Robert Owens (rao7utn)
CS 4102 Algorithms
Basic Written Homework: Divide and Conquer Sorting Basic

1 Secret Array

Preform a binary search where the function *f(l1, l2)*, where lists *l1* and *l2* are the indicies of the sub-arrays of the given array A of size n. The split for where sub-array *l1* ends and where *l2* begins will be determined by the 'binary search' like application of checking half the array that the previous iteration did.

Algorithm 1 pseudo-coded solution

```
procedure findTwoIndex(i, j)
                        ▷ let i be the beginning of the sub-array, and j be the end of the sub-array
    i = 0
    j = A.length − 1

    while i ≠ j do
        mid = (i + j)/2                                         ▷ floor division
                        ▷ A[beginning, end] creates a sub-array. Both arguments inclusive
        l1, l2                  ▷ Let l1 and l2 be lists of array indices to be passed to f
        for k from i to mid do                                  ▷ mid is inclusive
            l1.append(k)
        for k from mid+1 to j do                                ▷ j is inclusive
            l2.append(k)

        weight = f(l1, l2)
        if l1.length == l2.length then
            if weight == −1 then
                i = mid + 1
            else
                j = mid
        else
            if weight == 0 then
                i = mid + 1
            else
                j = mid
    Return i
```

The logic for the if else cases is as follows. If the lengths of the sub-arrays (indices) are equal then whichever sub-array has a greater length must contain the 2. Hence, *weight == -1* indicates that the *upper* sub-array must contain the 2. Otherwise, the 2 must exist in the *lower* sub-array. If it is not the case that lengths of the sub-arrays (indices) are equal then the *lower* sub-array must be 1 longer than the *upper* sub-array. Hence, if the sum of the *upper* and *lower* sub-arrays are equal (*weight == 0*) then it must be the case that the 2 exists in the *upper* sub-array since with 1 less element the sums are the same (one of the elements must be a 2 instead of a 1).

For the run-time complexity of this solution, while while loop performs $\theta(log(n))$ work, since in each iteration half the array is computed to *not* have the 2. Filling lists *l1* and *l2* does $\theta(n)$ work and the secret function f does $\Theta(n/2)$ work (Since Max(*l1*, *l2*)) is guaranteed to be n/2 by the nature of how this algorithm partitions the input. Finally since the filling list work and secret function work is preformed within the while loop the run time complexity is:

$$\theta(log(n) \cdot (n + \frac{n}{2}))$$

Which reduces to:

$$\theta(nlog(n))$$

2 Quicksort Worst-Case

a)

$$\frac{2}{n}$$

b)Restrict n to $n > 1$

$$\frac{2}{n} \cdot \frac{2}{n-1} \cdot \ldots \cdot \frac{2}{4} \cdot \frac{2}{3} \cdot \frac{2}{2} = \frac{2^n}{n!}$$

c)

$$\lim_{n \to \infty} (\frac{2^n}{n!})$$

$$\lim_{n \to \infty} (n!) > \lim_{n \to \infty} (2^n)$$

$$\therefore \lim_{n \to \infty} (\frac{2^n}{n!}) = 0$$

d) As the input to Quicksort grows the likely hood that Quicksort will execute in its worst case time complexity ($\Theta(n^2)$) is remarkably low; very very unlikely.

3 Unrolling Recurrence

$$T(n) = T(n-1) + n$$

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

$$T(n-2) = T(n-3) + n - 2$$

$$T(n) = T(n-3) + n - 2 + n - 1 + n$$

$$T(n) = T(n-3) + 3n - 2$$

$$\therefore T(n-k) = T(n-k) + kn - \frac{k(k-1)}{2}$$

let

$$k = n - 1$$

$$T(n) = T(1) + (n-1)n - \frac{(n-1)(n-2)}{2}$$

$$T(n) \in \theta(n^2)$$

4 Induction 1

Show that:

$$T(n) \in O(log(n) \cdot log(log())$$

$$2T(\sqrt{n}) + log(n) \leq log(n) \cdot log(log(n))$$

let

$$m = log(n) n = 2^m$$

let

$$Q(m) = T(2^m)$$

$$T(2^m) = 2T(n^{\frac{m}{2}}) + m = Q(m) = 2Q(\frac{m}{2}) + m$$

Show that:

$$2Q(\frac{m}{2}) + m \leq c \cdot log(n)$$

Proof by induction:
For m = 2

$$Q(2) = 2Q(1) + 2 \leq c \cdot 2log(2)$$

$$4 \leq 2 \cdot c$$

$$2 \leq c$$

Proven for base case
Inductive Hypothesis: For any $p \in \mathbb{R}$

$$Q(p) \leq c \cdot nlog(n) + n$$

For any $p + 1$

$$Q(p+1) = 2Q(\frac{p+1}{2}) + p + 1 \leq c \cdot nlog(n)$$

Because $\frac{p+1}{2} < p$ for large p

$$Q(p+1) \leq 2[c \cdot \frac{p+1}{2} \cdot log(\frac{p+1}{2})] + p + 1$$

$$Q(p+1) \leq c \cdot (p+1) \cdot log(p+1) - c \cdot (p+1) \cdot log(2) + p + 1$$

