



본 PSet 은 저의 강의 경험과 학생들의 의견 및 Stanford CS106 과 Harvard CS50 같은 강의에서 수집된 자료를 토대로 작성되었습니다.
본 PSet 에 문제점이나 질문 혹은 의견이 있다면, 저의 이메일(idebtor@gmail.com)로 알려 주시면 강의 개선에 많은 도움이 되겠습니다.

Note: Your homework is listed at the last three pages of this doc.

Recurrence Relations

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.

There are various methods for solving recurrence relations.

Here we will use a method called **"telescoping"**. The method consists in the following:

- Re-write the recurrence relation for subsequent smaller values of n so that the left side of the re-written relation is equal to the first term of the right side of the previous relation. Re-write the relation until a relation involving the initial condition is obtained.
- Add the left sides and the right sides of the relations. Cancel the equal terms on the left and the right side.
- Add the remaining terms on the right side. The sum will give the general formula of the sequence.

The standard method for solving recurrence relations, called **"unfolding"**, makes repeated substitutions applying the recursive rule until the base case is reached.

Useful formulas:

$$1 + 2 + 3 + \dots + N = N(N+1)/2$$

$$1 + 2 + 4 + 8 + \dots + 2^n = 2^{n+1} - 1$$

Example 1:

We may describe that the time complexity of the linear search is

$$T(0) = c_0 \quad // \text{ the time to process an array of 1 element is 1}$$

$$T(n) = T(n-1) + c \quad // \text{ to process } n \text{ items is to process } (n-1) \text{ elements} + 1 \text{ element}$$

The cost of searching n elements is the cost of looking at 1 element, plus the cost of searching $n-1$ elements. We may express the constants c_0 and c as 1, respectively, for our purpose. The expressions become:

$$T(0) = 1$$

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

Telescoping:

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

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

$$T(n-2) = T(n-3) + 1$$

...

$$T(1) = T(0) + 1$$

Add the left and right sides, respectively:

$$T(n) + T(n-1) + T(n-2) + \dots + T(1) = T(n-1) + T(n-2) + \dots + T(0) + 1 + 1 + \dots + 1$$

Cancel equal terms on both sides:

$$T(n) = T(0) + 1 + 1 + \dots + 1$$

How many n's are there?

$$T(n) = T(0) + n$$

$$= 1 + n$$

Therefore, $T(n)$ is $O(n)$.

Unfolding:

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

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

$$= T(n-3) + 3$$

...

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

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

$$T(n) = O(n)$$

Example 2:

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: $10 = 6 + 4$. We want to obtain a formula for the general term of this sequence.

$$a_1 = 1$$

$$a_n = a_{n-1} + n, n = 2, 3, 4, \dots$$

Telescoping:

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

$$a_{n-1} = a_{n-2} + (n-1)$$

$$a_{n-2} = a_{n-3} + (n-2)$$

....

$$a_4 = a_3 + 4$$

$$a_3 = a_2 + 3$$

$$a_2 = a_1 + 2$$

Add the left and right sides:

$$a_n + a_{n-1} + a_{n-2} + a_{n-3} + \dots + a_4 + a_3 + a_2 =$$

$$a_{n-1} + a_{n-2} + a_{n-3} + \dots + a_4 + a_3 + a_2 + a_1 + 2 + 3 + 4 + \dots + (n-1) + n$$

Cancel equal terms on both sides:

$$a_n = a_1 + 2 + 3 + 4 + \dots + (n-1) + n$$

Replace with the value given in the initial condition:

$$a_n = 1 + 2 + 3 + 4 + \dots + (n-1) + n = n(n+1)/2 \quad (\text{open form})$$

Thus we have obtained the general formula for each term in the sequence:

$$a_n = n(n+1)/2 \quad (\text{closed form})$$

Example 3: findmax()

Find recursively the max element in an array.

