CMPS 6610 Problem Set 02

Name: Yan Zhu

In this assignment we'll work on applying the methods we've learned to analyze recurrences, and also see their behavior in practice. As with previous assignments, some of of your answers will go in `main.py`. You should feel free to edit this file with your answers; for handwritten work please scan your work and submit a PDF titled `problemset-02.pdf` and push to your github repository.

1. Prove that $\log n! \in \Theta(n \log n)$.

- $\log n! = \log 1 + \log 2 + \log 3 \dots \log n$
- For every $\log k$ $1 \le k \le n$ $\log k < n$.
- Thus, $\log n! \le n \log n$
- By using $\log n$ as the upper bound for all logarithmic term $\log k$, we conclude that the upper bound for $\log n!$ is $n \log n$,
- which proves $\log n! \in \Theta(n \log n)$

2. Derive asymptotic upper bounds for each recurrence below, using a method of your choice.

- $T(n) = 2T(n/6) + 1$.
  - Each of size $n/6$
  - Level 0 : $O(1)$
  - Level 1 = 2 subproblems, each does $O(1)$, total $O(1)$
  - Level 2 = 4 subproblems, each does $O(1)$, total $O(1)$
  - Each level does $O(1)$
  - Tree depth $(\log_6 n)$
  - total work: $T(n) = O(n \log_6^2)$

- $T(n) = 6T(n/4) + n$.
  - Each of size $n/4$
  - Level 0: $O(n)$
  - Level 1 = 6 subproblems, each does $O(n/4)$, total $= O(1.5n)$
  - Level 2 = 36 subproblems, each does $O(n/16)$, total $= O(\frac{36}{16} n) = O(2.25n)$
  - Each level does $O(n)$.
  - Tree depth $(\log_4^n)$
  - total work: $T(n) = O(n \log_4^6)$

- $T(n) = 7T(n/7) + n$.
  - Each of size $n/7$
  - Level 0 : $O(n)$
  - Level 1 = 7 subproblems, each does $O(n/7)$, total $= O(n)$
  - Level 2 = 49 subproblems, each does $O(n/49)$, total $= O(n)$
  - Tree depth $\log_7^n$, total work: $T(n) = O(n \log^n)$

- $T(n) = 9T(n/4) + n^2$.
  - Each of size $= n/4$
  - Level 0 = $O(n^2)$
  - Level 1 = 9 subproblems, each does $O(\frac{n^2}{16})$, total $= O(\frac{9n^2}{16})$
  - Level 2 = 81 subproblems, each does $O(\frac{n^2}{16^2})$ total $= O(\frac{81 \cdot 9^2 n^2}{16^2})$
  - Tree depth $\log_4^n$
  - Total work: $T(n) = O(n^2)$

- $T(n) = 4T(n/2) + n^3$.
  - Each of size $= n/2$
  - Level 0 = $O(n^3)$
  - Level 1 = 4 subproblems, each does $O(\frac{n^3}{8})$, total $= O(\frac{n^3}{2})$
  - Level 2 = 4^2 subproblems, each does $O(\frac{n}{4})$, total $= O(\frac{n^3}{4})$
  - Tree depth $\log_2^n$, Total work : $T(n) = O(n^3)$

- $T(n) = 49T(n/25) + n^{3/2} \log n$.
  - Each of size $= n/25$
  - Level 0 : $O(n^{3/2} \log n)$
  - Level 1 = 49 subproblems, each does $O(\frac{n}{25}^{3/2} \log n)$. total $= O(49 \cdot \frac{n^{3/2}}{25^{3/2}} \log n)$ $\approx O(n^{3/2} \log n)$
  - Level 2 = 49^2 subproblems, each does $O(\frac{n^{3/2}}{25^3} \log n)$ total $= O(49^2 \frac{n^{3/2}}{25^3} \log n) \approx O(n^{3/2} \log n)$
  - Tree depth $\log_{25}^n$, total work : $T(n) = O(\log^{3/2} \log n)$

- $T(n) = T(n-1) + 2$.