$$Q(p+1) \leq c \cdot (p+1) \cdot log(p+1) \leq c \cdot (p+1) \cdot log(p+1) - c \cdot (p+1) + p + 1$$

must be true for $c \geq 1$

$$\therefore T(n) = T(2^m) = Q(m) \in O(mlog(m)) \in O(log(n) \cdot log(log(n)))$$

$$T(n) \in O(log(n) \cdot log(log(n)))$$

$$Q.E.D.$$

5 Induction 2

Show that:

$$T(n) = 4T(\frac{n}{3}) + n \in \Theta(n^{\log_3(4)})$$

True if:

$$T(n) \in \Omega(n^{\log_3(4)}) \cap T(n) \in O(n^{\log_3(4)})$$

let $f(n) = n^{\log_3(4)}$
Prove:

$$T(n) \in O(f(n))$$
$$T(n) \leq c \cdot f(n)$$
$$T(n) - dn \leq c \cdot f(n) - n$$
$$4(c \cdot (\frac{n}{3})^{\log_3(4)} - dn) + n \leq c \cdot f(n) - n$$
$$\frac{4 \cdot c \cdot n^{\log_3(4)}}{3^{\log_3(4)}} - 4dn + n \leq c \cdot n^{\log_3(4)} - n$$
$$-4dn \leq -n$$
$$d \geq \frac{1}{4}$$
$$\therefore T(n) \in O(f(n))$$

Prove:

$$T(n) \in \Omega(f(n))$$
$$T(n) \geq c \cdot f(n)$$
$$T(n) + dn \geq c \cdot f(n) + n$$
$$4(c \cdot (\frac{n}{3})^{\log_3(4)} + dn) + n \geq c \cdot f(n) + n$$
$$\frac{4 \cdot c \cdot n^{\log_3(4)}}{3^{\log_3(4)}} + 4dn + n \geq c \cdot n^{\log_3(4)} + n$$
$$4dn \geq n$$
$$d \geq \frac{1}{4}$$
$$\therefore T(n) \in \Omega(f(n))$$
$$T(n) \in O(f(n)) \cap T(n) \in \Omega(f(n))$$
$$\therefore T(n) \in \Theta(n^{\log_3(4)})$$
$$Q.E.D.$$

6 Master Theorem 1

For

$$T(n) = 2T(\frac{n}{4}) + 1$$

$k = \log_4(2) = 0.5$ and $f(n) = 1$
By case 1 of the Master Theorem where $\epsilon = 0.5$

$$f(n) \in O(n^{k-\epsilon})$$
$$\therefore T(n) \in \Theta(\sqrt(n))$$

7 Master Theorem 2

For

$$T(n) = 2T(\frac{n}{4}) + \sqrt{n}$$

$k = \log_4(2) = 0.5$ and $f(n) = \sqrt{n}$
By case 2 of the Master Theorem

$$f(n) \in \Theta(n^k)$$
$$\therefore T(n) \in \Theta(\sqrt{n} \cdot log(n))$$

8 Master Theorem 3

For

$$T(n) = 2T(\frac{n}{4}) + n$$

$k = \log_4(2) = 0.5$ and $f(n) = n$
By case 3 of the Master Theorem where $\epsilon = 0.5$

$$f(n) \in \Theta(n^{k+\epsilon})$$

Check of Regularity Condition:

$$2f(n/4) \leq c \cdot f(n)$$
$$2[\frac{n}{4}] \leq c \cdot n$$
$$\frac{n}{2} \leq c \cdot n$$
$$\frac{1}{2} \leq c$$

Since $c < 1$ the regularity condition is satisfied

$$\therefore T(n) \in \Theta(n)$$

9 Master Theorem 4

For

$$T(n) = 2T(\frac{n}{4}) + n^2$$

$k = \log_4(2) = 0.5$ and $f(n) = n^2$
By case 3 of the Master Theorem where $\epsilon = 1.5$

$$f(n) \in \Theta(n^{k+\epsilon})$$

Check of Regularity Condition:

$$2f(n/4) \leq c \cdot f(n)$$
$$2[\frac{n^2}{4}] \leq c \cdot n^2$$
$$\frac{n^2}{2} \leq c \cdot n^2$$
$$\frac{1}{2} \leq c$$

Since $c < 1$ the regularity condition is satisfied

$$\therefore T(n) \in \Theta(n^2)$$

10 Honor Pledge

All above work is my own. However I worked on the problems with Christopher Osborne and Mac McLean within the parameters designated as acceptable by the professors.

On my honor as a student I have neither given nor received unauthorized aid on this assignment.

Robert Atticus Owens