Assignment 2 - CS 4071 - Spring 2018

Due: 2018-02-19 Group #13: Robert DiMartino (dimartrt), Hayden Schiff (schiffha), Jeremiah Leak (leakjz)

1. Exercise 2.24

Problem: Give pseudocode for interpolation search, and analyze its worst-cast complexity.

The idea here is to modify a Binary Search to assign `mid` the index we where would expect to find to find `X` if the elements of `L` were on a straight line from `(low,L[low])` to `(high,L[high])`. This requires solving the following:

```
y-y_{1} = m(x-x_{1}) \\
m = \frac{y_{2}-y_{1}}{x_{2}-x_{1}} \\
y - y_{1} = \left(\frac{y_{2}-y_{1}}{x_{2}-x_{1}}\right)(x-x_{1}) \\
x = \frac{(y-y_{1})(x_{2}-x{1})}{y_{2}-y_{1}}+x_{1}
```

Preview

OK

$$y - y_1 = m(x - x_1)$$
$$m = \frac{y_2 - y_1}{x_2 - x_1}$$
$$y - y_1 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x - x_1)$$
$$x = \frac{(y - y_1)(x_2 - x1)}{y_2 - y_1} + x_1$$

Where x = `mid`, x_1 = `low`, x_2 = `high`, y = `X`, y_1 = `L[low]`, y_2 = `L[high]`.

function `InterpolationSearch(L[0:n-1], low, high, X)` **recursive Input**: `L[0:n-1]`, (an array of real numbers sorted in increasing order), `X` (a search number), `low` and `high` (the lower and upper indexes to be searched) **Output**: returns an index of an occurence of `X` in the sublist `L[low:high]` or `-1` if `X` is not in the sublist

```
if low>high then return(-1) endif
mid ← floor((X-L[low])*(high-low)/(L[high]-L[low]))+low
case
    :X = L[mid]: return(mid)
    :X < L[mid]: return(InterpolationSearch(L[0:n-1], low, mid-1, X))
    :default: return(InterpolationSearch(L[0:n-1], mid+1, high, X))
endcase
```

The worst-case complexity for Interpolation search occurs when searching for the second to last element when the last element is much larger than the other elements in the list, e.g. searching for the index of `5` in `[1,2,3,4,5,99999]`. In this case, `mid` will be set to `low` in the interpolation step, so every index from `0` to `n-1` will be checked. This means that the worst-case complexity $W(n) \in O(n)$.

2. Exercise 3.6

Problem: Using the Ratio Limit Theorem, prove the following:

$$O(108) \subset O(\log n) \subset O(n) \subset O(n \log n) \subset O(n^2) \subset O(n^3) \subset O(2^n) \subset O(3^n)$$

The Ratio Limit theorem says that $\lim_{n\to\infty} f(n)/g(n) = 0 \implies O(f(n)) \subset O(g(n))$. So in order to prove thie former, we need to show that the corresponding limit is zero for each consecutive pair of functions. This will require liberal application of L'Hôpital's Rule.

i.

$$\lim_{n\to\infty} \frac{108}{\log n} = 0$$
$$\therefore O(108) \subset O(\log n)$$

ii.

$$\lim_{n\to\infty} \frac{\log n}{n}$$
$$= \lim_{n\to\infty} \frac{\frac{1}{n}}{1}$$
$$= \frac{0}{1} = 0$$
$$\therefore O(\log n) \subset O(n)$$

iii.

$$\lim_{n\to\infty} \frac{n}{n\log n}$$
$$= \lim_{n\to\infty} \frac{1}{1+\log n}$$
$$= 0$$
$$\therefore O(n) \subset O(n\log n)$$

iv.

$$\lim_{n\to\infty} \frac{n\log n}{n^2}$$
$$= \lim_{n\to\infty} \frac{1+\log n}{2n}$$
$$= \lim_{n\to\infty} \frac{\frac{1}{n}}{2}$$
$$= \frac{0}{2} = 0$$
$$\therefore O(n\log n) \subset O(n^2)$$

v.

