CSC236 Winter 2017
Assignment #2: Analysis of Recursive Algorithms - Sample Solutions
Due March 8th, by 9:00 pm

The aim of this assignment is to give you some practice analyzing runtime and proving correctness of recursive algorithms.

For each question, please write up detailed answers carefully. Make sure that you use notation and terminology correctly, and that you explain and justify what you are doing. Marks will be deducted for incorrect or ambiguous use of notation and terminology, and for making incorrect, unjustified, ambiguous, or vague claims in your solutions. You should clearly cite any sources or people you consult, other than the course notes, lecture materials, and tutorial exercises.

Your assignment must be typed to produce a PDF document **a2.pdf** (hand-written submissions are not acceptable). You may work on the assignment in groups of 1, 2, or 3, and submit a single assignment for the entire group on MarkUs.

1. Consider the following recursive linear search algorithm.

```
REC-LIN-SEARCH(A, i, x):
    if i == len(A):
        return False
    else:
        return A[i] == x or REC-LIN-SEARCH(A, i + 1, x)
```

(a) Define a recurrence $T(n)$ for the worst-case runtime of REC-LIN-SEARCH(A, i, x), where $n = len(A) - i$.

Sample solution:

$$T(n) = \begin{cases} 1 & \text{if } n = 0 \\ T(n-1) + 1 & \text{if } n > 0 \end{cases}$$

(b) Use unwinding to find a closed from for $T(n)$. Based on the closed form, state a conjecture for $f(n)$ such that $T(n) \in \Theta(f(n))$.

Sample solution:

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

Closed form: $T(n) = T(n-n) + n = T(0) + n = n + 1$.

Conjecture: $T(n) \in \Theta(n)$

(c) Prove the bound on $T(n)$ you gave in part (b) is correct.

Sample solution:

Sufficient to show that $T(n) = n + 1$.

Proof, by induction (other proofs OK too):

Let $n > 0$.

Assume $H(n)$**:** $\forall i \in \mathbb{N}, 0 \leq i < n, T(i) = i + 1$.

Show $H(n) \rightarrow C(n)$**:** $T(n) \leq n + 1$.

$T(n) = T(n-1) + 1 \leq ((n-1) + 1) + 1$ (by $H(n)$) $= n + 1$.

Base case: $T(0) = 1 = 0 + 1$.

Conclude: $T(n) = n + 1, \forall n \in \mathbb{N}$.

2. Consider the problem of finding the maximum sum of consecutive values in a sequence of integers. For example, the maximum consecutive sum in $2, -5, 3, -1, -1, 4$ is $3 + (-1) + (-1) + 4 = 5$, and the maximum consecutive sum in $1, -2, 3$ is 3. Let n be the number of integers in the sequence.

   (a) Give a brute force algorithm that solves the problem with worst-case running time $O(n^2)$.

   Sample solution:

```
# n = len(A)
def BruteForceSum(A):
    maxSum = 0
    for i = 0..len(A):
        innerSum = 0
        for j = i..len(A):
            innerSum += A[j]
            if innerSum > maxSum:
                maxSum = innerSum
    return maxSum
```

   (b) Informally state why your algorithm would have worst-case running time $O(n^2)$.

   Sample solution:

   The algorithm considers the sum of every possible subsequence of A. The inner loop iterates n times on the first iteration of the outer loop, $n-1$ on the section, and so on. $n + (n-1) + ... + 1 = \frac{n(n+1)}{2} \in \Theta(n^2)$.

Consider a Divide & Conquer approach to this problem, based on the idea that, for any sequence, either the middle element is included in the maximum consecutive sum, or it is not.

   (c) Based on the idea above, consider how many subproblems the original problem would be split into, and approximately how large each subproblem would be. Then, consider a recurrence $T(n)$ that represents the worst-case running time for an algorithm based on this idea. If the Master Theorem was applied to this algorithm, what would a and b be for $T(n)$?

   Sample solution: Each of the two subproblems would have roughly half the size of the original problem, so $a = 2$ and $b = 2$.

   (d) Let $f(n)$ be the cost of splitting and recombining from $T(n)$, and suppose $f(n) \in \Theta(g(n))$. For this D&C algorithm to perform better than the brute force approach in part (a), what must $g(n)$ be?

   Sample solution: $g(n) = n^d$ s.t. $d < 2$. (n is sufficient here, although there is no requirement that $d \in \mathbb{N}$.

   (e) Give a D&C algorithm for this problem, with worst-case runtime $T(n)$ that has a, b, and $f(n)$ as described in parts (c) and (d).

