Foundations of Algorithms
Homework 3

Arthur Nunes-Harwitt

1. (**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 (an array and a data value) and return the same result (a Boolean) as the tail-recursive formulation in the lecture.

2. Consider the following **incorrect** formulation of binary search.

$$\mathit{searchHelp}(a, v; \ell, h) \quad = \quad \begin{cases} \textsc{False} & \text{if } \ell > h \\ \textsc{True} & \text{if } v = a[\hat{m}] \\ \mathit{searchHelp}(a, v; \ell, \hat{m}) & \text{if } v < a[\hat{m}] \\ \mathit{searchHelp}(a, v; \hat{m}, h) & \text{if } v > a[\hat{m}] \end{cases}$$
$$\text{where } \hat{m} = \ell + \lfloor (h - \ell)/2 \rfloor$$

$$\mathit{search}(a, v) \quad = \quad \mathit{searchHelp}(a, v; 0, |a| - 1)$$

   Give a small example of input that causes this formulation to go into an infinite loop; also show the calculation that illustrates that the example leads to an infinite loop.

3. (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 functional pseudo-code for Strassen's algorithm.

   (c) Strassen's algorithm computes $C_{2,1}$ using the formula $C_{2,1} = P_3 + P_4$. Verify that $C_{2,1} = A_{2,1}B_{1,1} + A_{2,2}B_{2,1}$.

   (d) Strassen's algorithm computes $C_{2,2}$ using the formula $C_{2,2} = P_5 + P_1 - P_3 - P_7$. Verify that $C_{2,2} = A_{2,1}B_{1,2} + A_{2,2}B_{2,2}$.

   (e) When we replaced 8 with 7, we didn't take into account the additional matrix sums and differences. The actual recurrence for Strassen's algorithm is the following.

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 7T(\tfrac{n}{2}) + \tfrac{9}{2}n^2 \end{aligned}$$

   Use iteration to solve this recurrence exactly assuming $n = 2^m$.

   (f) 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 $\Theta(n^{\lg(7)})$.

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

4. 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 $T(n) = \mathcal{O}(n \log(n))$.

5. (**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 $\mathcal{O}(n)$. Note that it is permissible to use one number in S twice. 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).

6. (**project**) Implement imperative `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.