Algorithm:

If the array has 1 element,

this is the maximum

Else,

- a. Find the maximum in the array of the first n-1 elements.
- b. Compare with the n-th element and return the greater one

```
int findmax(int [] array, int size) {
    if (size == 1)
        return array[0];
    else {
        int max = findmax(array, size-1)
        if (max >= array[size-1])
            return max;
        else
            return array[size-1];
    }
}
```

Here we reason in the following way:

- The time to find the maximum in an array of size N is equal to the time to find the maximum in an array of size N-1 plus the time to do the comparison.
- Since the time to do the comparison does not depend on the size of the array, we consider it to be a constant and we use 1.
- When the array has only one element the time is a constant (again we use 1)

Thus we come up with the following recurrence relation:

$T(1) = 1$ // the time to process an array of 1 element is 1

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

Telescoping:

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

$T(N-1) = T(N-2) + 1$

$T(N-2) = T(N-3) + 1$

....

$T(3) = T(2) + 1$

$T(2) = T(1) + 1$

$T(1) = 1$

Adding the left and the right side and canceling equal terms, we obtain:

$T(N) = 1 + 1 + \dots + 1$ (Open form)

There are N lines in the telescoping, thus the sum of the 1s is

$T(N) = N$ (Closed form)

Therefore, $T(N) = O(n)$

Notice: Your homework begins here. Submit the image captures of the following pages.

Problem 1 - insertion sort

Recurrence relation: The time to sort an array of N elements is equal to the time to sort an array of $N-1$ elements plus $N-1$ comparisons. Initial condition: the time to sort an array of 1 element is constant:

$$T(1) = 1$$

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

Next we perform **telescoping**: re-writing the recurrence relation for $N-1, N-2, \dots, 2$

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

$$T(N-1) = T(N-2) + N-2$$

$$T(N-2) = T(N-3) + N-3$$

$$T(2) = T(1) + 1$$

Next we **sum up the left and the right sides** of the equations above:

$$T(N) + T(N-1) + T(N-2) + T(N-3) + \dots + T(3) + T(2) = T(N-1) + T(N-2) + T(N-3) + \dots + T(2) + T(1) + N-1 + N-2 + N-3 + \dots + 3 + 2 + 1$$

Finally, we cross the equal terms on the opposite sides and **simplify** the remaining sum on the right side:

$$T(N) = T(1) + N-1 + N-2 + \dots + 2 + 1 \quad (\text{Open form})$$

$$T(N) = 1 + \frac{1}{2}N^2 - \frac{1}{2}N \quad \left(\because \sum_{k=2}^N (k-1) = \frac{N(N+1)}{2} - N \right) \quad (\text{Closed form})$$

Therefore, the running time of insertion sort is:

$$T(N) = O(N^2) \quad (\text{big O})$$

$$\begin{aligned} \sum_{k=1}^N (k-1) &= \frac{N(N+1)}{2} - N \\ &= \frac{1}{2}N^2 + \frac{1}{2}N - N \\ &= \frac{1}{2}(N^2 - N) + 1 \end{aligned}$$

Problem 2

$$T(1) = 1$$

$$T(N) = T(N-1) + 2 \quad // 2 \text{ is a constant like } c$$

Telescoping:

$$T(N) = T(N-1) + 2$$

$$T(N-1) = T(N-2) + 2$$

$$T(N-2) = T(N-3) + 2$$

$$T(2) = T(1) + 2$$

Next we sum up the left and the right sides of the equations above:

$$T(N) + T(N-1) + T(N-2) + T(N-3) + \dots + T(2) = T(N-1) + T(N-2) + T(N-3) + \dots + T(2) + T(1) + 2 + 2 + \dots + 2 + 2$$

Finally, we cross the equal terms on the opposite sides and simplify the remaining sum on the right side:

$$T(N) = T(1) + \underline{2 + 2 + \dots + 2 + 2} \quad (\text{open form})$$

$$T(N) = 1 + \underline{2(N-1)} \quad (\text{closed form})$$

Therefore, the running time of reversing a queue is:

$$T(N) = \underline{O(N)} \quad (\text{Big O})$$

Problem 3 - Power()

```
long power(long x, long n) {
    if (n == 0)
        return 1;
    else
        return x * power(x, n-1);
}
```

$$\begin{aligned} T(0) &= 1 \\ T(1) &= T(0) + 1 \\ T(2) &= T(1) + 1 \\ &\vdots \\ T(n) &= T(n-1) + 1 \end{aligned}$$