   Sample solution:

```
def DCSum(A, b, e):
    if b == e:
        return A[b]
    else:
        m = (b + e) // 2  # find midpoint
        # recursively solve non-empty subproblems, excluding mid
        if b <= m - 1:
            maxLeft = DCSum(A, b, m-1)
        if m + 1 <= e:
            maxRight = DCSum(A, m+1, e)
        # try all possible sums including midpoint
        # keeping track of the largest
        maxMid = A[m]
```

```
midSum = A[m]
for i = m+1..e:
    midSum += A[i]
    if midSum > maxMid:
        maxMid = midSum
midSum = maxMid
for i = m-1..b:  # decreasing
    midSum += A[i]
    if midSum > maxMid:
        maxMid = midSum
return max(maxLeft, maxRight, maxMid)
```

(f) Prove that $f(n) \in \Theta(g(n))$.

Sample solution:

Want to show that the lines following the recursive calls in the algorithm are in $\Theta(n)$.

All statements outside the loop are in $\Theta(1)$, as they make no function calls. For the same reason, the body of each loop is in $\Theta(1)$.

The first loop iterates $e - (m+1) + 1$ times, and the second loop iterates $(m-1) - b + 1$ times, for a total of $e - (m+1) + 1 + (m-1) - b + 1 = e - m + m - b = e - b = (e - b + 1) - 1 = n - 1$ iterations.

Thus, $f(n) \in \Theta(n)$.

(g) Use the Master Theorem determine $h(n)$ such that $T(n) \in \Theta(h(n))$.

Sample solution: Since $T(n)$ has $a = 2$, $b = 2$, and $f(n) \in \Theta(n)$ (ie. $d = 1$), then by the Master Theorem, $T(n) \in \Theta(n^d log n) = \Theta(n log n)$.

3. Recall that gcd(x, y) is the *greatest common divisor* of x and y, for all natural numbers x, y. Consider the following recursive algorithm to compute gcd(x, y).

```
REC-GCD(x, y):
    Precondition: x, y ∈ ℕ.
    Postcondition: the algorithm returns gcd(x, y).
    if x = 0 or y = 0:   return x + y
    if x = 1 or y = 1:   return 1
    if x is even:
        if y is even:    return 2 × REC-GCD(x/2, y/2)
        else:            return REC-GCD(x/2, y)
    else:
        if y is even:    return REC-GCD(x, y/2)
        else:
            if x < y:    return REC-GCD((y − x)/2, x)
            else:        return REC-GCD((x − y)/2, y)
```

(a) Give a detailed analysis of the algorithm's worst-case running time. (Start from a recurrence relation, show your work to obtain a closed form, and give a rigorous proof by induction that your closed form is correct).

Let n be the total number of bits to represent both x and y (i.e., $n = n_1 + n_2$ where it takes n_1 bits to represent x and n_2 bits to represent y). Also, assume that it takes linear time to perform basic arithmetic operations (like $+$ and $-$).

Sample solution:

Let n be the total number of bits to represent both x and y.

If x and y are represented using only 1 bit in total, then ($x = 0$ and $y = 1$) or ($x = 1$ and $y = 0$) so the algorithm performs only a constant number of operations.

If x and y are represented using $n > 1$ bits in total, then in the worst-case, the algorithm makes a recursive call. This involves computing $y - x$ (or $x - y$) in time $\Theta(n)$ (in the worst-case), and the input size for the recursive call has size at most $n - 1$ (n_2 bits for $y - x$, so $n_2 - 1$ bits for $(y - x)/2$).

This gives the following recurrence relation for the worst-case running time:

$$\begin{aligned} T(1) &= \Theta(1), \\ T(n) &= T(n-1) + \Theta(n) \qquad \text{for all } n > 1. \end{aligned}$$

The Master Theorem does not apply, but repeated substitution yields:

$$\begin{aligned} T(n) &= T(n-1) + n \\ &= T(n-2) + (n-1) + n \\ &= \cdots \\ &= T(1) + 2 + \cdots + n \\ &= n(n+1)/2 \end{aligned}$$

We can prove formally that this is correct (using 1 in place of $\Theta(1)$ and n in place of $\Theta(n)$ in the recurrence):

Let $n \geq 1$.

Assume $H(n)$**:** $T(n) = n(n+1)/2$.

Show $H(n) \to C(n)$**:** $T(n+1) = (n+1)((n+1)+1)/2$

$$\begin{aligned} T(n+1) &= T(n) + (n+1) \qquad \text{(from the recurrence)} \\ &= n(n+1)/2 + (n+1) \qquad \text{(by } H(n)\text{)} \\ &= (n/2+1)(n+1) \\ &= (n+1)((n+1)+1)/2 \end{aligned}$$

Base Case: $T(1) = 1 = 1(1+1)/2$.