$$\lim_{n\to\infty} \frac{n^2}{n^3}$$
$$= \lim_{n\to\infty} \frac{1}{n}$$
$$= 0$$
$$\therefore O(n^2) \subset O(n^3)$$

vi.

$$
\begin{aligned}
&\lim_{n\to\infty} \frac{n^3}{2^n} \\
&= \lim_{n\to\infty} \frac{3n^2}{2^n \log 2} \\
&= \lim_{n\to\infty} \frac{6n}{2^n \log^2 2} \\
&= \lim_{n\to\infty} \frac{6}{2^n \log^3 2} \\
&= 0 \\
&\therefore O(n^3) \subset O(2^n)
\end{aligned}
$$

vii.

$$
\begin{aligned}
&\lim_{n\to\infty} \frac{2^n}{3^n} \\
&= \lim_{n\to\infty} \left(\frac{2}{3}\right)^n \\
&= 0 \\
&\therefore O(2^n) \subset O(3^n)
\end{aligned}
$$

3. Exercise 3.26

Problem: Obtain a formula for the order of $S(n) = \sum_{i=1}^{n} (\log i)^2$.

We start by showing that $S(n) \in O(n \log^2 n)$.

$$
\begin{aligned}
S(n) &= \sum_{i=1}^{n} (\log i)^2 = \sum_{i=1}^{n} \log^2 i \\
&= \log^2 1 + \log^2 2 + \cdots + \log^2 n
\end{aligned}
$$

It can be shown that $f(x) = \log^2 x$ has a global minimum as $x = 1$ and is increasing for $x \geq 1$. So it holds that $\log^2 a \leq \log^2 b$ when $1 \leq a \leq b$. It follows that,

$$
\begin{aligned}
\log^2 1 + \log^2 2 + \cdots + \log^2 n &\leq \log^2 n + \log^2 n + \cdots + \log^2 n \\
&= n \log^2 n \\
\therefore S(n) &\in O(n \log^2 n)
\end{aligned}
$$

Now, we intend to show that $S(n) \in \Omega(n \log^2 n)$. Let $m = \lfloor n/2 \rfloor$. Then,

$$S(n) = \sum_{i=1}^{n} \log^2 i = \sum_{i=1}^{m} \log^2 i + \sum_{i=m+1}^{n} \log^2 i$$

$$\geq \sum_{i=m+1}^{n} \log^2 i = \log^2(m+1) + \log^2(m+2) + \cdots + \log^2 n$$

$$\geq \log^2(m+1) + \log^2(m+1) + \cdots + \log^2(m+1)$$

$$= (n-m)\log^2(m+1)$$

$$\geq \frac{n}{2}\log^2\left(\frac{n}{2}\right)$$

$$= \frac{n}{2}(\log n - \log 2)^2$$

For sufficiently large n,

$$\frac{n}{2}(\log n - \log 2)^2 \geq \frac{n}{2}\left(\log n - \frac{\log n}{2}\right)^2$$

$$= \frac{n}{2}\left(\frac{\log n}{2}\right)^2$$

$$= \frac{n}{8}(\log n)^2$$

$$= \frac{1}{8}(n \log^2 n)$$

$$\therefore S(n) \in \Omega(n \log^2 n)$$

Since $S(n) \in O(n \log^2 n)$ and $S(n) \in \Omega(n \log^2 n)$, then $S(n) \in \Theta(n \log^2 n)$.

4. Exercise 3.37

a.

Problem: Give a recurrence relation for the worst-case complexity $W(n)$ of `TriMergeSort` for an input list of size n.

The worst-case complexity of `TriMergeSort` will be similar to the worst-case complexity of `MergeSort`, which is $W(n) = 2W(n/2) + n - 1$.

The $2W(n/2)$ term comes from the fact that we recursively call `MergeSort` twice, once for each half of the list, i.e. $n/2$ elements. `TriMergeSort` recursively calls `TriMergeSort` 3 times, once for each third of the list. The corresponding term in the worst-case complexity then is $3W(n/3)$.

