Homework Assignment #3

CMPUT 204
Department of Computing Science
University of Alberta

Due: Wednesday, Feb. 9, 2022 (11:59pm)

(Total 40 marks)

In all problems, we assume $T(n) = O(1)$ for small n.

Problem 1. (20 marks)

- Use iterated substitution to guess a good solution (a tight asymptotic upper bound) to the recurrence $T(n) = T(n-10)+n$ and prove your guess is indeed a solution. What about $T(n) = T(n-100)+100n$? No justification is required (hope you can answer this question immediately with confidence).

  Solution: Assuming that $T(n) = 0$ for small values of n, by iterative substitution, we get that

  $$T(n) = \sum_{i=1}^{n/10} 10i = \frac{n}{20}(10+n) = \frac{n^2+10n}{20} \Rightarrow T(n) \in O(n^2)$$

  To verify, we can show that $T(n) \leq cn^2$ for all $n \geq n_0$, where c and n_0 are positive constants.

  $$\begin{aligned} T(n) &= T(n-10) + n \\ &\leq c(n-10)^2 + n \\ &= cn^2 - 20cn + 100c + n \\ &\leq cn^2 \end{aligned}$$

  The last step holds if $-20cn + 100c + n \leq 0$. We can pick $c \geq 1$ and $n \geq 100c/(20c-1)$.

- Show the solution to $T(n) = 2T(\lfloor n/2 \rfloor + 8) + n$ is $O(n \log n)$.

  Solution: Expected solution method: CLRS 4.3-6.

  Assume that $T(n) \leq c(n-16)\log(n-16)$ for all $n \geq n_0$, where c and n_0 are positive constants.

  $$\begin{aligned} T(n) &= 2T(\lfloor n/2 \rfloor + 8) + n \\ &\leq 2c(n/2-8)\log(n/2-8) + n \\ &= c(n-16)\log(n-16) - c(n-16) + n \\ &\leq c(n-16)\log(n-16) \end{aligned}$$

  The last step holds if $-c(n-16) + n \leq 0$. We can pick $c \geq 1$ and $n \geq 16c/(c-1)$.

- Given the recurrence $T(n) = T(n/3) + T(2n/3) + 3n$, use the recurrence tree method to guess a tight upper bound and a tight lower bound and prove that your guesses are correct (to simplify the question, for this latter part, you only need to present a proof either for the upper bound or for the lower bound).

  Solution: Expected solution method: CLRS 4.4-6.

  By constructing the recursion tree, it can be seen that the tight upper bound is $O(n \log n)$ (branch generated by $T(2n/3)$) and the tight lower bound is $\Omega(n \log n)$ (branch generated by $T(n/3)$)

To verify the upper bound (lower bound is symmetrical), prove that $T(n) \leq cn\log(n)$ for all $n \geq n_0$ for some n_0 and c:

$$\begin{aligned}
T(n) &= T(n/3) + T(2n/3) + 3n \\
&\leq c(n/3)\log(n/3) + c(2n/3)\log(2n/3) + 3n \\
&= cn\log(n) + cn((1/3)\log(1/3) + (2/3)\log(2/3)) + 3n \\
&= cn\log(n) + (cn/3)\log(4/27) + 3n \\
&\leq cn\log(n)
\end{aligned}$$

The last step holds if $(cn/3)\log(4/27) + 3n \leq 0$. We can pick $n \geq 0$ and $c \leq 9/\log(27/4)$.

- Solve the recurrence $T(n) = 2T(\sqrt{n}) + \log\log n$. Hint: You may consult any solutions you can find online for CLRS 4.3-9.

  Solution: Expected solution method: CLRS 4.3-9

  Let $n = 2^m$, then $T(2^m) = 2T(2^{m/2}) + \log m$. With the substitution $T(2^m) = S(m)$, we get that $S(m) = 2S(m/2) + \log m$. Solving with Master Theorem, $S(m) \in \Theta(m)$ and therefore, $T(n) \in \Theta(\log n)$.

- Solve the recurrence $T(n) = 2T(n-2) + 1$. You can use any method to guess a tight solution (you only need to sketch how you guessed) and then prove your guess is correct.

