Homework 6 - Algorithms

Michael Gould
CS 506 - Online Foundations of CS

March 17, 2018

1. *Prove the following polynomial is* $\Theta(n^3)$. *That is, prove* $T(n)$ *is both* $O(n^3)$ *and* $\Omega(n^3)$

$$T(n) = 2n^3 - 10n^2 + 100n - 50$$

(a) *Prove* $T(n)$ *is* $O(n^3)$: *By definition, you must find positibe constants* C_1 *and* n_0 *such that*

$$T(n) \leq C_1 n^3, \forall n \geq n_0$$

(b) *Prove* $T(n)$ *is* $\Omega(n^3)$: *By definition, you must find positibe constants* C_1 *and* n_0 *such that*

$$T(n) \geq C_1 n^3, \forall n \geq n_0$$

2. (a) *Compute and tabulate the following functions for* $n = 1, 2, 4, 8, 16, 32, 64$.
(b) *Order the following complexity functions (growth rates) form the smallest to the largest.*

$$n^2 logn, 5, nlog^2 n, 2^n, N^2, N, \sqrt{n} logn, \frac{n}{logn}$$

3. *Find the exact number of times (in terms of* n*) the innnermost statement* $(X = X - 1)$ *is executed in the following code. That is, find the value of* X*, then express the total running time in terms of* $O(), \Omega(), \Theta()$ *as appropriate*

```
X = 0;
for k = 1 to n
    for j = 1 to n − k
        X = X + 1;
```

4. *the following program computes and returns* $(log_2 n)$*, assuming the input* n *is an integer power of 2. That is* $n = 2^j$ *for some integer* $j \geq 0$

```
int LOG (int n){
int m, k;
m = n;
k = 0;
```

while $(m > 1)$ {
$m = m/2;$
$k = k + 1;$ }
return (k)
}

(a) *First trace the execution of this program for a specific input value, $n = 16$. Tabulate the values of m and k at the beginning, just before the first execution of the while loop and after each execution of the while loop.*

(b) *Prove by induction that at the end of each execution of the while loop, the following relation holds between variables m and k.*

$$m = \frac{n}{2^k}$$

(c) *Then conclude that at the end, after the last iteration of the while loop, the program returns $k = log_2 n$.*

5. *The following pseudocode computes the sum of an array of n integers*

int sum (int A[], int n) {
$T = A[0];$
for $i = 1$ to $n - 1$
$T = T + A[i];$
return T;
}

(a) *Write a recursive version of this code*

(b) *Let $f(n)$ be the number of additions performed by this computation. Write a recurrance equation for $f(n)$.*

(c) *Prove by induction that the solution of the recurrance is $f(n) = n - 1$.*

6. *The following pseudocode finds the maximum element in an array of size n.*

int MAX (int A[], int n){
$M = A[0];$
for $i = 1$ to $n - 1$
if $(A[i] > M)$
$M = A[i]$ //Update the max
return M; }

(a) *Write a recursive version of this code*

(b) *Let $f(n)$ be the number of key comparisons performed by this algorithm. Write a recurrance equation for $f(n)$.*

(c) *Prove by induction that the solution of the recurrance is $f(n) = n - 1$.*

7. *Consider the following pseudocode for insertion-sort algorithm. The algorithm sorts an arbitrary array $A[0..n - 1]$ of n elements*

void ISORT(dtype A[], int n) {
int i, j;
for $i = 1$ to $n - 1$ {
//Insert $A[i]$ in the sorted part $A[0...i - 1]$
$j = i;$
while $(j > 0$ and $A[j] < A[j - 1])$ {

SWAP($A[j], A[j-1]$) {
$j = j - 1$ }
}
}
}

(a) *Illustrate the algorithm on the following array by showing each comparison/swap operation. What is the total number of comparisons made for this worst-case data?*

$$A = (5, 4, 3, 2, 1)$$

(b) *Write a recursive version of this algorithm.*

(c) *Let $f(n)$ be the worst-case number of key comparisons made bt this algorithm to sort n elements. Write a recurrance equation for $f(n)$*

(d) *Find the solution for $f(n)$ by repeated substition*

8.*Consider the bubble-sort algorithm described below.*

```
void bubble (dtype A[], int n) {
int i, j;
for (i = n − 1; i > 0; i − −)
    for(j = 0; j < i; j + +)
        if(A[j] > A[j + 1])     SWAP(A[j], A[j + 1]);     }
```

(a) *Analyze the time complexity, $T(n)$, of the bubble-sort algorithm.*

(b) *Rewrite the algorithm using recursion.*

(c) *Let $f(n)$ be the worst-case number of key-comparisons used by this algorithm to sort n elements. Write a recurrance for $f(n)$. Solve the recurrance by repeated subsititions*

9. *The wollowing algorithm uses a* **divide-and-conquer** *technique to find the maximum element in an array of size n. The initial call to this recursive function is max(arrayname, 0, n).*

```
dtype Findmax(dtype A[], int i, int n)
{   //i is the starting index and n is the number of elements
dtype Max1, Max2;
if(n == 1) return A[i];
Max1 = Findmax(A, i, ⌊n/2⌋);              //Find max of the first half
Max2 = Findmax(A, i + ⌊n/2⌋ , ⌈n/2⌉);     //Find max of the second half
if (Max1 ≥ Max2)      return Max1;
    else return Max2;
}
```

Let $f(n)$ be the worst-case number of key comparisons for finding the max of n elements

(a) *Assuming n is a power of 2, write a recurrance relation for $f(n)$. Find the solution by each of the following methods.*

(i) *Apply the repeated substition method.*

(ii) *Apply induction to prove that $f(n) = An + B$ and find the constants A and B.*

(b) *Now consider the general case where* n *is any integer. Write a recurrance for* $f(n)$. *Use induction to prove that the somution is* $f(n) = n - 1$.

10. *The following divide-and-conquer algorithm is designed to return TRUE if and only if all the elements of the array have equal values. For simplicity, suppose the array size is* $n = 2^k$ *for some integer* k. *Input* S *is the starting index and* n *is the number of elements starting at* S. *The initial call is* $SAME(A, 0, n)$.

Boolean SAME (Int $A[]$, int S, int n) {
Boolean $T1, T2, T3$;
if ($n == 1$) return TRUE;
T1 = SAME $(A, S, n/2)$;
T2 = SAME $(A, S + n/2, n/2)$;
T3 = $(A[S] == A[S + n/2])$;
return $(T1 \wedge T2 \wedge T3)$;
}

(a) *Explain how this program works*

(b) *Prove by induction that the algorithm returns TRUE if and only if all the elements of the array have equal values.*

(c) *Let* $f(n)$ *be the number of key comparisons in this algorithm for any array of size* n. *Write a Recurrance for* $f(n)$.

(d) *Find the solutio by repeated subsitition.*