Homework 1 Solutions
CSI 503 Spring 2021

Rahul Bhenjalia

February 19, 2021

1. First, determine the asymptotically tight bounds for each of the following running time functions. Then, arrange them in non-decreasing order of their asymptotic complexity. Explain your answer.

   - $f1(n) = 2^{log(n)}$: Since the $log(n)$ is an exponent the resulting time complexity is $O(2^{log(n)})$
   - $f2(n) = 5n^2 + n^{2.5}$: Since $2.5 > 2$, the resulting time complexity is $O(n^2.5)$
   - $f3(n) = nlog(n^2)$: The give equation can be further solved into $2nlog(n)$, which gives us $O(nlog(n))$
   - $f4(n) = \sqrt{10n}$: Time Complexity : $O(\sqrt{n})$
   - $f5(n) = n(log_3(n) + 10^{10000})$: Even with having astronomically high constant with n, $nlog_3(n)$ has higher preference, hence time complexity is $O(nlog_3(n))$
   - The non-decreasing order of their asymptotic complexity are as follows:

     O$(2^{log(n)}) < O(\sqrt{n}) < O(nlog(n)) < O(nlog_3(n)) < O(n^2.5)$

.
.
.
.
.
.
.
.
.
.
.
.
.

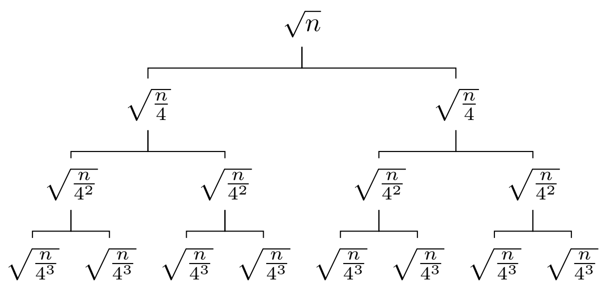
2. Show that the following algorithm correctly reverses the input string by defining a loop invariant and clearly justifying the correctness based on its initialization, maintenance, and termination. Here, S is an array of characters of length n.

```
function STRREV(S, n)
 for i = 1 to floor(n/2) do
   c = S[i]
   S[i] = S[n - i + 1]
   S[n - i + 1] = c
```

The Loop Invariant is used to help understand the correctness of the give algorithm. In the given case, it is a string reversal algorithm with inputs as a string array e.g [h,e,l,l,o] and the length of array n = 5.
.
Initialization:
The algorithm should be true prior to first iteration which in our example case is a string array [h,e,l,l,o] and length n=5, which we consider to be true and Initialization holds.
.
Maintenance:
Let us consider the first iteration after which the resulting string array would [o,e,l,l,h] which is true before the iteration [h,e,l,l,o] and after the iteration [o,e,l,l,h], since no new character were introduced in the array and the algorithm continues it objective. Hence, Maintenance holds.
.
Termination:
The termination condition states that when the loop ends, the invariant gives us something useful which shows us that algorithm is correct.
In our example case, these would be states of the string array until the loop terminates at i equals 2
. [h,e,l,l,o](i at 1)
. [o,e,l,l,h](i at 2)
. [o,l,l,e,h](i at 3) (Terminates here since $3 < 2$)
.
Since, we have our objective reached, which is getting a reversed string, hence, termination holds.
Since, the loop variant hold on every level, we can say that the given algorithm is correct.
.
.
.
.
.
.

3. For the following recurrence, establish a guess about its asymptotically tight bound by drawing a recursion tree. (See Figure 2.5d in the textbook for an example.) Then, use the substitution method to justify both its lower and upper bounds.

   $T(n) = 2T(n/4) + \sqrt{n}$

   
   $\sqrt{n}$
$\sqrt{\frac{n}{4}}$ $\sqrt{\frac{n}{4}}$
$\sqrt{\frac{n}{4^2}}$ $\sqrt{\frac{n}{4^2}}$ $\sqrt{\frac{n}{4^2}}$ $\sqrt{\frac{n}{4^2}}$
$\sqrt{\frac{n}{4^3}}$ $\sqrt{\frac{n}{4^3}}$ $\sqrt{\frac{n}{4^3}}$ $\sqrt{\frac{n}{4^3}}$ $\sqrt{\frac{n}{4^3}}$ $\sqrt{\frac{n}{4^3}}$ $\sqrt{\frac{n}{4^3}}$ $\sqrt{\frac{n}{4^3}}$

   Let us consider a base case where algorithm has to stop: T(1) : 4

   .

   Considering the pattern in the tree levels, we can generate a general term which would work at each level: $\sqrt{(\frac{n}{4^i})}$

   .

