CMPE 300 - Analysis of Algorithms
Fall 2014
Assignment 1

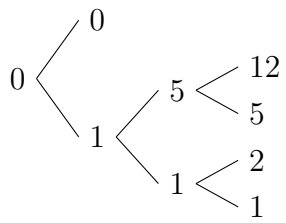
Question 1 (40 Points)

1. Write a pseudocode for an **efficient** algorithm which finds both the **largest** and **second-smallest** element in a list L[1:n] of size n.

2. Find worst-case complexity *W(n)* and average-case complexity *A(n)* of the algorithm in (a). You can assume input is uniformly distributed.

Answer 1

1. Pseudocode: **(20 pts)** Optimal algorithm is the one that uses a tournament style comparison of items for finding the second smallest item in a list. Since we are also looking for the largest element, we can use two functions to calculate them separately and return the results. For finding the second smallest, think of this input: $\{1, 2, 5, 12, 0\}$



Since n is odd, 0 should be compared in the last tournament. The other elements should be compared 2 by 2, finding the smallest element at each time, and saving it as a parent node, i.e. adding the larger element to the smaller element's children list.

In order to find the largest element, we need to compare the elements in the list to the largest found item in the list so far.

The pseudocode for the algorithm is as follows:

procedure LARGESTSECONDSMALLEST(L[1:n])
  input: L[1:n] (an array of integers with size n)
  output: max, smin (largest and second-smallest elements in the list)
  $max \leftarrow$ Largest(L[1:n])
  $smin \leftarrow$ SecondSmallest(L[1:n])

procedure LARGEST(L[1:n])
  input: L[1:n] (an array of integers with size n)

```
    output: max (largest elements in the list)
    Set max ← L[1]
    for i = 2 to n do
        if L[i] > max then
            max ← L[i]
procedure SecondSmallest(L[1:n])
    input: L[1:n] (an array of integers with size n)
    output: smin (second-smallest elements in the list)
    Declare vector children[n]
    Declare vector compare as {1, ..., n}
    odd ← 0
    while size(compare) > 1 do
        if size(compare) is odd then
            odd ← 1
        Declare vector compare2
        for i = 1 to size(compare) − odd, increment i by 2 do
            i0 ← compare[i]
            i1 ← compare[i + 1]
            if L[i0] < L[i1] then
                add children[i] ← L[i1]
                add compare2 ← i0
            else
                add children[i + 1] ← L[i0]
                add compare2 ← i1
        if odd is 1 then
            add compare2 ← compare[size(compare)]
        compare ← compare2
    min ← compare[1]
    if size(children[min]) is 0 then
        smin ← min
    else
        smin ← children[min][1]
        for i = 2 to size(children[min]) do
            if children[min][i] < smin then
                smin ← children[min][i]
```

2. Worst case complexity and average complexity of the algorithm: **(20 pts)**

$$W(n) = 2n + \lceil \log_2 n \rceil - 3$$
$$A(n) = 2n + \lceil \log_2 n \rceil - 2$$

The analysis is as follows:

For Largest(L[1:n]) function:

Basic operation is comparison.

The worst case complexity will be $n-1$, since algorithm loops over $n-1$ items. Since for every item a comparison will be made, no matter the item in the list or the list size, average case complexity of Largest(A[1:n]) is also $A(n) = n - 1$.

For SecondSmallest(L[1:n]) function:

Worst case complexity anaysis:

Basic operation is comparison.

$W(n) = n - 1$ (first loop for the tournament for finding the smallest element) + $\lceil \log_2 n \rceil - 1$ (second loop for finding the second-smallest element in children of the smallest element)

Therefore,

$$W(n) = n + \lceil \log_2 n \rceil - 2$$

For the tournament, assume $n = 2^k$. The total number of comparisons will be $n/2 + n/4 + ... + n/2^k = n - 1$.

In order to find the second smallest element, we need to find an element which has lost a match to the smallest element (winner). There will be at most $\lceil k \rceil - 1 = \lceil \log_2 n \rceil - 1$ comparisons for finding the second smallest in the children of smallest, since there will be at most k children of the smallest element.

Total worst case complexity:

$$W(n) = 2n + \lceil \log_2 n \rceil - 3$$

See text book page: 367-368 for the proof of optimality.

