n) = T(n - 1) + 1
```

Recurrence Relations: Time Complexity Example

- Consider a recursive function called foo() as shown below:

```
void foo(int n) {
  if (n > 0) {
    cout << n << end;
    foo(n - 1);
  }
}
```

```
T(n)
1
1
T(n - 1)
```

- What is the time complexity of the function foo() in terms of n or T(n)?

```
T(n) = T(n - 1) + 2

T(n) = T(n - 1) + c

T(n) = T(n - 1) + 1
T(0) = 1
```

Recurrence Relations

- **For example,**
  we may describe that the time complexity of the linear search is
  - $T(1) = c_0$ // the time to process an array of 1 element is 1
  - $T(n) = T(n-1) + c_0$ // to process n items is to process (n-1) elements + 1 element

  We may express the constants c_0 and c as 1, respectively, for our purpose. The expressions become:
  - $T(0) = 1$
  - $T(n) = T(n-1) + 1$

Recurrence Relations

In general, there are two ways to solve **recurrence relations**:

- **"Unfolding"**
  - It does ***repeated substitutions*** applying the recursive rule <u>*until the base case is reached.*</u>
- **"Telescoping"** consists in the following:
  1. ***Rewrite the recurrence relation*** *for subsequent* ***smaller values of n*** so that the left side of the rewritten relation is equal to the first term of the right side of the previous relation. Rewrite the relation <u>*until a relation involving the initial condition is obtained.*</u>
  2. Add the left sides and the right sides of the relations.
     Cancel the equal terms on the left and the right side.
  3. Add the remaining terms on the right side.
     The sum will give the general formula (or **closed form**) of the sequence.

Recurrence Relations: Linear search

- **Telescoping:** Rewrite the equation for a smaller value of n, $T(0) = 1$

open form $T(n) = T(n-1) + 1$,   Replace n with $n - 1$ both sides, it becomes

  - $T(n-1) = T(n-2) + 1$
  - $T(n-2) = T(n-3) + 1$
  - …
  - $T(2) = T(1) + 1$
  - $T(1) = T(0) + 1$
- Add the left and right sides:
  - $T(n) + T(n-1) + T(n-2) + \ldots + T(2) + T(1) = T(n-1) + T(n-2) + \ldots + T(0) + 1 + 1 + \ldots + 1$
- Cancel equal terms on both sides:
  - $T(n) = T(0) + 1 + 1 + \ldots + 1$

Recurrence Relations: Linear search

- **Telescoping:** Rewrite the equation for a smaller value of n, $T(0) = 1$

open form $T(n) = T(n-1) + 1$, Replace n with $n-1$ both sides, it becomes

  - $T(n-1) = T(n-2) + 1$
  - $T(n-2) = T(n-3) + 1$
  - …
  - $T(2) = T(1) + 1$
  - $T(1) = T(0) + 1$
- Add the left and right sides:
  - $T(n) + T(n-1) + T(n-2) + \ldots + T(2) + T(1) = T(n-1) + T(n-2) + \ldots + T(0) + 1 + 1 + \ldots + 1$
- Cancel equal terms on both sides:
  - $T(n) = T(0) + 1 + 1 + \ldots + 1$
  - How many n's are there?
  - $T(n) = T(0) + n$
  - $\quad = 1 + n$
- Therefore, $T(n) = O(n)$. closed form

Recurrence Relations: Linear search

- **Unfolding:** repeated substitutions, $T(0) = 1$

open form $\mathbf{T(n) = T(n-1) + 1}$,

Since $\mathbf{T(n-1)} = T((n-1)-1) + 1 = \mathbf{T(n-2) + 1}$

Since $\mathbf{T(n-2)} = T((n-2)-1) + 1 = \mathbf{T(n-3) + 1}$

  - $T(n) = T(n-2) + 1 + 1 = T(n-2) + 2$
  - $T(n) = T(n-3) + 1 + 2 = T(n-3) + 3$
  - $T(n) = T(n-4) + 1 + 3 = T(n-4) + 4$
  - …
  - $T(n) = T(n-(n-1)) + 1 + (n-2)$
  - $T(n) = T(n-n) + 1 + n - 1 = T(0) + n$

Recurrence Relations: Linear search

- **Unfolding:** repeated substitutions, $T(0) = 1$

open form

$\mathbf{T(n) = T(n - 1) + 1,}$

Since $\mathbf{T(n - 1)} = T((n - 1) - 1) + 1 = \mathbf{T(n - 2) + 1}$

Since $\mathbf{T(n - 2)} = T((n - 2) - 1) + 1 = \mathbf{T(n - 3) + 1}$

  - $T(n) = T(n - 2) + 1 + 1 = T(n - 2) + 2$
  - $T(n) = T(n - 3) + 1 + 2 = T(n - 3) + 3$
  - $T(n) = T(n - 4) + 1 + 3 = T(n - 4) + 4$
  - …
  - $T(n) = T(n - (n - 1)) + 1 + (n - 2)$
  - $T(n) = T(n - n) + 1 + n - 1 = T(0) + n$

  Since $T(n) = T(0) + n$
  - $= 1 + n$

closed form

- Therefore, $T(n) = O(n)$.

Recurrence Relations: Sequence Example

- Consider the sequence 1, 3, 6, 10, 15, 21, 28, 36, … Each term is obtained from the previous by adding the number that shows the position of the current term to the previous term, e.g., 10 is in position 4 and the previous term is 6: 10 = 6 + 4. **What is a formula for this sequence in terms of n or a(n)?**