   Let say that we reach actual base case where n :: 1

   .

   $1 = \sqrt{(\frac{n}{4^i})}$

   .

   $\sqrt{(4^i)} = \sqrt{(n)}$

   .

   $2^i = \sqrt{(n)}$

   Taking log base 2 both sides, for value of i:

   .

   $i = \frac{log(n)}{2}$

   .

   Since, the sum of nodes at each level is $\sqrt{(n)}$, the time taken will be the sum of all costs up until we reach base case.

   $\sum_{i=0}^{\frac{log(n)}{2}} \sqrt{(n)}$

   .

   $\sqrt{(n)} \sum_{i=0}^{\frac{log(n)}{2}} (1)$

   .

   $\sqrt{(n)}(\frac{log(n)}{2} + 1)$

   .

   $\sqrt{(n)}\frac{log(n)}{2} + \sqrt{(n)}$

   .

   Hence, given the time function we just got above, we can assume that tight bound asymptotic complexity of the recursion equation would be $\theta(\sqrt{(n)}log_2 n)$

   .

4. Derive asymptotic upper and lower bounds for T(n) in each of the following cases. Clearly justify your answer even when you use the master theorem.

   Master's Theorem

   .

   Let's say we have an the expressing in terms of the following part:

   .

   $T(n) = aT(\frac{n}{b}) + f(n)$

   .

   where $f(n) = n^k log^p(n)$

   .

   Considering above format, we make three cases on Master's theorem:

   .

   CASE:1 where $log_b(a) > k : \theta(n^{log_b a})$

   .

   CASE:2 where $log_b(a) equalsk$
   ....Sub-Case1: if $p > -1 : \theta(n^k log^{p+1}(n))$
   ....Sub-Case2: if $pequals - 1 : \theta(n^k log(log(n)))$
   ....Sub-Case3: if $p < -1 : \theta(n^k)$

   .

   CASE:3 where $log_b(a) < k$
   ....Sub-Case1: if $p > 0orp : 0 : \theta(n^k log^p(n))$
   ....Sub-Case2: if $p < 0 : \theta(n^k)$

   (a) $T(n) = 4T(n/3) + nlog(n)$

   .

   Applying the above properties of the Master's theorem, we have a:4, b:3 and k:1, p:1

   .

   Since, $log_3 4 > 1$, we apply CASE:1 of the theorem:

   .

   Which gives us $\theta(n^{log_3 4})$

   .

   (b) $T(n) = T(n/3) + T(n/6) + n$

   For this instance we can break down the given time equation in two parts:
   $T(n) = T(n/3) + C(n)$ AND $C(n) = T(n/6) + n$

   .

   Hence, we can solve C(n) first using master's theorem, which results in $O(n)$

   .

   Which we can basically incorporate as n in the original $T(n) = T(n/3) + n$

   .

   which ultimately results, in the final time complexity as $O(n)$

   .

   .

(c) $T(n) = 5T(n/2) + n^3$

.

Applying the above properties of the Master's theorem, we have a:5, b:2 and k:3, p:0

.

Since, $log_2 5 < 3$, we apply CASE:3 of the theorem:

.

And p:0, which activates Sub-Case:1

.

Which gives us $\theta(n^3)$

.

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

For this we cannot use master's theorem, since there is a arithmetic decrease happening with n, in every iteration.

.

Hence, assuming the decrease, we can assume that the algorithm will end at some point.
We can break it down using substituion:
$T(n) = T(n-2) + n^2$: Substituting $n-2$ to n

.

$T(n) = T(n-4) + (n-2)^2 + n^2$: Keep Substituting..

.

$T(n) = T(n-6) + (n-4)^2 + (n-2)^2 + n^2$:

.

If we keep substituting we are gonna reach end point at some point.

.

Depending upon what kind value we start with even n or odd n, there will be two cases:

.

If n is even, we get $T(n) = T(0) + ... + (n-4)^2 + (n-2)^2 + n^2$:

.

Calculating sum of all even number squared we have: $T(n) = \frac{2n(n+1)(2n+1)}{3}$

.

Hence, the time complexity would be $O(n^3)$

.

If n is odd, we get $T(n) = T(1) + ... + (n-4)^2 + (n-2)^2 + n^2$:

.

Calculating sum of all odd number squared we have:
$T(n) = 1 + \frac{n(2n+1)(2n-1)}{3}$

.

Hence, the time complexity would be $O(n^3)$

.

So, it doesn't matter, either the n is odd or even, in both cases the complexity will be $O(n^3)$