Analysis 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.

$$
\begin{aligned}
searchHelp(a, v; \ell, h) \quad &= \quad \begin{cases} \text{FALSE} & \text{if } \ell > h \\ \text{TRUE} & \text{if } v = a[\hat{m}] \\ searchHelp(a, v; \ell, \hat{m}) & \text{if } v < a[\hat{m}] \\ searchHelp(a, v; \hat{m}, h) & \text{if } v > a[\hat{m}] \end{cases} \\
&\quad\ \ \text{where } \hat{m} = \ell + \lfloor (h - \ell)/2 \rfloor \\
search(a, v) \quad &= \quad searchHelp(a, v; 0, |a| - 1)
\end{aligned}
$$

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.

V = -1

A = { 2, 3, 3, 4)

1. Search(a, -1, 0, 3)
    a. M = 0 + (3-0)/2 = 1.5
    b. Floor(1.5) = 1 = M
    c. A[m] = a[1] = 3
    d. -1 < 3; v < a[m]
    e. SearchHelp(a, v, l, m)
2. Search(a, -1, 0, 1)
    a. L = 0, h = 1
    b. M = 0 + (1-0)/2 = 0.5
    c. Floor(0.5) = 0 = m

    d. A[m] = a[0] = 2
    e. -1 < 2; v < a[m]
    f. searchHelp (a, v, l, m)
3. Search(a, -1, 0, 0)
    a. M = 0 + (0-0)/2 = 0
    b. A[m] = a[0] = 2
    c. -1 < 2; v < a[m]
    d. SearchHelp(a, v, l, m)
4. Search(a, -1, 0, 0)
    a. * Infinite loop. Will never reach a false condition.

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.

1.

1* 6 + 3 * 4	1*8 + 3*2
7*6 + 5*4	7*8 + 5*2

2.

18	14
62	66

(b) Write functional pseudo-code for Strassen's algorithm.

1. STRASSEN(A, B)
2.   n = A.rows
3.   if n == 1
4.   return a[1, 1] * b[1, 1]
5.   let C be a new n × n matrix
6.   A[1, 1] = A[1..n / 2][1..n / 2]

```
A[1, 2] = A[1..n / 2][n / 2 + 1..n]
A[2, 1] = A[n / 2 + 1..n][1..n / 2]
A[2, 2] = A[n / 2 + 1..n][n / 2 + 1..n]
B[1, 1] = B[1..n / 2][1..n / 2]
B[1, 2] = B[1..n / 2][n / 2 + 1..n]
B[2, 1] = B[n / 2 + 1..n][1..n / 2]
B[2, 2] = B[n / 2 + 1..n][n / 2 + 1..n]
S[1] = B[1, 2] - B[2, 2]
S[2] = A[1, 1] + A[1, 2]
S[3] = A[2, 1] + A[2, 2]
S[4] = B[2, 1] - B[1, 1]
S[5] = A[1, 1] + A[2, 2]
S[6] = B[1, 1] + B[2, 2]
S[7] = A[1, 2] - A[2, 2]
S[8] = B[2, 1] + B[2, 2]
S[9] = A[1, 1] - A[2, 1]
S[10] = B[1, 1] + B[1, 2]
P[1] = STRASSEN(A[1, 1], S[1])
P[2] = STRASSEN(S[2], B[2, 2])
P[3] = STRASSEN(S[3], B[1, 1])
P[4] = STRASSEN(A[2, 2], S[4])
P[5] = STRASSEN(S[5], S[6])
P[6] = STRASSEN(S[7], S[8])
P[7] = STRASSEN(S[9], S[10])
C[1..n / 2][1..n / 2] = P[5] + P[4] - P[2] + P[6]
C[1..n / 2][n / 2 + 1..n] = P[1] + P[2]
C[n / 2 + 1..n][1..n / 2] = P[3] + P[4]
C[n / 2 + 1..n][n / 2 + 1..n] = P[5] + P[1] - P[3] - P[7]
return C
```

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

C21 = P3 + P4

C21 = 72 - 10

C21 = 62

C21 = A21B21 + A22B24

= 7 * 6 + 5 * 4

= 42 + 20

= 62

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

C22 = P5 + P1 - P3 - P7

= 66

C22 = A21B12 + A22B22

= 7 * 7 + 5 * 2

= 56 + 10

= 66

(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 $n = 2^m$.

N = 2^n

T(2^n) = T(2^{m-1}) + 9/2 * 2^{2m}

S(m-1) = 7^2* (m - 2) + 7^2* 9/2 * 4^{m-1}

S(m-2) = 7^2* (m - 3) + 7^2* 9/2 * 4^{m-1}

S(1) = 7^m* (s (0)) + Σ7^i* 9/2 * 4^{m-i}

S(m) = 7^m* (s (0)) - 9/2 * 4/3 * (4^k- 7^k)

T(n) = $n^{log_2 7}$ * T(1) - 4/3 ($n^2 - n^{log_2 7}$)

T(1) = 1, n > 0

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

$2^{k-1} < n < 2^k = m$

M < 2n

$O(2n^{log7})$

$O(2n^{log7} * n^{log7})$

$O(n^{log7})$

(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.

(a + b) * (c+ d) = a* c + a*d + b*c + b*d

Multiplication 1 = a*c
Multiplication 2 = b*d
Multiplication 3 = (a + b) * (c + d)

Real:

M1-m2
a*c - b*d

Imaginary =

m3-m1-m2
(a + b) * (c + d) - (a*c - b*d)

4.

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

A. (n+1)/2 = n/2
   a. N = ∞
   b. (∞+1)/2 = ∞/2
   c. ∞ = ∞
   d. As N approaches infinity constants drop and this is true.

B.

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 *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 *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 *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.