Recurrence Relations: Sequence Example

- Consider the sequence 1, 3, 6, 10, 15, 21, 28, 36, … Each term is obtained from the previous by adding the number that shows the position of the current term to the previous term, e.g., 10 is in position 4 and the previous term is 6: 10 = 6 + 4. **What is a formula for this sequence in terms of n or a(n)?**

a(1) = 1
a(2) = 3
a(3) = 6
a(4) = 10
…
a(10) =
a(100) =
a(n) = ?

Recurrence Relations: Sequence Example

- Consider the sequence 1, 3, 6, 10, 15, 21, 28, 36, … Each term is obtained from the previous by adding the number that shows the position of the current term to the previous term, e.g., 10 is in position 4 and the previous term is 6: 10 = 6 + 4. **What is a formula for this sequence in terms of n or a(n)?**

a(1) = 1	
a(2) = 3	a(2) = a(1) +
a(3) = 6	a(3) = a(2) +
a(4) = 10	a(4) = a(3) +
…	
a(10) = 55	
a(100) = 5050	
a(n) = ?	**a(n) = a(n – 1) + n**

Recurrence Relations: Sequence Example

- Consider the sequence 1, 3, 6, 10, 15, 21, 28, 36, … Each term is obtained from the previous by adding the number that shows the position of the current term to the previous term, e.g., 10 is in position 4 and the previous term is 6: 10 = 6 + 4. **What is a formula for this sequence in terms of n or a(n)?**
  - $a(1) = 1$ for $n = 1$
  - $a(n) = a(n-1) + n$ for $n = 1, 2, 3, \ldots$
- What is a(n) in a **closed form** of n?

Recurrence Relations: Sequence Example

- **Telescoping:**

  open form

  Substitute n with n – 1, then a(n – 1) becomes

  a(n – 1) = a((n – 1) – 1) + (n – 1) = **a(n – 2) + (n – 1)**

  a(n) = a(n – 1) + n
  - a(n – 1) = a(n – 2) + (n – 1)
  - a(n – 2) = a(n – 3) + (n – 2)
  - ....
  - a(4) = a(3) + 4
  - a(3) = a(2) + 3
  - a(2) = a(1) + 2
- Add the left and right sides:
  - a(n) + a(n – 1) + a(n – 2) + … + a(3) + a(2) =
    a(n – 1) + a(n – 2) + … + a(2) + a(1) + 2 + 3 + 4 + (n – 1) + n
- Cancel equal terms on both sides:
  - a(n) = a(1) + 2 + 3 + … + (n – 1) + n
  - a(n) = 1 + 2 + 3 + … + (n – 1) + n
  - = n (n + 1) / 2
- Therefore, a(n) = n(n + 1) / 2

  closed form

Recurrence Relations: Selection sort

- The time complexity of the selection sort or bubble sort can be expressed in terms of recurrence relation as shown below:
  - $T(1) = 1$ // the time to process an array of 1 element is 1
  - $T(n) = n + T(n-1)$ // to process n items is to n comparisons + (n-1) items processing
- **Unfolding:**
  - $T(n) = n + T(n - 1)$. Since $T(n - 1) = (n - 1) + T(n - 2)$, it becomes
  - $T(n) = n + (n - 1) + T(n - 2)$ Since $T(n - 2) = (n - 2) + T(n - 3)$, it becomes
  - $T(n) = n + (n - 1) + (n - 2) + T(n - 3)$
  - …
  - $T(n) = n + (n - 1) + (n - 2) + \ldots + 2 + T(1)$
  - $T(n) = n + (n - 1) + (n - 2) + \ldots + 2 + 1$

Recurrence Relations: Selection sort

- The time complexity of the selection sort or bubble sort can be expressed in terms of recurrence relation as shown below:
  - $T(1) = 1$ // the time to process an array of 1 element is 1
  - $T(n) = n + T(n-1)$ // to process n items is to n comparisons + (n-1) items processing
- **Unfolding:**
  - $T(n) = n + T(n-1)$. Since $T(n-1) = (n-1) + T(n-2)$, it becomes
  - $T(n) = n + (n-1) + T(n-2)$ Since $T(n-2) = (n-2) + T(n-3)$, it becomes
  - $T(n) = n + (n-1) + (n-2) + T(n-3)$
  - …
  - $T(n) = n + (n-1) + (n-2) + \ldots + 2 + T(1)$
  - $T(n) = n + (n-1) + (n-2) + \ldots + 2 + 1$
- Therefore, $T(n) = n(n+1) / 2$; → $g(n) = n^2$
  $T(n)$ is big O of n^2 or $T(n) = O(n^2)$
- The time complexity of selection sort is $O(n^2)$.

Recurrence Relations: Selection sort

- **Telescoping:**
  - $T(n) = n + T(n-1)$
  - $T(n-1) = (n-1) + T(n-2)$
  - $T(n-2) = (n-2) + T(n-3)$
  - $\ldots$
  - $T(3) = 3 + T(2)$
  - $T(2) = 2 + T(1)$
- Add all terms in each side and cancel the equal terms, then it becomes
  - $T(n) = n + (n-1) + \ldots + 2 + T(1)$
- Therefore, $T(n) = n(n + 1) / 2$;
- The time complexity of selection sort is $O(n^2)$.

Recurrence Relations: Binary search

- The time complexity of binary search can be expressed in terms of recurrence relation as shown below:

  Base case: $T(1) = 1$

  Recurrence: $T(n) = 1 + T\left(\frac{n}{2}\right)$

