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, …, 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) + …. 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

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) + … + 2 + 1 (Open form)

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

Therefore, the running time of insertion sort is:

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

Problem 2

T(1) = 1

T(N) = T(N – 1) + 2 // 2 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) + … + T(3) + T(2) =
T(N-1) + T(N-2) + T(N-3) + … + T(3) + T(2) + T(1) + 2 + 2 + … + 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) + 2+2+ ··· +2+2 (open form)

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

Therefore, the running time of reversing a queue is:

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

T(n – 2) = T(n-3)+1

….

T(2) = T(1) +1

T(1) = T(0) +1

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

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

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

T(n) = T(0)+1+1+··+1+1 (Open form)

T(n) = N+1 (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 // Assume n is power of 2, +1 is a constant

Solution by unfolding:

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

....

= 1k + T(n/2^k) in terms of n, 2^k, k

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

We solve directly when we reach T(1)

$n/2^k = 1$

$n = 2^k$ $\log n = \log 2^k = k \log 2 = k$

log n = k

T(n) = k + T(1) (Open form) in terms of n, 2^k, k

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

= log n + 1 (Closed form)

Therefore, T(n) = O(log n) (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