For average case analysis:

The tournament will take $n - 1$ comparisons, no matter the input.

$$A(n) = n - 1 + E[T]$$

If n is odd and L[n] is the smallest element in the list, there will be only 1 comparison, since the odd number is compared only at the end to the smallest so far found number, as in our example input. Otherwise there will be $\lceil \log_2 n \rceil$ comparisons for finding the second smallest element in the children list of the smallest element. Since the input is uniformly distributed:

(a) Assume that the probability of a successful search is $p (0 \leq p \leq 1)$

(b) Assume that it is equally likely that n will be odd as n will be even

(c) Assume that it is equally likely that second-smallest element can be found in any position

$$E[T] = \frac{1}{2}[\frac{p}{n} + \frac{p(n-1)}{n}\lceil \log_2 n \rceil] + \frac{1}{2}p\lceil \log_2 n \rceil$$
$$E[T] = \frac{1}{2}p[[\frac{1}{n} + \lceil \log_2 n \rceil - \frac{\lceil \log_2 n \rceil}{n}] + \lceil \log_2 n \rceil]$$

If $n \to \infty$ and $p = 1$

$$E[T] = \lceil \log_2 n \rceil$$
$$A(n) = n - 1 + \lceil \log_2 n \rceil$$

Total average complexity:

$$A(n) = 2n + \lceil \log_2 n \rceil - 2$$

Another (not optimal) algorithm for second smallest:

```
procedure SecondSmallestNE(L[1:n])
    input: L[1:n] (an array of integers with size n)
    output: smin (second-smallest elements in the list)
    if L[1] ≤ L[2] then
        Set min ← L[1]
        Set smin ← L[2]
    else
        Set min ← L[2]
        Set smin ← L[1]
    for i = 3 to n do
        if L[i] < smin then
            if L[i] < min then
                smin ← min
                min ← L[i]
            else
                smin ← L[i]
```

Worst case complexity for SecondSmallestNE(L[1:n]):

There will be 1 comparison made for the first If statement, and $2(n-2)$ for the for loop, at the worst case. Therefore,

$$W(n) = 2n - 3$$

Total worst case complexity for LargestSecondSmallest(L[1:n]):

$$W(n) = 3n - 4$$

Average complexity for SecondSmallestNE(L[1:n]):

Let Y_i be the indicator variable that has value 1 if $L[i]$ is bigger than the second largest element of $\{L[j]|1 \leq j < i\}$, and 0 otherwise. Then

$E[Y_i] = Prob[Y_i = 1] = Prob[L[i]$ is the largest or second largest element in $L[1..i]] = \frac{1}{i} + \frac{1}{i} = \frac{2}{i}$.

In this algorithm, there is 1 element comparison before the loop and 1 element comparison (with *smin*) each iteration of the loop. Furthermore, there is 1 element comparison (with *smin*) each iteration of the loop in which $L[i] > smin$ i.e. in which $Y_i = 1$. Thus the expected number of element comparisons performed is

$$A(n) = 1 + n - 2 + \sum_{i=3}^{n} E[Y_i]$$

$$A(n) = n - 1 + \sum_{i=3}^{n} \frac{2}{i}$$

$$A(n) = n + \Theta(\log n)$$

Total average case complexity for LargestSecondSmallest(L[1:n]):

$$A(n) = 2n + \Theta(\log n) - 1$$

NOTE: *Those who have written an algorithm of $W(n) = 3n - 4$ will be given 15 pts for the part (1) of the question, because of the* ***efficient*** *part in the question. If your algorithm has worse than $W(n) = 3n - 4$ complexity you will get 0 pts for the part (1) of the question.*

Question 2 (30 Points)

Find the recurrence relation for the following algorithm:

procedure SplitArMultiplication(A[1:n]) **recursive**
input: A[1:n] (an array of integers with size n)
output: A[1:n] (array altered by the procedure)
if $n = 1$ **then** return
for $i = 1$ to $n/2$ **do**
$A[i] := A[i] * A[i + n/2]$
SplitArMultiplication(A[1:$n/2$])

Answer 2

Assume n to be $n = 2^i$ then $i = \log n$, where $n >= 2$
Relation: **(10 pts)**

$$T(n) = T(n/2) + n/2$$
$$T(1) = 1$$

Complexity Analysis: **(20 pts)**

$$\begin{aligned} T(n) &= T(n/2^2) + n/2^2 + n/2 \\ &= T(n/2^3) + n/2^3 + n/2^2 + n/2 \\ &= T(n/2^4) + n/2^4 + n/2^3 + n/2^2 + n/2 \\ &= T(n/2^i) + n/2^i + ... + n/2^2 + n/2 \\ &= T(1) + n * (1/2^{\log n} + ... + 1/2^2 + 1/2) \\ &= 1 + O(n) \quad since(1/2^{\log n} + ... + 1/2^2 + 1/2) < 1 \\ &= O(n) \end{aligned}$$

Question 3 (30 Points)

Consider the given function f(n) and determine whether the following cases are true or false. Justify your answers formally.*(Hint: Use Stirling's Approximation)*

$$f(n) = n^3 + n^3 log(n^5 * n!) + n^2 \tag{1}$$

1. $f(n) \in O(n^3)$ **(6 pts)**
2. $f(n) \in o(n^3 log(n))$ **(8 pts)**
3. $f(n) \in \Omega(n^2 log(n))$ **(8 pts)**
4. $f(n) \in \Theta(n^3 log(n))$ **(8 pts)**

Answer 3

We use Stirlings approximation to simplify the term $log(n!)$.