```
int _binarySearch(int *a, int key, int lo, int hi) {
   int mid = (hi + lo)/2;

   if (lo == hi)       return -1;
   if (a[mid] == key) return  k;

   if (a[mid]  < key) return _binarySearch(a, key, mid+1, hi);
   else               return _binarySearch(a, key, lo,    mid-1);
}

// returns the index of key in the array if found, -1 if not found.
boolean binarySearch(int *a, int key, size){  // *a must be sorted.
   return _binarySearch(a, key, 0, size-1);
}
```

Recurrence Relations: Binary search

- Telescoping:

$$\begin{aligned} T(n) &= 1 + T(n/2) \\ T(n/2) &= 1 + T(n/4) \\ T(n/4) &= 1 + T(n/8) \\ &\dots \\ T(4) &= 1 + T(2) \\ T(2) &= 1 + T(1) \end{aligned}$$

- Sum up the left and right sides of the equations above:

$$T(n) = 1 + 1 + \ \dots + 1 + \ T(1)$$

- How many 1's on the right side?
  How many times do you need to divide n by 2 to reach 1?

$$T(n) = 1 + 1 + \ \dots + 1 + \ T(1)$$
$$T(n) = \log(n) + \ T(1)$$
$$T(n) = \log(n) + 1$$

$$\frac{n}{2^k} = 1$$
$$n \ = \ 2^k$$
$$\log_2 n = \log_2 2^k$$
$$\log_2 n = k$$

- The time complexity of selection sort is T(n) = O(log(n)).

Recurrence Relations: Binary search

Unfolding:

- $T(n) = 1 + T(n/2)$ $\quad\quad\quad\quad$ $Since\ T(n/2) = 1 + T(n/4)$

$$
\begin{aligned}
&= 1 + 1 + T(n/4) \\
&= 1 + 1 + 1 + T(n/8) \\
&= 1 + 1 + 1 + 1 + T(n/16) \\
&= 1 + 1 + \ldots\ldots + 1 + T(n/n) \\
&= 1 + 1 + \ldots\ldots + 1 + T(1) \\
&= 1k + T\left(\frac{n}{2^k}\right)
\end{aligned}
$$

- How many 1's on the right side?
  How many times do you need to divide n by 2 to reach 1?

$$T(n) = 1 + 1 + \ldots + 1 + T(1)$$
$$T(n) = \log(n) + T(1)$$
$$T(n) = \log(n) + 1$$

$$\frac{n}{2^k} = 1$$
$$n = 2^k$$
$$\log_2 n = \log_2 2^k$$
$$\log_2 n = k$$

- The time complexity of selection sort is T(n) = O(log(n)).

Asymptotic Analysis

- Suppose that two algorithms, A and B, solving the same problem have the running time of $O(n)$ and $O(n^2)$, respectively. Then this implies that algorithm A is **asymptotically better** than algorithm B.
- We can use the **big-Oh** notation to order classes of functions by **asymptotic growth rate**. Seven functions below are often used and ordered by increasing growth rate.

※ $O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n)$

n	log n	n	n log n	n^2	n^3	2^n
1	0	1	0	1	1	2
2	1	2	2	4	8	4
4	2	4	8	16	64	16
8	3	8	24	64	512	256
16	4	16	64	256	4,096	65,536
32	5	32	160	1,024	32,768	4,294,967,296
64	6	64	384	4,096	262,144	1.84 x 10^19
128	7	128	896	16,384	2,097,152	3.40 x 10^38
256	8	256	2,048	65,536	16,777,216	1.15 x 10^77

※ Even if we achieve a dramatic speed-up in hardware, we still cannot overcome the handicap of an asymptotically slow program.

Data Structures
Chapter 2

1. **Recurrence Relations**
   - Recurrence Equations
   - Folding
   - Telescoping
2. Discrete Math
3. Structures