**CMPE 300 ANALYSIS OF ALGORITHMS**

###### MIDTERM ANSWERS

1. function InsertionSort (L[0:n-1], i)

 if (i>0) then

 call InsertionSort (L[0:n-1], i-1) // sort L[0:i-1] by recursive calls;

 current 🡨 L[i] // then place L[i] into proper position

 position 🡨 i-1 // in L[0:i-1]

 while (position ≥ 0) and (current < L[position]) do

 L[position+1] 🡨 L[position]

 position 🡨 position-1

 endwhile

 L[position+1] 🡨 current

 endif

end

The algorithm is called initially as “call InsertionSort (L[0:n-1], n-1)”.

1. Basic operation is the comparison “(current < L[position])”.

We can view the algorihm as formed of two parts: i) the recursive call (T1) and ii) placing L[i] into proper position in L[0:i-1] (T2). Thus, we can write

T = T1 + T2

So,

A(n) = E[T] = E[T1] + E[T2] = A(n-1) + E[T2]

Now, we need to find E[T2], i.e. part (ii) for data size n. This depends on where L[n-1] (current) will be placed in L[0:n-2]. We have the following cases:

There will be 1 comparison if L[n-2]≤L[n-1]

There will be 2 comparisons if L[n-3]≤L[n-1]<L[n-2]

There will be 3 comparisons if L[n-4]≤L[n-1]<L[n-3]

...

There will be n-1 comparisons if L[0]≤L[n-1]<L[1]

There will be n-1 comparisons if L[n-1]<L[0]

Assuming that each case is equally likely (there are n cases), we have the following probability distribution:

p(T2=i) = 1/n for 1≤i≤n-2

p(T2=i) = 2/n for i=n-1

Using the expectation formula,

$$E\left[T\_{2}\right]=\sum\_{i=1}^{n-1}i\*P(T\_{2}=i)=\left[\sum\_{i=1}^{n-2}\frac{1}{n}i\right]+\frac{2(n-1)}{n}=\left[\sum\_{i=1}^{n-1}\frac{1}{n}i\right]+\frac{n-1}{n}=\frac{n-1}{2}+\frac{n-1}{n}$$

So,

A(n) = A(n-1) + $\left(\frac{n-1}{2}+\frac{n-1}{n}\right)$. That is,

A(n) = A(n-1) + x(n), for x(n)=$ \left(\frac{n-1}{2}+\frac{n-1}{n}\right)$; A(0)=0

If we solve this by backward substitution, we get

$$A\left(n\right)=\sum\_{i=1}^{n}x(i)=\left(\sum\_{i=1}^{n}\frac{i-1}{2}+\frac{i-1}{i}\right)=\left(\sum\_{i=1}^{n}\frac{i-1}{2}+1-\frac{1}{i}\right)=\frac{1}{2}\frac{n(n-1)}{2}+n-H(n)\in θ(n^{2})$$

1. Assume that n=2k+1.

Assume that when the algorithm is called with L[low:high] and the search element is compared with L[low+(high-low)/4)] (say, L[middle]), the two sublists will be L[low:middle] and L[middle:high] (instead of L[low:middle-1] and L[middle+1:high] as in the original algorithm) to maintain the 2k+1 data size at each call.

Best case occurs if we choose the smaller sublist at each iteration until we reach a sublist with size one. Then, the search element will be compared, for example, with L[2k/4] (i.e. L[2k-2]), L[2k-2/4] (i.e. L[2k-4]), L[2k-4/4] (i.e. L[2k-6]), ..., L[0]. The number of comparisons is k/2. So,

$B\left(n\right)=\frac{k}{2}=\frac{log\_{2}(n-1)}{2}=log\_{4}(n-1)\in θ(log n)$, for n=2k+1.

$log\_{4}(n-1)$ and $log n$ are eventually nondecreasing; $log n$ is Ө-invariant under scaling. So, by interpolation,

$B\left(n\right)\in θ(log n)$, for all n.

1. Assume that n=3k+1.

Worst case occurs if we choose the larger sublist at each iteration and the search element does not occur in the list. Then, X will be compared with L[n-3], L[n-6], L[n-9], ..., L[1]. The number of comparisons is $\frac{n-1}{3}$. That is,

$W\left(n\right)=\frac{n-1}{3}\in θ(n)$, for n=3k+1.

