Notice: 여기서부터 과제가 시작된다. 다음의 페이지들을 스크린 캡처하여 제출한다.

Problem 1 – 삽입 정렬

점화 관계: N 개의 원소로 이루어진 배열을 정렬하는데 걸리는 시간은 N – 1 개의 원소를 가진 배열을 정렬하는데 걸리는 시간과, N – 1 번의 비교를 하는데 걸리는 시간의 합이다. 초기 조건: 1 개의 원소를 가진 배열을 정렬하는데 걸리는 시간은 상수 1 이다.

T(1) = 1

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

Next **telescoping** 을 해봅시다: 다음 조건들을 위해 새로운 점화 관계식을 작성한다: N-1, N-2, …, 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 위에서 구한 식의 왼편과 우편을 더한다:

T(N) + T(N-1) + T(N-2) + T(N-3) + …. T(3) + T(2) =
T(N-1) + T(N-2) + T(N-3) + …. T(3) + T(2) + T(1) + (N-1)+(N-2)+(N-3)+ … + 3+2+1

마지막으로, 같은 항들을 제거하고 우편에 합을 간소하게 만든다:

T(N) = T(1) + (N-1)+(N-2)+(N-3)+ … +3+2+1 (Open form)

T(N) = 1 + $\frac{N(N-1)}{2}$ (Closed form)

그리하여서, 삽입 정렬의 시간 복잡도는:

T(N) = O(N)^2 (big O)

Problem 2

T(1) = 1

T(N) = T(N – 1) + 2 // 2 는 c 와 같은 상수이다

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 위에서 구한 식의 왼편과 우편을 더한다:

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

마지막으로, 같은 항들을 제거하고 우편에 합을 간소하게 만든다:

T(N) = T(1) + 2+2+2+ ··· +2+2 (open form)

T(N) = 1 + 2N (closed form)

그리하여서, 큐 (queue)를 뒤집는 시간 복잡도는:

T(N) = O(N) (Big O)

Problem 3 - Power()

```
long power(long x,long n) {
  if(n==0)
    return1;
  else
    return x * power(x,n-1);
}
```

T(n) = n 크기의 문제를 해결하는데 걸리는 시간.
점화 관계는 재귀 프로그램들의 시간 복잡도를 구하는데 사용된다 – 점화 관계 그들 자신도 재귀다.

T(0) = 크기가 0 인 문제를 해결하는데 걸리는 시간.

- 초기 조건 T(0) = 1

T(n) = 크기가 n 인 문제를 해결하는데 걸리는 시간.

- 재귀 조건 T(n) = n+1

T(0) = 1

T(n) = T(n – 1) + 1 // +1 은 상수다.

Telescoping 을 사용한 풀이:

T(n – 1)을 알면, T(n)을 풀 수 있다.

T(n) = T(n – 1) + 1

T(n – 1) = T(n-2) + 1

T(n – 2) = T(n-3) + 1

….

T(2) = T(1) + 1

T(1) = T(0) + 1

Next 위에서 구한 식의 왼편과 우편을 더한다:

T(n) + T(n-1) + T(n-2) + ·· + T(2) + T(1)

= T(n-1) + T(n-2) + ··· + T(2) + T(1) + T(0) + 1+1+1+1 · +1+1

마지막으로, 같은 항들을 제거하고 우편에 합을 간소하게 만든다:

T(n) = T(0)+1+1+ ·· +1+1 (Open form)

T(n) = 1+ n (Closed form)

T(n) = O(n) (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(n) = T(n / 2) + 1 // n 은 2 의 제곱 수이고, +1 은 상수다.

Unfolding 을 사용한 풀이:

T(0) = 1

T(1) = 1

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

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

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

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

....

= $T(n/2^k)$ + k n, 2^k, k 으로 표현.

$T(n/2^k)$ 을 제거 하기를 원한다.

T(1) 에 도달하였을 때 해결을 한다.

$n/2^k = 1$

$n = 2^k$

log n = k

T(n) = T(0) + log(n) (Open form) n, 2^k, k 으로 표현.

= T(1) + log(n) (Open form)

= log (n) + 1 (Closed form)

그리하여서, T(n) = O(log n) (Big O)

제출할 파일들

Piazza 폴더에 현재 파일의 마지막 3 페이지의 스크린 캡쳐를 제출한다.

제출 마감

11:55 PM

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