Problem Set #3 CSC236 Fall 2018

Mengning Yang, Licheng Xu, Chenxu Liu

Nov 1st 2018

We declare that this assignment is solely our own work, and is in accordance with the University of Toronto Code of Behaviour on Academic Matters.

This submission has been prepared using LaTeX.

Problem 1.

(Warmup - this problem will NOT be marked).

Consider the following recurrence that results from some unspecified divide-and-conquer algorithm, where k is a positive constant and $1 \leq z \leq m-1$:

$$T(m,n) = \begin{cases} km, & n \leq 2 \\ kn, & m \leq 2 \\ kmn + T(z, n/2) + T(m-z, n/2), & m, n > 2 \end{cases}$$

Use **induction** to prove a Theta bound for $T(m,n)$. Do **NOT** use the substitution method. To help you guess the Theta bound, here are two possibilities; perhaps one of these is correct: $T(m,n) = \Theta(mn), T(m,n) = \Theta(m^2n^2)$.

Problem 2.

(6 Marks) Consider the following three methods of solving a particular problem (input size n):

1. You divide the problem into three subproblems, each $\frac{1}{5}$ the size of the original problem, solve each recursively, then combine the results in time linear in the original problem size.

2. You divide the problem into 16 subproblems, each $\frac{1}{4}$ of size of the original problem, solve each recursively, then combine the results in time quadratic in the original problem size.

3. You reduce the problem size by 1, solve the smaller problem recursively, then perform an extra "computation step" that requires linear time.

Assume the base case has size 1 for all three methods.

For each method, write a recurrence capturing its worst-case runtime. Which of the three methods yields the fastest asymptotic runtime?

In your solution, you should use the Master Theorem wherever possible. In the case where the Master Theorem doesn't apply, *clearly state why not* based on your recurrence, and show your work solving the recurrence using another method (no proofs required).

1. $T(n) = 3T(\frac{n}{5}) + \theta(n)$

   By Master Theorem, $a = 3, b = 5, k = 1, \log_b a = \log_5 3 < k$ which satisfies the third case of Master Theorem, therefore, $T(n) = \mathcal{O}(n)$

2. $T(n) = 16T(\frac{n}{4}) + \theta(n^2)$

   By Master Theorem, $a = 16, b = 4, k = 2, \log_b a = \log_4 16 = 2 = k$ which satisfies the first case of Master Theorem, therefore, $T(n) = \mathcal{O}(n^2 \log n)$

3. $T(n) = T(n-1) + \theta(n)$

   which doesn't satisfy the form of Master Theorem, because the T(n-1) is not in the form of $aT(\frac{n}{b})$, so we have to use repeated substitution to find the closed form first.

$$
\begin{aligned}
T(n) &= T(n-1) + n \text{ when k} = 1 \\
T(n) &= T(n-2) + 2n \text{ when k} = 2 \\
T(n) &= T(n-3) + 3n \text{ when k} = 3 \\
T(n) &= T(n-k) + kn \\
\text{Let } n-k &= 1 \\
k &= n-1 \\
T(n) &= T(1) + n(n-1) \\
T(n) &= constant + n(n-1) \\
Therefore, T(n) &= O(n^2)
\end{aligned}
$$

Problem 3.

(8 MARKS) (Modelled after Exercise 14 from lecture notes, p.48).

Recall the recurrence for the worst case runtime of quicksort:

$$T(n) = \begin{cases} c, & \text{if } n \leq 1 \\ T(|L|) + T(|G|) + dn, & \text{if } n > 1 \end{cases}$$

where L and G are the partitions of the list. Clearly, how the list is partitioned matters a great deal for the runtime of quicksort.

1. Suppose the lists are split as follows: $|L| = \frac{n}{4}, |G| = \frac{3n}{4}$ at each recursive call.

   Find a tight asymptotic bound on the runtime of quicksort using this assumption.

2. Now suppose that the lists are always very unevenly split: $|L| = n - 4$ and $|G| = 3$ at each recursive call for $n > 4$. Find a tight asymptotic bound on the runtime of quicksort using this assumption.