Conclusion: $\forall n \geq 1, T(n) = n(n+1)/2$.

(b) Write a detailed proof by induction that the algorithm is correct.

HINT: Take the time to write down a precise definition of "greatest common divisor" (gcd) and clearly state the properties of gcd you need to use in your proof.

Sample solution:

By induction on $n \geq 1$, we prove that for all x, y represented using a total of n bits, the call REC-GCD(x, y) terminates and returns $\gcd(x, y)$, the greatest common divisor of x and y.

Inductive Step: Let $n > 1$

Assume $H(n)$**:** for all x, y represented using fewer than n bits, the call REC-GCD(x, y) terminates and returns $\gcd(x, y)$.

Let x, y be inputs represented using exactly n bits, and consider the call REC-GCD(x, y).

Then either $x = 0$ or $y = 0$ or $x = 1$ or $y = 1$ or ($x > 1$ and $y > 1$). And in the last case, either x is even or x is odd, and independently, either y is even or y is odd.

- If $x = 0$, then $\gcd(x, y) = \gcd(0, y) = y$ and the algorithm returns $x + y = 0 + y = y$ on the first line.
- Similarly, if $y = 0$, then the algorithm returns $x = \gcd(x, y)$ on the first line.
- If $x = 1$, then $\gcd(x, y) = \gcd(1, y) = 1$ and the algorithm returns 1 on the second line.
- Similarly, if $y = 1$, then the algorithm returns $1 = \gcd(x, y)$ on the second line.
- If $x > 1$ and $y > 1$ and x is even and y is even, then the algorithm makes a recursive call on $x/2$ and $y/2$, whose total bit-length is $n - 2$. By $H(n)$, the recursive call terminates and returns $\gcd(x/2, y/2)$. This means the call REC-GCD(x, y) also terminates and returns $2\gcd(x/2, y/2) = \gcd(2x/2, 2y/2) = \gcd(x, y)$.
- If $x > 1$ and $y > 1$ and x is even and y is odd, then the algorithm makes a recursive call on $x/2$ and y, whose total bit-length is $n - 1$. By $H(n)$, the recursive call terminates and returns $\gcd(x/2, y)$. This means the call REC-GCD(x, y) also terminates and returns $\gcd(x/2, y) = \gcd(x, y)$, since 2 is not a factor of y.

- Similarly, if $x > 1$, $y > 1$, x is odd, y is even, then the algorithm terminates and returns $\gcd(x, y/2) = \gcd(x, y)$.
- If $x > 1$ and $y > 1$ and x is odd and y is odd, then either $x < y$ or $x > y$ or $x = y$.
  - If $x < y$, then the algorithm makes a recursive call on $(y - x)/2$ and x, whose total length is at most $n - 1$ (since $y - x \leq y$ so $(y - x)/2 \leq y/2$). By $H(n)$, the recursive call terminates and returns $\gcd((y - x)/2, x)$.
    Now, by definition of gcd and since $x < y$, there exist positive integers $k < \ell$ such that $x = k \cdot \gcd(x, y)$ and $y = \ell \cdot \gcd(x, y)$ and k and ℓ have no common divisor. Since x and y are both odd, both k and ℓ are also odd, and $y - x = (\ell - k) \cdot \gcd(x, y)$. Since there are no divisors common to k and ℓ, there are no divisors common to k and $\ell - k$ so $\gcd((y - x)/2, y) = \gcd(x, y) \cdot \gcd((\ell - k)/2, k) = \gcd(x, y)$. This means the call REC-GCD(x, y) also terminates and returns $\gcd((y - x)/2, x) = \gcd(x, y)$.
  - Similarly, if $x > y$, the algorithm terminates and returns $\gcd((x - y)/2, y) = \gcd(x, y)$.
  - Finally, if $x = y$, then the algorithm makes a recursive call on $(x - y)/2 = 0$ and x, which returns x. This means the call REC-GCD(x, y) terminates and returns $x = \gcd(x, y)$.

  In every case, the call REC-GCD(x, y) terminates and returns $\gcd(x, y)$.

Base Case: If x and y are represented using a total of 1 bit, then either $x = 0$ and $y = 1$, or $x = 1$ and $y = 0$. In either case, the algorithm returns $1 = \gcd(x, y)$ on the first line.

Conclusion: By induction on n, for all $n \geq 1$ and all x, y represented using a total of n bits, the call REC-GCD(x, y) terminates and returns the greatest common divisor of x and y.