$T(n)$ = Time required to solve a problem of size n

Recurrence relations are used to determine the running time of recursive programs—recurrence relations themselves are recursive

$T(0)$ = time to solve problem of size 0

– Base Case

$T(n)$ = time to solve problem of size n

– Recursive Case

$$T(0) = 1$$

$$T(n) = T(n-1) + 1 \quad // \text{+1 is a constant}$$

Solution by telescoping:

If we knew $T(n-1)$, we could solve $T(n)$.

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

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

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

....

$$T(2) = \underline{T(1) + 1}$$

$$T(1) = \underline{T(0) + 1}$$

Next we sum up the **left and the right sides** of the equations above:

$$\begin{aligned} T(n) + \cancel{T(n-1)} + \cancel{T(n-2)} + \dots + \cancel{T(2)} + \cancel{T(1)} &= \\ \cancel{T(n-1)} + \cancel{T(n-2)} + \dots + \cancel{T(2)} + \cancel{T(1)} + T(0) + 1 + 1 + \dots + 1 + 1 &= \\ \Rightarrow T(n) = T(0) + \underbrace{1 + 1 + \dots + 1}_{n \text{ th}} + 1 &= \end{aligned}$$

Finally, we cross the equal terms on the opposite sides and simplify the remaining sum on the right side:

$$T(n) = \underline{T(0) + 1 + 1 + \dots + 1 + 1} \quad (\text{Open form})$$

$$T(n) = \underline{1 + n} \quad (\text{Closed form})$$

$$T(n) = \underline{O(n)} \quad (\text{Big O})$$

Problem 4 - Power()

```

long power(long x, long n) {
    if (n == 0) return 1;
    if (n == 1) return x;
    if ((n % 2) == 0)
        return power(x * x, n/2);
    else
        return power(x * x, n/2) * x;
}

```

$$T(0) = 1$$

$$T(1) = 1$$

$$T(2) = T(2/2) + 1$$

$$T(3) = T(3/2) + 1$$

$$T(4) = T(4/2) + 1$$

$$\vdots$$

$$T(n) = T(n/2) + 1$$

$$T(0) = 1$$

$$T(1) = 1$$

$$T(n) = T(n/2) + 1 \quad // \text{ Assume } n \text{ is power of 2, } +1 \text{ is a constant}$$

Solution by unfolding:

$$T(0) = 1$$

$$T(1) = 1$$

$$T(n) = T(n/2) + 1$$

$$T(n/2) = T(n/4) + 1$$

$$T(n/4) = T(n/8) + 1$$

$$T(n/8) = T(n/16) + 1$$

$$\text{since } T(n/2) = T(n/4) + 1$$

$$\text{since } T(n/4) = T(n/8) + 1$$

$$T(n/2^k) = \dots = T(n/2^k) + 1$$

$$\text{in terms of } n, 2^k, k$$

$$(T(2) = T(1) + 1)$$

$$T(1) = T(n/2^k)$$

$$\Rightarrow \frac{n}{2^k} = 1 \sim n = 2^k$$

$$\sim \log_2 n = k //$$

We want to get rid of $T(n/2^k)$.

We solve directly when we reach $T(1)$

$$n/2^k = 1$$

$$n = 2^k$$

$$\log n = k$$

$$T(n) = \frac{T(n/2^k) + 1 + \dots + 1}{\text{(Open form)}}$$

$$\text{in terms of } n, 2^k, k$$

$$= \frac{T(1) + 1 + 1 + \dots + 1}{\text{(Open form)}}$$

$$= \frac{1 + \log_2 n}{\text{(Closed form)}}$$

$$\text{Therefore, } T(n) = O(\log_2(n)) \quad (\text{Big O})$$

Files to submit

Submit image captures of the last three pages of this file on Piazza folder.

Due

11:55 PM

One thing I know, I was blind but now I see. John 9:25