$\frac{n-1}{3}$ and $n$ are eventually nondecreasing; $n$ is Ө-invariant under scaling. So, by interpolation,

$W\left(n\right)\in θ(n)$, for all n.

1. x(n) = 5 x(n-1) – 6 x(n-2)

Characteristic equation: α2 = 5α – 6

When we solve, we obtain two real roots: α = 2, α = 3

So, the solution will have the form: x(n) = c1 2n + c2 3n

From the initial conditions,

 9 = c1 + c2

 20 = 2 c1 + 3 c2

We obtain c1 = 7 and c2 = 2

Thus, the solution is:

x(n) = 7 2n + 2 3n

1. By backward substitution:

$$T\left(m,n\right)=2T\left(\frac{m}{2},\frac{n}{2}\right)+mn$$

$$ =2^{2}T\left(\frac{m}{2^{2}},\frac{n}{2^{2}}\right)+\frac{mn}{2}+mn$$

$$ =2^{3}T\left(\frac{m}{2^{3}},\frac{n}{2^{3}}\right)+\frac{mn}{2^{2}}+\frac{mn}{2}+mn$$

$$ =2^{4}T\left(\frac{m}{2^{4}},\frac{n}{2^{4}}\right)+\frac{mn}{2^{3}}+\frac{mn}{2^{2}}+\frac{mn}{2}+mn$$

...

$$ =2^{k}T\left(\frac{m}{2^{k}},\frac{n}{2^{k}}\right)+\frac{mn}{2^{k-1}}+\frac{mn}{2^{k-2}}+…+\frac{mn}{2}+mn$$

Set $k=log\_{2}m$

$$ =mT\left(1,\frac{n}{2^{k}}\right)+\frac{mn}{2^{k-1}}+\frac{mn}{2^{k-2}}+…+\frac{mn}{2}+mn$$

$$ =\frac{mn}{2^{k}}+\frac{mn}{2^{k-1}}+\frac{mn}{2^{k-2}}+…+\frac{mn}{2}+mn$$

$$ =mn\left(\frac{1}{2^{k}}+\frac{1}{2^{k-1}}+\frac{1}{2^{k-2}}+…+\frac{1}{2}+1\right)$$

$$ =mn\sum\_{i=0}^{k}\left(\frac{1}{2}\right)^{i}$$

$$ =mn\left(\frac{\left(^{1}/\_{2}\right)^{k+1}-1}{^{1}/\_{2}-1}\right)$$

$$ =mn\left(2-\frac{1}{2^{k}}\right)\in θ(mn)$$

1. For any functions f(n), g(n), h(n) ∈ ℑ,

 Reflexivity: f(n) ∈ Ө (f(n))

There must exist positive constants c1, c2, and no such that

c1 f(n) ≤ f(n) ≤ c2 f(n), whenever n ≥ no.

If we take c1 = c2 = no = 1, it is obvious that this equation is satisfied.

 Symmetricity: f(n) ∈ Ө (g(n)) 🡺 g(n) ∈ Ө (f(n))

From the left part of the equation, we know that there exist positive constants c1, c2, and no such that

c1 g(n) ≤ f(n) ≤ c2 g(n), whenever n ≥ no.

It follows that

g(n) ≤ (1/c1) f(n) and g(n) ≥ (1/c2) f(n)

Thus, (1/c2) f(n) ≤ g(n) ≤ (1/c1) f(n), whenever n ≥ no (for the same no value)

Since (1/c1) and (1/c2) are constants, the property g(n) ∈ Ө (f(n)) is satisfied.

 Transitivity: f(n) ∈ Ө (g(n)) and g(n) ∈ Ө (h(n)) 🡺 f(n) ∈ Ө (h(n))

From the left part of the equation, we know that there exist positive constants c1, c2, no, d1, d2, mo such that

c1 g(n) ≤ f(n) ≤ c2 g(n), whenever n ≥ no

d1 h(n) ≤ g(n) ≤ d2 h(n), whenever n ≥ mo.

It follows that

f(n) ≤ c2 (g(n)) ≤ c2 (d2 h(n)), whenever n ≥ maximum of (no, mo)

and

f(n) ≥ c1 (g(n)) ≥ c1 (d1 h(n)), whenever n ≥ maximum of (no, mo)

Thus, c1 d1 h(n) ≤ f(n) ≤ c2 d2 h(n), whenever n ≥ maximum of (no, mo).

Since c1 d1 and c2 d2 are constants, the property f(n) ∈ Ө (h(n)) is satisfied.