University of Nevada, Las Vegas Computer Science 477/677 Fall 2020

Answers to Assignment 3: Due Monday September 14, 2020

Name:_____

Write your answers into a pdf file and email it to the graduate assistent, Miss Chua, on Monday, September 14, before midnight.

1. Work problem 2.4 in your textbook. Suppose you are choosing between the following three algorithms:

   - Algorithm A solves problems by dividing them into five subproblems of half the size, recursively solving each subproblem, and then combining the solutions in constant time.
     The recurrence is $T(n) = 5T(n/2) + 1$
     $T(n) = \Theta(n^{\log_2 5}) = \Theta(5^{\log_2 n})$ by the master theorem.

   - Algorithm B solves problems of size n by recursively solving two subproblems of size $n-1$ and then combining the solutions in constant time.
     The recurrence is $T(n) = 2T(n-1) + 1$
     Let $m = 2^n$, then $m/2 = 2^{n-1}$, and define $G(m) = T(n) = T(\log_2 m)$.
     $G(m/2) = T(\log_2(m/2)) = T(\log_2 m - 1) = T(n-1)$.
     Substituting in the recurrence, we have $G(m) = 2G(m/2) + 1$
     $G(m) = \Theta(m)$ by the master theorem. hence
     $T(n) = \Theta(2^n)$

   - Algorithm C solves problems of size n by dividing them into nine subproblems of size $n/3$, recursively solving each subproblem, and then combining the solutions in $O(n^2)$ time.
     The recurrence is $T(n) = 9T(n/3) + n^2$.
     $T(n) = \Theta(n^2 \log n)$ by the master theorem.

   What are the running times of each of these algorithms (in O notation), and which would you choose?
   C. B is exponential, and $\log_2 5 > 2$. The log factor doesn't count if the exponents of n are different.

2. Work problem 2.5 in your texctbook. Do not replace any transcendental constant with a decimal. For example "$\log_2 3$" should be left as is, but "$\log_2 4$" should be written as 2.
   Parts (a) through (f) are solved using the master theorem. Note that $x^{log_y z} = z^{\log_y x}$

   (a) $T(n) = 2T(n/3) + 1$
   $T(n) = \Theta(n^{\log_3 2)} = \Theta(2^{\log_3 n})$

   (b) $T(n) = 5T(n/4) + n$
   $T(n) = \Theta(n^{\log_4 5)} = \Theta(5^{\log_4 n})$

   (c) $T(n) = 7T(n/7) + n$
   $T(n) = \Theta(n \log n)$

   (d) $T(n) = 9T(n/3) + n^2$
   $T(n) = \Theta(n^2 \log n)$

   (e) $T(n) = 8T(n/2) + n^3$
   $T(n) = \Theta(n^3 \log n)$

(f) $T(n) = 49T(n/25) + n^{3/2}\log n$
$T(n) = \Theta(n^{3/2}\log n)$ because $25^{3/2} = 125 > 49$ The factor of $\log n$ is just carried along.

(g) $T(n) = T(n-1) + 2$
$T(n) = \Theta(2n) = \Theta(n)$ by the anti-derivative method.

(h) $T(n) = T(n-1) + n^c$ where $c \geq 1$ is a constant.
$T(n) = \Theta(n^{c+1})$ by the anti-derivative method.

(i) $T(n) = T(n-1) + c^n$ where $c > 1$ is a constant.
$T(n) = \Theta\left(\frac{c^n}{\ln c}\right) = \Theta(c^n)$ by the anti-derivative method.

(j) $T(n) = 2T(n-1) + 1$
$T(n) = \Theta(2^n)$, just as for algorithm B in the first problem.