1. Beacuse of $|L| = \frac{n}{4}$ and $|G| = \frac{3n}{4}$
Then, $T(n) = T(\frac{n}{4}) + T(\frac{3n}{4}) + dn$
plug it in

$$
\begin{aligned}
T(n) &= T(\frac{n}{4}) + T(\frac{3n}{4}) + dn \\
&= T(\frac{n}{16}) + T(\frac{3n}{16}) + T(\frac{3n}{16}) + T(\frac{9n}{16}) + d\frac{n}{4} + d\frac{3n}{4} \\
&= T(\frac{n}{16}) + T(\frac{3n}{16}) + T(\frac{3n}{16}) + T(\frac{9n}{16}) + dn \\
&= T(\frac{n}{64}) + T(\frac{3n}{64}) + T(\frac{3n}{64}) + T(\frac{9n}{64}) + T(\frac{3n}{64}) + T(\frac{9n}{64}) + T(\frac{9n}{64}) + T(\frac{27n}{64}) + dn \\
&= ..... \\
&= T(\frac{n}{4^i} = 1) + ... + T(\frac{3^i n}{4^i}) + dn \\
&= T(\frac{n}{4^i} = 1) + ... + T(\frac{3^{i+1} n}{4^{i+1}}) + dn + d^{'} n \quad where \quad d^{'} n < dn \\
&= ..... \\
&= T(\frac{n}{4^{i_1}} = 1) + T(\frac{3n}{4^{i_2}} = 1) + T(\frac{3^2 n}{4^{i_3}} = 1) + ... + T(\frac{3^k n}{4^k} = 1) + cn
\end{aligned}
$$

$T(\frac{3x}{4})$ decreases slower than $T(\frac{x}{4})$, so there are at most total k steps and

$$
\begin{aligned}
\frac{3^k n}{4^k} &= 1 \\
\frac{n}{4/3^k} &= 1 \\
(\frac{4}{3})^k &= n \\
k &= \log_{\frac{4}{3}} n
\end{aligned}
$$

so $T(n) \leq cn \log_{\frac{4}{3}} n = O(n \log n)$

2. Similarly, we can get the following:

$$T(n) = T(n-4) + T(3) + dn$$
$$T(n-4) = T(n-8) + T(3) + d(n-4)$$

plug it in

$$\begin{aligned} T(n) &= T(n-8) + 2T(3) + dn + d(n-4), \quad k=1 \\ &= T(n-12) + 3T(3) + dn + d(n-4) + d(n-8), \quad k=2 \\ &= T(n-16) + 4T(3) + dn + d(n-4) + d(n-8) + d(n-12), \quad k=3 \\ &= ..... \\ &= T(n-4(k+1)) + (k+1)T(3) + (k+1)dn - \sum_{i=1}^{k} 4d \end{aligned}$$

base case:

$$\begin{aligned} n - 4(k+1) &\leq 4 \\ k &\geq \frac{n-8}{4} \end{aligned}$$

because n-4(k+1) has linear complexity, being similar or equal to 4 will not affect the complexity. So we can treat the base case: $k = \frac{n-8}{4}$.
therefore:

$$\begin{aligned} T(n) &= T(n \leq 4) + (\frac{n-8}{4} + 1)T(3) + (\frac{n-8}{4} + 1)dn - \sum_{i=1}^{\frac{n-8}{4}} 4d \\ &= T(n \leq 4) + (\frac{n-8}{4} + 1)T(3) + \Theta(n^2) + \Theta(n) \\ &= \Theta(n^2) \end{aligned}$$

Problem 4.

(8 MARKS)

Video rankings. The InstaVid social network collects user preferences by asking them to rank their favorite videos. One of the features of InstaVid is offering friendship suggestions for users with similar tastes, using the following metric.

If user *One* ranks the videos using the sequence $1, 2, \ldots, n$, and user *Two* ranks same videos using the sequence $v_1, v_2, \ldots, v_n$ of numbers $1, 2, \ldots, n$, then their social distance is computed by counting all pairs (v_i, v_j) from the ranking of *Two* that satisfy the condition $v_i > v_j$ for $i < j$. For example, if the user *One* ranks four videos as $1, 2, 3, 4$ and *Two* ranks them as $3, 1, 2, 4$ then their social distance is 2 because of the pairs $(3, 1)$ and $(3, 2)$ in the rankings of *Two*.