The last part of the worst-case complexity for `MergeSort` comes from the worse-case complexity of `merge` for n elements, which is $n - 1$. `TriMergeSort` calls `merge` twice, once for the first two-thirds of the list, then again for the whole list. That means the worse-case complexity for both calls to `merge` in `TriMergeSort` is $(2n/3 - 1) + (n - 1) = 5n/3 - 2$.

Then, for `TriMergeSort` the worse-case complexity is $W(n) = 3W(n/3) + \frac{5}{3}n - 2$.

b.

Problem: Solve the recurrence formula you have given in (a) to obtain an explicit formula for the worst-case complexity $W(n)$ of `TriMergeSort`.

The initial condition for `TriMergeSort` is the same as for `MergeSort`. A list of 1 element is already sorted, therefore $W(1) = 0$.

Assume that $n = 3^k$ for some positive integer k. Then

$$
\begin{aligned}
W(n) &= 3W(n/3) + \frac{5}{3}n - 2 \\
W(n) &= 3\left(3(W(n/3^2) + \frac{5n}{3^2} - 2\right) + \frac{5}{3}n - 2 \\
&= 3^2W(n/3^2) + \frac{5}{3}n - 6 + \frac{5}{3}n - 2 \\
&= 3^2W(n/3^2) + 2 \times \frac{5}{3}n - (2 + 3 \times 2) \\
W(n) &= 3^2(3W(n/3^3) + \frac{5n}{3^3} - 2) + 2 \times \frac{5}{3}n - (2 + 3 \times 2) \\
&= 3^3W(n/3^3) + 3 \times \frac{5}{3}n - 2(1 + 3 + 3^2) \\
&\vdots \\
W(n) &= 3^kW(n/3^k) + \frac{5}{3}kn - 2(1 + 3 + 3^2 + \cdots + 3^{k-1}) \\
&= 3^kW(1) + \frac{5}{3}kn - 2(1 + 3 + 3^2 + \cdots + 3^{k-1}) \\
&= \frac{5}{3}kn - 2(3^k - 1)
\end{aligned}
$$

Since $k = \log_3 n$, we have

$$
W(n) = \frac{5}{3}n\log_3 n - 2n - 2
$$

c.

Problem: Which is more efficient in the worst case, `MergeSort` or `TriMergeSort`? Discuss.

The asymptotic behavior of `MergeSort` and `TriMergeSort` is similar. For sufficiently large n, the $n\log_2 n$ and the $n\log_3 n$ terms, respectively, dominant the growth of the worst-case complexities. We also recall that the change of base for logarithms is a constant, which implies that $n\log_a n \in \Theta(n\log n)$ for any base $a > 1$. Therefore, despite the different bases, `MergeSort` and `TriMergeSort` are both order $n\log n$.

Because of the additional implementation complexity of `TriMergeSort`, `MergeSort` should be preferred.

5.

Problem: Consider the sorting algorithm Insertion Sort for sorting a list `L[0:n - 1]`. **Derive** a recurrence relation for the worst-case complexity $W(n)$ and **solve**.

To measure the complexity of Insertion Sort we consider the number of operations it takes to scan the list and shift elements in the list as appropriate.

For a list of size $n = 1$, the list is already sorted and it requires no further operations. This gives us our initial condition: $W(1) = 0$.

In a list that is sorted in reverse order, we have to scan and shift the entire remaining list to move the element to the start of the list. This is the worst case for Insertion Sort.

To measure the complexity of this worst case scenario, we can consider a recursive implementation of Insertion Sort. For a list of size n, it would take $n - 1$ scans and shifts of the elements to put the last element in the list in the first position. Then the we still need to recursively perform Insertion Sort with second to last element in the list. This gives us the recurrence relation:
$W(n) = W(n-1) + n - 1$.