(k) $T(n) = T(\sqrt{n}) + 1$ Use substitution: $m = \log_2 n$.
$Let G(m) = G(\log_2 n) = T(n)$. $G(m/2) = G\left(\frac{\log_2 n}{2}\right) = G(\log_2 \sqrt{n}) = T(\sqrt{n})$
$G(m) = T(n) = T(\sqrt{n}) + 1 = G(m/2) + 1$
By the master theorem $T(n) = G(m) = \Theta(\log m) = \Theta(\log\log n)$

3. Work problem 2.12 in your textbook. How many lines, as a function of n, does the following program print? Write a recurrence and solve it. You may assume n is a power of 2.

```
function f(n)
   if n > 1:
      print_line(''still going'')
      f(n/2)
      f(n/2)
```

Let $L(n)$ be the number of lines printed when $r(n)$ executes. The recurrence is then $L(n) = 2L(n/2)+1$. By the master theorem, $L(n) = \Theta(n)$.
However, we want an exact solution. For that, we use the same recurrence, but with the boundary condition $L(1) = 0$. By induction, we can then prove that $L(n) = n - 1$ if n is a power of 2.

4. Walk through the steps of mergesort for the input file L B G S M K U J

Sort pairs:

BL GS KM JU

Merge blocks in pairs:

BGLS JKMU

Merge blocks in pairs:

BGJKLMSU

There is now only one block, Which is the sorted list.

5. Walk through the steps of polyphase mergesort for the input file LBGSMKUJ
I will show you the steps of external polyphase mergeort for that example. The memory need hold at most four items at any time. At most four files need be opened simultaneously. A new write file can be opened. A write file can be closed and then reopened as a read file. The algorithm is allowed to read

only the first item in the file, but cannot "peek" at it without reading it. The algorithm can query the file to determine whether it is empty. nitially, all items are on one file, and at the end, the items are sorted on one file.
File 0: LBGSMKUJ
File 0 is read only, files 1 and 2 are open for writing.
First step of Phase 1: write the first run of file 0 to file 1
File 0: BGSMKUJ
File 1: L
File 2: (empty)
next step: write the first run of file 0 to file 2
File 0: MKUJ
File 1: L
File 2: BGS
next step: write the first run of file 0 to file 1
File 0: KUJ
File 1: LM
File 2: BGS
next step: write the first run of file 0 to file 2
File 0: J
File 1: LMKU
File 2: BGS
next step: write the first run of file 0 to file 1
File 0: (empty)
File 1: LMKU
File 2: BGSJ
Phase 1 ends because file 0 is empty.
Files 1 and 2 become read files. New write files 3 and 4 are opened.
First step of Phase 2: merge the first runs of files 1 and 2, write to file 3
File 1: KU
File 2: J
File 3: BGLMS
File 4: (empty)
next step: merge the first runs of files 1 and 2, write to file 4
File 1: (empty)
File 2: (empty)
File 3: BGLMS
File 4: JKU
Phase 2 ends becaus file 1 and 2 are empty.
Open files 3 and 4 for reading, files 1 and 2 for writing.
First step of phase 3: merge the first runs of files 3 and 4, write to file 1
File 1: BGJKLMSU
File 2: (empty)
File 3: (empty)
File 4: (empty)

Phase 3 ends because files 3 and 4 are empty.
Algorithm ends because there are 3 empty files. The remaining file has the sorted list.

6. What does the following program do? What is the loop invariant of its main loop?

```
int f(int n)
  {
   assert(n >= 0);
   int m = n;
   int d = 0;
   while(m > 0)
     {
      m = m-1;
      d = d+2;
     }
   return d;
  }
```

If we don't see it, we can run the program for various values of n. We will then see that the function always returns 2n.
Note that during each iteration of the loop, m decreases by 1 and d increases by 2. 2m + d decreases by 2 and increases by 2 during each iteration, and thus has same value and the end of each iteration as at the beginning of that iteration. What is that value? Well ... before the first iteration 2m+d = 2m = 2n. Since 2m+d is not changed by an iteration, 2m+d = 2n at the beginning and end of each iteration, and is also true before the first iteration. Thus, 2m+d = 2n is the loop invariant.
We can use the loop invariant to prove correctness, namely that an input of n gives an output of 2n.
Proof: After the last itermation, m = 0, the output is d = 2m+d = 2n.