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \tag{2}$$

The function $f(n)$ becomes

$$\begin{aligned} f(n) &= n^3 + 5n^3 log(n) + n^3 log(n!) + n^2 \\ &= n^3 + 5n^3 log(n) + n^3 log(\sqrt{2\pi n}\left(\frac{n}{e}\right)^n) + n^2 \\ &= n^3 + 5n^3 log(n) + n^3 log(\sqrt{2\pi n}) + n^4 log\left(\frac{n}{e}\right) + n^2 \\ &= n^3 + 5n^3 log(n) + \frac{1}{2}n^3 log(2\pi n) + n^4 log(n) - n^4 log(e) + n^2 \\ &= n^4 log(n) - n^4 log(e) + \frac{11}{2}n^3 log(n) + \frac{1}{2}n^3 log(2\pi) + n^3 + n^2 \end{aligned}$$

1. $f(n) \in O(n^3) : False$

   Solution 1:

   $\lim_{n\to\infty} \frac{f(n)}{n^3} = \infty$

   $f(n) \in \omega(n^3)$

   Therefore $f(n) \notin O(n^3)$.

   Solution 2:

   We can directly show that there is no c and n_0 such that $f(n) \leq n^3$. Let's look at the term $n^4(log(n) - log(e))$.

   $n^4(log(n) - log(e)) \leq cn^3, \qquad \forall n \geq n_0$

   $n(log(n) - log(e)) \leq c$

   $nlog(n)$ is a monotonically increasing function and c is constant. For all $\{c, n_0\}$ pairs, there is an n value, which is greater than n_0, which makes $n(log(n) - log(e)) > c$. We don't need to look at other terms.

2. $f(n) \in o(n^3 log(n)) : False$

   Solution 1:

   $\lim_{n\to\infty} \frac{f(n)}{n^3 log(n)} = \infty$

   $f(n) \in \omega(n^3 log(n))$

Therefore $f(n) \notin o(n^3 log(n))$.

Solution 2:

We can directly show that there is no c and n_0 such that $f(n) < n^3$. Let's look at the term $n^4 log(n)$.

$n^4 log(n) < cn^3 log(n), \qquad \forall n \geq n_0$

$n < c$

n is a monotonically increasing function and c is constant. For all $\{c, n_0\}$ pairs, there is an n value, which is greater than n_0, which makes $n > c$. We don't need to look at other terms.

3. $f(n) \in \Omega(n^2 log(n)) : True$

Solution 1:

$\lim_{n\to\infty} \frac{f(n)}{n^2 log(n)} = \infty$

$f(n) \in \omega(n^2 log(n))$

Solution 2:

If we can find c and n_0 such that $cg(n) \leq f(n) \forall n \geq n_0$, then $f(n) \in \Omega(n^2 log(n))$. Consider the term $n^4(log(n) - log(e))$:

$cn^2 log(n) \leq n^4(log(n) - log(e))$

$c \leq n^2 \frac{log(n) - log(e)}{log(n)}$

Let $c = 1$ for all values of $n \geq 4 (n_0 = 4)$. This implies that $cg(n) \leq f(n)$ since the remaining terms of $f(n)$ are positive.

4. $f(n) \in \Theta(n^3 log(n)) : False$

Solution 1:

$\lim_{n\to\infty} \frac{f(n)}{n^3 log(n)} = \infty$

$f(n) \in \omega(n^3 log(n))$

Therefore $f(n) \notin \Theta(n^3 log(n))$.

Solution 2:

We need to find c_1, c_2 and n_0 such that

$c_1(n^3 log(n)) \leq f(n) \leq c_2(n^3 log(n)) \qquad \forall n \geq n_0$

Let's look at the term: $n^4(log(n) - log(e))$:

$c_1(n^3 log(n)) \leq n^4(log(n) - log(e)) \leq c_2(n^3 log(n))$

$c_1 \leq n\frac{log(n)-log(e)}{log(n)} \leq c_2$

Since $n\frac{log(n)-log(e)}{log(n)}$ is an increasing function, as $n \to \infty$ it cannot be bounded by a constant c_2. Therefore, we cannot find a c_2 value.