CS3340 Assignment 1

Jin Zhao, 251138547

February 1, 2026

Q1

a) Proof that $A^{n-2} \leq F_n \leq A^n, n \geq 1$

Proof. Proof by **Induction**: Let $A = \frac{1+\sqrt{5}}{2}$, from the equation $A^2 = A + 1$. Also given the Fibonacci number sequence, $F_n = F_{n-1} + F_{n-2}, n > 1; F_1 = 1; F_0 = 0$,we will show for $n \geq 1$, the following statement holds: $A^{n-2} \leq F_n \leq A^n$.

Base case: Let $n = 1$.

$$\begin{aligned} A^{n-2} &\leq F_n \leq A^n \\ A^{1-2} &\leq F_1 \leq A^1 \\ A^{-1} &\leq F_1 \leq A^n \\ \frac{-1+\sqrt{5}}{2} &\leq 1 \leq \frac{1+\sqrt{5}}{2} \end{aligned}$$

So $A^{n-2} \leq F_n \leq A^n$ holds for $n=1.

Let $n = 2$ as well.

$$\begin{aligned} A^{n-2} &\leq F_n \leq A^n \\ A^{2-2} &\leq F_{n-1} + F_{n-2} \leq A^2 \\ A^0 &\leq F_{2-1} + F_{2-2} \leq A^2 \\ 1 &\leq F_1 + F_0 \leq A^2 \\ 1 &\leq 1 + 0 \leq A^2 \\ 1 &\leq F_1 + F_0 \leq A^2 \end{aligned}$$

So $A^{n-2} \leq F_n \leq A^n$ holds for $n = 2$.

Therefore, the statement holds for the base case.

Induction hypothesis: Assume $A^{n-2} \leq F_n \leq A^n$ and $A^{n-3} \leq F_{n-1} \leq A^{n-1}$ for $n \geq 2$, from the base case above. We want to show for $n+1$, $A^{n-2} \leq F_n \leq A^n$ holds, which is to show

$$A^{(n+1)-2} \leq F_{n+1} \leq A^{n+1} A^{n-1} \leq F_{n+1} \leq A^{n+1}$$

that this statement holds.

Inductive step: We want to show $A^{n-1} \leq F_{n+1} \leq A^{n+1}$.

$$F_{n+1} = F_{(n+1)-1} + F_{(n+1)-2}$$
$$F_{n+1} = F_n + F_{n-1}$$

$$A^{n-2} + A^{n-3} \leq F_n + F_{n-1} \leq A^n + A^{n-1}$$
$$A^{n-2} + A^{n-3} \leq F_{n+1} \leq A^n + A^{n-1}$$
$$A^{n-1}(A^{-1} + A^{-2}) \leq F_{n+1} \leq A^n + \frac{A^n}{A}$$
$$A^{n-1}(\frac{1}{A} + \frac{1}{A^2}) \leq F_{n+1} \leq \frac{AA^n + A^n}{A}$$
$$A^{n-1}(\frac{A+1}{A^2}) \leq F_{n+1} \leq \frac{A^{n+1} + A^n}{A}$$
$$A^{n-1}(\frac{A^2}{A^2}) \leq F_{n+1} \leq \frac{A^n(A^1+1)}{A}, \text{ using } A^2 = A + 1$$
$$A^{n-1} \leq F_{n+1} \leq \frac{A^n A^2}{A}$$
$$A^{n-1} \leq F_{n+1} \leq A^{n+1}$$

■.

b) Based on a), the number of bits needed to present F_{50} and F_{500}

Number of bits needed for an integer is calculated by the formula $\log_2$(integer) (from the text). F_{50} and F_{500} themselves are difficult to efficiently calculate by hand, however we can use it's relation to $A = \frac{1+\sqrt{5}}{2}$ and the previously proved inequality, $A^{n-2} \leq F_n \leq A^n$ to give a best estimate. For a conservative estimate, we will alot the necessary bits based on the upper bound, A^n.

For F_{50}:

$$F_{50} \leq A^{50}$$
$$\log_2 F_{50}$$
$$= \log_2 A^{50}$$
$$\approx 34.71209568153087$$
$$\approx 35 \text{ bits}$$

For F_{500}:

$$F_{500} \leq A^{500}$$
$$\log_2 F_{500}$$
$$= \log_2 A^{500}$$
$$\approx 347.12095681530866$$
$$\approx 348 \text{ bits}$$

Need at most 35 bits for F_{50}, and 348 bits for F_{500}.

Q2

a) Show insertion sort step is $\Theta(n/k)$

Theorem: an insertion sort (from inside a merge sort) can sort the n/k, k length sublists with $\Theta(n/k)$ worst-case time.

Proof. We know that the 'divide' step divides the original problem set to $\frac{n}{k}$ sublists, each of length k. In the proposed modified recursive step, an insertion sort is done instead. In the worst case, the insertion sort will need to run $\frac{n}{k}$ times, each time traversing through the entire k length of the sublists.