  Solution: Assuming that $T(1)$ is a constant, say 1. By iterative substitution, we get that

$$T(n) = \sum_{i=0}^{n/2-1} 2^i = \frac{2^{n/2} - 1}{2 - 1} = 2^{n/2} - 1 \Rightarrow T(n) \in \Theta(2^{n/2})$$

  To verify we can show by induction that $T(n) \leq 2^{n/2} - 1$. For the induction step we will have:

$$\begin{aligned}
T(n) &= 2T(n-2) + 1 \\
&\leq 2(2^{(n-2)/2} - 1) + 1 \\
&= 2^{n/2} - 1
\end{aligned}$$

Problem 2. (8 marks) Apply the master method to solve the following recurrences.

a) $T(n) = 8T(n/4) + n^2$.

   Solution: $\Theta(n^2)$ (3rd case of M.T.)

b) $T(n) = 5T(n/9) + \sqrt{n}$.

   Solution: $\Theta(n^{log_9(5)})$ (1st case of M.T.)

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

   Solution: $\Theta(n^{1/2})$ (1st case of M.T.)

d) $T(n) = 9T(n/8) + n^2 + n$.

   Solution: $\Theta(n^2)$ (3rd case of M.T.)

Problem 3. (12 marks) Consider the following very simple and elegant(?) sorting algorithm:

SomeSort (A, b, e)
if $e = b + 1$ **then**
  if $A[b] > A[e]$ **then**
    exchange $A[b]$ and $A[e]$
  end if
else if $e > b + 1$ **then**
  $p \longleftarrow \lfloor \frac{e-b+1}{3} \rfloor$
  SomeSort $(A, b, e - p)$
  SomeSort $(A, b + p, e)$
  SomeSort $(A, b, e - p)$
end if

a. Does SomeSort correctly sort its input array A (assuming that $n = e-b+1$ is the length of the array)? Justify your answer.

Solution: Yes. Let $n = e - b + 1$ be the number of elements in the array A. We use an inductive argument to show that for any value of n this algorithm sorts array A. The claim is obvious for $n \leq 2$. So let's assume that SomeSort works for arrays of size smaller than $n > 2$ and assume that A is an array of size n. To make the arguments easier assume that $n = 3p$. Argue that after the first recursive call the top p largest elements of A that were in locations b to $e - p$ in A is now located between $e - 2p$ to $e - p$ in A. Therefore, after the 2nd recursive call, all the top p largest elements of A are in the last p positions. Thus the last call to SomeSort puts the rest of the numbers into their correct locations between b and $e - p$, by induction hypothesis.

b. Consider the input array A: 8 1 4 9 7 3 2 6 5. List five valid states of the array during the execution of the algorithm.

Solution: All valid states are listed below.
A: 1 8 4 9 7 3 2 6 5
A: 1 4 8 9 7 3 2 6 5
A: 1 4 8 7 9 3 2 6 5
A: 1 4 7 8 9 3 2 6 5
A: 1 4 7 8 3 9 2 6 5
A: 1 4 7 3 8 9 2 6 5
A: 1 4 3 7 8 9 2 6 5
A: 1 3 4 7 8 9 2 6 5
A: 1 3 4 7 8 2 9 6 5
A: 1 3 4 7 2 8 9 6 5
A: 1 3 4 2 7 8 9 6 5
A: 1 3 4 2 7 8 6 9 5
A: 1 3 4 2 7 6 8 9 5
A: 1 3 4 2 7 6 8 5 9
A: 1 3 4 2 7 6 5 8 9
A: 1 3 4 2 7 5 6 8 9
A: 1 3 4 2 5 7 6 8 9
A: 1 3 4 2 5 6 7 8 9
A: 1 3 2 4 5 6 7 8 9
A: 1 2 3 4 5 6 7 8 9

A: 1 2 3 4 5 6 7 8 9

c. Find a recurrence for the worst-case running time of SomeSort. Give a tight (i.e. Θ) asymptotic bound for the worse-case running time of SomeSort (Hint: consider $n = 3^k$ for some k).

Solution: The running time function $T(n)$ is constant for small values of n and $T(n) = 3T(\lceil \frac{2n}{3} \rceil) + c$ For simplicity, we assume that $n = 3^k$ for some $k > 0$. In this case, $\lceil \frac{2n}{3} \rceil = \frac{2n}{3}$. Thus the recurrence relation can be written as $T(n) = 3T(\frac{2n}{3}) + c$. And now using the master theorem we get $T(n) \in \Theta(n^{\log_{3/2} 3})$.

d. By comparing SomeSort with insertion sort, merge sort, heap-sort, and quicksort argue if this simple algorithm is efficient.

Solution: Since $\log_{3/2} 3 > 2.7$ the running time of SomeSort is $\Omega(n^{2.7})$ which is worse than insertion sort, merge sort, heapsort, and quicksort. So this algorithm is not so elegant!.