To solve this recurrence relation:

$$
\begin{aligned}
W(n) &= W(n-1) + n - 1 \\
W(n) &= (W(n-2) + (n-1) - 1) + n - 1 \\
&= W(n-2) + n - 2 + n - 1 \\
W(n) &= (W(n-3) + (n-2) - 1) + n - 2 + n - 1 \\
&= W(n-3) + n - 3 + n - 2 + n - 1 \\
&\vdots \\
W(n) &= W(n-k) + (n-k) + (n-k+1) + \cdots + (n-2) + (n-1)
\end{aligned}
$$

When $k = n$,

$$
W(n) = W(0) + 0 + 1 + 2 + \cdots + (n-2) + (n-1)
$$

Applying the initial condition, $W(0) = 0$,

$$
\begin{aligned}
W(n) &= 0 + 0 + 1 + 2 + \cdots + (n-2) + (n-1) \\
W(n) &= \sum_{i=0}^{n-1} i \\
W(n) &= \frac{n(n-1)}{2} \\
W(n) &= \frac{1}{2}(n^2 - n)
\end{aligned}
$$

6. Exercise 3.35

Problem: Solve the following recurrence relations

a. $t(n) = 3t(n-1) + n, n \geq 1$, init. cond. $t(0) = 0$

$$
\begin{aligned}
t(n) &= 3t(n-1) + n \\
t(n) &= 3(3t(n-2) + n - 1) + n \\
&= 3^2 t(n-2) + 3(n-1) + n \\
t(n) &= 3^2(3t(n-3) + n - 2) + 3(n-1) + n \\
&= 3^3 t(n-3) + 3^2(n-2) + 3(n-1) + n \\
&\vdots \\
t(n) &= 3^k t(n-k) + 3^{k-1}(n-k+1) + \cdots + 3^2(n-2) + 3(n-1) + n
\end{aligned}
$$

When $k = n$,

$$t(n) = 3^n t(0) + 3^{n-1}(1) + 3^{n-2}(2) + \cdots + 3^2(n-2) + 3(n-1) + n$$

Applying the initial conditon, $t(0) = 0$,

$$
\begin{aligned}
t(n) &= 3^{n-1} + 3^{n-2}(2) + \cdots + 3^2(n-2) + 3(n-1) + n \\
t(n) &= 3^{n-1}\left(1 + \frac{2}{3} + \cdots + \frac{n-2}{3^{n-3}} + \frac{n-1}{3^{n-2}} + \frac{n}{3^{n-1}}\right)
\end{aligned}
$$

Note that for the sum of the first $n+1$ terms of a geometric series (where $r \neq 1$)

$$1 + r + r^2 + r^3 + \cdots + r^{n-1} + r^n = \frac{1 - r^{n+1}}{1 - r}$$

And differentiation both sides with respect to r yields:

$$0 + 1 + 2r + 3r^2 + \cdots + (n-1)r^{n-2} + nr^{n-1} = \frac{nr^{n+1} - (n+1)r^n + 1}{(1-r)^2}$$

And when $r = \frac{1}{3}$,

$$
\begin{aligned}
1 + \frac{2}{3} + \cdots + \frac{n-2}{3^{n-3}} + \frac{n-1}{3^{n-2}} + \frac{n}{3^{n-1}} &= \frac{n(\frac{1}{3})^{n+1} - (n+1)(\frac{1}{3})^n + 1}{(1 - \frac{1}{3})^2} \\
&= \frac{1}{4} \times \frac{1}{3^{n-1}} \times (-2n + 3^{n+1} - 3)
\end{aligned}
$$

Plugging that result back into the recurrence relation gives:

$$t(n) = 3^{n-1}\left(\frac{1}{4} \times \frac{1}{3^{n-1}} \times (-2n + 3^{n+1} - 3)\right)$$

Which reduces to:

$$t(n) = \frac{1}{4}(-2n + 3^{n+1} - 3)$$

b. $t(n) = 4t(n-1) + 5, n \geq 1$, init. cond. $t(0) = 2$

$$\begin{gathered}
t(n) = 4t(n-1) + 5 \\
t(n) = 4(4t(n-2) + 5) + 5 \\
= 4^2 t(n-2) + 4 \times 5 + 5 \\
t(n) = 4^2(4t(n-3) + 5) + 4 \times 5 + 5 \\
= 4^3 t(n-3) + 4^2 \times 5 + 4 \times 5 + 5 \\
\vdots \\
t(n) = 4^k t(n-k) + 4^{k-1} \times 5 + \cdots + 4^2 \times 5 + 4 \times 5 + 5
\end{gathered}$$

