Foundations of Algorithms
Homework 3

Arthur Nunes-Harwitt

1. (a) Use Strassen's algorithm to compute the following matrix product.

   $$\begin{pmatrix} 1 & 3 \\ 7 & 5 \end{pmatrix} \begin{pmatrix} 6 & 8 \\ 4 & 2 \end{pmatrix}$$

   Show your work.

   (b) Write pseudocode for Strassen's algorithm.

   (c) How would you modify Strassen's algorithm to multiply $n \times n$ matrices in which n is not an exact power of 2? Show that the resulting algorithm runs in time $\Theta(n^{\lg(7)})$.

   (d) Show how to multiply complex numbers $a+bi$ and $c+di$ using only three multiplications of real numbers. The algorithm should take a, b, c, and d as input and produce the real component $ac-bd$ and the imaginary component $ad+bc$ separately.

2. Consider the recurrence $T(1) = 0$, $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n$.

   (a) Prove that for any $n \in \mathbb{N}$, $\lfloor (n+1)/2 \rfloor = \lceil n/2 \rceil$.

   (b) Prove that for any $n \in \mathbb{N}$, $\lfloor n/2 \rfloor + 1 = \lceil (n+1)/2 \rceil$.

   (c) Let $D(n) = T(n+1) - T(n)$. Prove that $D(1) = 2$, $D(n) = D(\lfloor n/2 \rfloor) + 1$.

   (d) Prove using the strong form of induction that for any $n \in \mathbb{N}$, if $n \geq 1$ then $D(n) = \lfloor \lg n \rfloor + 2$.

   (e) Then prove that $T(n) - T(1) = \sum_{k=1}^{n-1} D(k)$, and show that an immediate consequence is that $T(n) = \sum_{k=1}^{n-1} (\lfloor \lg k \rfloor + 2)$.

   (f) Now show that $T(n) = \sum_{k=1}^{n-1} (\lfloor \lg k \rfloor + 2)$ implies that $T(n) = \mathcal{O}(n \log(n))$.

3. (**project**) Transform the tail-recursive binary search algorithm on arrays involving the two functions *search* and *searchHelp* into a single imperative procedure `search` that performs the search using a while-loop. It should take the same arguments and return the same results as the tail-recursive formulation in the notes.

4. (**project**)

   (a) Write a function `sortedHasSum` that takes a sorted array S of n numbers and another number x, and returns a Boolean indicating whether or not there is a pair of numbers in S whose sum is x that is $\mathcal{O}(n)$. Your implementation may *not* use a hash table (or any auxiliary data structure).

   (b) Write a function `hasSum` that is $\mathcal{O}(n \log(n))$ that does the same thing when S is an arbitrary array of numbers. Your implementation may *not* use a hash table (or any auxiliary data structure).

5. (**project**) Implement `quicksort` so that the size of the stack is $\mathcal{O}(\log n)$ regardless of running time. Hint: Consider the order in which sub-problems are executed in the presence of tail-recursion. Your implementation may *not* modify the partition algorithm.