(i) Design an algorithm `social_distance(prefa, prefb)` with time complexity in $\Theta(n^2)$ that computes the social distance for users with video rankings `prefa` and `prefb`. Please write your solution in the form of a Python function. Justify the run time of your code.

(ii) Improve your algorithm using divide and conquer approach. Fully analyse your algorithm; you may use the Master Theorem. Write your solution in the form of a Python function. (You may also write pseudocode if desired; we will not run your code anyway, but it is aceptable to actually implement your code in Python and copy it in your soluton.)

1.
```
def social_distance(prefa, prefb):
    acc1 = {}
    acc2 = []
    num = 0
    for i in range(len(prefa)):
        acc1[prefa[i]] = i
    for j in prefb:
        acc2.append(acc1[j])
    for n in range(len(acc2)):
        for m in acc2[n+1:]:
            if acc2[n] > m:
                num += 1
    return num
```

For this function, it has a nested for loop, so we have

$$\sum_{i=1}^{n}\sum_{i=2}^{n}1+\sum_{i=1}^{n}1+\sum_{i=1}^{n}1$$

which gives us n^2+n+n.
Therefore, the overall runtime complexity for this function is $\Theta(n^2)$.

2.
```
def social_distance(prefa, prefb):
    ranka = {}
    for i in range(len(prefa)):
        ranka[prefa[i]] = i

    rankb = []
    for j in prefb:
        rankb.append(ranka[j])

    return distance(rankb)

def comb(lst1,lst2):
    comb_num = 0
    sort_lst1 = lst1
    sort_lst2 = lst2
    sort_lst1.sort()
    sort_lst2.sort()
    x = 0
    y = 0
    while x < len(sort_lst1) and y < len(sort_lst2):
        if sort_lst1[x] > sort_lst2[y]:
            comb_num += len(sort_lst1)
            y += 1
        else:
            x += 1
    return comb_num

def distance(rankb):
    if len(rankb) <= 1: #base case1
```

```
    return 0
elif len(rankb) == 2:#base case2
    if rankb[0] > rankb[1]:
        return 1
    else:
        return 0
else:
    mid = len(rankb)//2
    left = rankb[0:mid]
    right = rankb[mid:]
    return distance(left) + distance(right) + comb(left,right)
```

For this part, the helper function comb(lst1,lst2) has a runtime complexity of nlogn, the other helper function which calls comb(lst1,lst2) has a runtime complexity of $2T(n/2) + nlogn$. Because this recurrence form does not satisfy the form of Master Theorem, so we have to use other methods to prove it.

Let us take $n = 2^m$. Then we have the recurrence:
$T(2^m) = 2T(2^{m-1}) + 2^m log(2^m) = 2T(2^{m-1}) + m2^m$
Calling $T(2^m)$ as $f(m)$, we get that:

$$\begin{aligned} f(m) &= 2f(m-1) + m2^m \\ &= 2(2f(m-2) + (m-1)2^{m-1}) + m2^m \\ &= 4f(m-2) + (m-1)2^m + m2^m \\ &= 4(2f(m-3) + (m-2)2^{m-2}) + (m-1)2^m + m2^m \\ &= 8f(m-3) + (m-2)2^m + (m-1)2^m + m2^m \end{aligned}$$

Proceeding on these lines, we get that:

$$\begin{aligned} f(m) &= 2^m f(0) + 2^m(1 + 2 + 3 + \ + m) \\ &= 2^m f(0) + \frac{m(m+1)}{2}2^m \\ &= 2^m f(0) + m(m+1)2^{m-1} \\ \text{Hence, } T(n) &= nT(1) + n(\frac{log(n)(1+log(n))}{2}) \\ &= \theta(nlogn) \end{aligned}$$

Therefore, overall the runtime complexity is $\theta(nlogn)$.