Given we know insertion sort is $\Theta(n^2)$ (as stated in the question), the recursive step run time becomes $\frac{n}{k}(k^2)$, which is $\Theta\big(\frac{n}{k}(k^2)\big) = \Theta(nk)$. ■.

b) Show how we would merge the n/k sublists in $\Theta(n \log(n/k))$ worst-case time.

Continuing from a), we know from the text that mergesort is $\Theta(n \log n)$, and this is from analysis of its algorithm which involves 3 steps: divide, recurse, and merge. The division step $D(n) = \Theta(1)$, and the recursive step is replaced with the insertion sort in the proposed modification.

The merge step was $\Theta(n)$ originally because merge-sort divides the problem set into n items and then needs to merge those items back together. In the proposed modification, the original problem set is instead divided into n/k sublists of length k, and so we merge n/k items back, in order, instead of n.

For the first level of merges to be done on the (ordered) n/k sublists of length k, the merge operation runs $n/k/2$ times to give the second level of $n/k/2$ sublists of length $2k$, traversing through the entirety of each sublist each time in the worst case, and so on for $\log_2(n/k)$ levels, which is when the final resulting sublist is of length n, and the final ordered array is returned.

Which means that for each $\log_2(n/k)$ level, a $\Theta(n)$ operation is run, thus the modified merge step is $\Theta(n \log(n/k))$.

Therefore, the **merge** algorithm in the **original** mergesort can be applied to the **modified** mergesort. The difference is that it will run on n/k sublists instead of n, which results in the observed discrepancy in worst-case time.

c) k at which $\Theta(n \log n) = \Theta(nk + n \log(n/k))$.

Let $k = \log n$. Then $\Theta(nk + n \log(n/k))$ becomes:

$$\Theta(n \log n + n \log(\frac{n}{(\log n)}))$$

$= \Theta(n \log n + n \log n - \log(\log n))$, where the $\log(\log n)$ term is comparatively small

$$= \Theta(n \log n)$$

The modified and conventional mergesort programs have the same time-complexity when $k = \log n$.

d) practical choice of k.

In practice, I would also consider the space complexity when choosing k.

Q3

A	B	O	o	Ω	ω	Θ
$\lg^k n$	n^ϵ	Yes	Yes	No	No	No
n^k	c^n	Yes	Yes	No	No	No
$\sqrt{n}$	$n^{\sin n}$	No	No	No	No	No
2^n	$2^{n/2}$	No	No	Yes	Yes	No
$n^{\lg c}$	$c^{\lg n}$	Yes	No	Yes	No	Yes
$\lg(n!)$	$\lg(n^n)$	Yes	No	Yes	No	Yes

Q4

For such a program with the recurrence relation:

$$T(2) \leq c$$
$$T(n) \leq T(n/2) + c, n > 2$$

The recurrence tree for it looks like the following: recur tree The runtime based on this tree is thus $c \log_2 n$ which is $\Theta(\log n)$.

Proving time complexity of the recurrence relation is $\Theta(\log n)$

In other words, we will now show that $T(n) \leq c \log_2 n$, $n > 2$

Proof. Proof by **Induction**: Given the recurrence relation, we will show for $n > 2$, $T(n) \leq clog_2 n$.

Base case: Let $n = 2$.

$$T(2) \leq c$$
$$T(2) \leq c \leq c \log_2 2$$

So $T(n) \leq c \log_2 n$ holds for $n = 2$.

Let $n = 4$ as well.

$$\begin{aligned} T(4) &\leq T(4/2) + c \\ T(4) &\leq c + c \\ T(4) &\leq 2c \\ T(4) &\leq 2c \leq c \log_2 4 \end{aligned}$$

So $T(n) \leq c \log_2 n$ holds for $n = 4$.

Therefore, the statement holds for the base case.

Induction hypothesis: Assume $T(n/2) \leq c \log_2 n/2$ for $n \geq 4$. We want to show for every $2n$, $T((2n)/2) \leq c \log_2 (2n)/2$, $T(n) \leq c \log_2 n$.

Inductive step: So we want to show $T((2n)/2) \leq c \log_2 (2n)/2$.

$$\begin{aligned} T(n) &\leq T(n/2) + c \\ T(n) \leq T(n/2) + c &\leq c \log_2 n/2 + c, \text{ from our assumption} \\ T(n) &\leq c(\log_2 n/2 + 1),\ n \geq 4 \\ T(n) &\leq c(\log_2 n/2 + \log_2 2) \\ T(n) &\leq c(\log_2 (n/2)(2)) \\ T(n) &\leq c \log_2 n \end{aligned}$$

Which is $\Theta(\log_n)$.

Therefore, $T(n) \leq c \log_2 n$ and is $\Theta(\log n)$. ■

Q5