Problem3 :

A. $w(n) = 2w(n/5) + O(n^2)$ $\quad w(n) = O(n^2)$ $\quad s(n) = O(n^2)$

B. $w(n) = w(n-1) + O(\log n)$ $\quad w(n) = O(n \log n)$ $\quad s(n) = O(n \log n)$

C. $w(n) = w(n/3) + w(n/2) + O(n^{1.1})$ $\quad w(n) = O(n^{1.1})$ $\quad s(n) = O(n^{1.1})$

I will select B, because the s(n) of B is $n \log n$, it is the most efficent in term of span and work.

Problem 4:

A. $w(n) = 5w(n/2) + O(n)$ $\quad w(n) = O(n^{2.33})$ $\quad s(n) = O(n)$

B. $w(n) = 2w(n-1) + O(1)$ $\quad w(n) = O(2^n)$ $\quad s(n) = O(n)$

C. $w(n) = 9w(n/3) + O(n^2)$ $\quad w(n) = O(n^2 \log^n)$ $\quad s(n) = O(\log n)$

I will select C, because it has a lowest work complexity $O(n^2 \log^n)$ and span.

- Each of size : $(n-1)$
- Level 0 : $O(2)$
  Level 1 : $O(2)$
- Level 2 : $O(2)$
- Tree depth $= O(n)$ Total works $= O(n)$

- $T(n) = T(n-1) + n^c$, with $c \geq 1$.
  - Each of size : $(n-1)$
    Level 0 : n^c
  - Level 1 : $O((n-1)^c) \approx O(n^c)$
  - Level 2 : $O((n-2)^c) \approx O(n^c)$
  - Tree depth $O(n)$, Total works : $O(n) \cdot O(n^c) = O(n^{c+1})$

- $T(n) = T(\sqrt{n}) + 1$.
  - Each of size : $n^{\frac{1}{2}}$
  - Level 0 : $O(1)$
    Level 1 : $O(1)$
  - Level 2 : $O(1)$
  - Tree depth $\log\log n$ Total works $T(n) = O(\log\log n)$

3. Suppose that for a given task you are choosing between the following three algorithms:

   - Algorithm $\mathcal{A}$ solves problems by dividing them into two subproblems of one fifth of the input size, recursively solving each subproblem, and then combining the solutions in quadratic time.
   - Algorithm $\mathcal{B}$ solves problems of size n by recursively one subproblems of size $n-1$ and then combining the solutions in logarithmic time.
   - Algorithm $\mathcal{C}$ solves problems of size n by dividing them into a subproblems of size $n/3$ and a subproblem of size $2n/3$, recursively solving each subproblem, and then combining the solutions in $O(n^{1.1})$ time.

   What is the work and span of these algorithms? For the span, just assume that it is the same as the work to combine solutions. Which algorithm would you choose? Why?

4. Suppose that for a given task you are choosing between the following three algorithms:

   - Algorithm $\mathcal{A}$ solves problems by dividing them into five subproblems of half the size, recursively solving each subproblem, and then combining the solutions in linear time.
   - Algorithm $\mathcal{B}$ solves problems of size n by recursively solving two subproblems of size $n-1$ and then combining the solutions in constant time.
   - Algorithm $\mathcal{C}$ 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.

   What is the work and span of these algorithms? For the span, just assume that it is the same as the work to combine solutions. Which algorithm would you choose? Why?

5. In Module 2 we discussed two algoriths for integer multiplication. The first algorithm was simply a recapitulation of the "grade school" algorithm for integer multiplication, while the second was the Karatsaba-Ofman algorithm. For this problem, you will use the stub functions in `main.py` to implement these two algorithms for integer multiplication. Once you've correctly implemented them, test the empirical running times across a variety of inputs to test whether your code scales in the manner predicted by our analyses of the asymptotic work.