When $k = n$,

$$t(n) = 4^k t(0) + 4^{n-1} \times 5 + \cdots + 4^2 \times 5 + 4 \times 5 + 5$$

Applying the initial conditon, $t(0) = 2$,

$$\begin{gathered}
t(n) = 4^n \times 2 + 4^{n-1} \times 5 + \cdots + 4^2 \times 5 + 4 \times 5 + 5 \\
t(n) = 4^n \times 5 - 4^n \times 3 + 4^{n-1} \times 5 + \cdots + 4^2 \times 5 + 4 \times 5 + 5 \\
t(n) = 4^n \times 5\left(1 + \frac{1}{4} + \frac{1}{4^2} + \cdots + \frac{1}{4^{n-2}} + \frac{1}{4^{n-1}}\right) - 4^n \times 3 \\
t(n) = 4^n \times 5\left(\frac{1 - (\frac{1}{4})^n}{1 - \frac{1}{4}}\right) - 4^n \times 3
\end{gathered}$$

Which reduces to:

$$t(n) = \frac{1}{3}(4^n \times 11 - 5)$$

7. Exercise 3.43

a.

Problem: Prove by induction that

$$\begin{pmatrix} fib(n) \\ fib(n+1) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Start with the base case when $n = 0$. We want to show that $fib(0) = 0$ and $fib(1) = 1$. Note, for matrix A, $A^0 = I$, where I is the identity matrix:

$$\begin{pmatrix} fib(0) \\ fib(1) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^0 \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$
$$= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$
$$= \begin{pmatrix} 1 \times 0 + 0 \times 1 \\ 0 \times 0 + 1 \times 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$
$$\therefore fib(0) = 0, fib(1) = 1$$

Then, for the induction step, assume the claim is true for $n - 2$, i.e.:

$$\begin{pmatrix} fib(n-2) \\ fib(n-1) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^{n-2} \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Then:

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^2 \begin{pmatrix} fib(n-2) \\ fib(n-1) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^2 \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^{n-2} \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$
$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} fib(n-2) \\ fib(n-1) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$
$$\begin{pmatrix} 0 \times 0 + 1 \times 1 & 0 \times 1 + 1 \times 1 \\ 1 \times 0 + 1 \times 1 & 1 \times 1 + 1 \times 1 \end{pmatrix} \begin{pmatrix} fib(n-2) \\ fib(n-1) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$
$$\begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} fib(n-2) \\ fib(n-1) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$
$$\begin{pmatrix} fib(n-2) + fib(n-1) \\ fib(n-2) + 2fib(n-1) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

By definition, we have $fib(n) = fib(n-2) + fib(n-1)$, and it easy to show that the following is true.

$$fib(n+1) = fib(n-2) + 2fib(n-1)$$
$$= fib(n-2) + fib(n-1) + fib(n-1)$$
$$= fib(n-1) + fib(n)$$

Therefore,

$$\begin{pmatrix} fib(n) \\ fib(n+1) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

b.

Problem: Briefly describe how the preceding formula can be employed to design an algorithm for computing $fib(n)$ using only at most $8 \log_2 n$ multiplications.

Assume that $n = 2^k$. Then the exponentiated matrix, $\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n$, can be calculated using a modified version of the recursive `Powers` algorithm based on the left-to-right binary method. The `Powers` algorithm uses $\log_2 n$ multiplications to compute x^n. A key difference here is that we are doing matrix multiplication, which requires 8 multiplications to do one matrix multiplication (2 multiplications for each cell in a 2x2 matrix).

The value of $fib(n)$ is computed when the power on the matrix term is $n - 1$, it is the value in the lower cell of the resulting matrix product. This means the exponentiated matrix can be calculated with $8 \log_2 (n - 1)$ multiplications. But it takes another four multiplication steps to multiply by that matrix by $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$, therefore it takes at most $8 \log_2 n$ multiplications to calculate $fib(n)$.