Name: Qiuyang Fu

ID: 108667420

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

Homework 2A (45 points) **Summer 2020, CU-Boulder**

Advice 1: For every problem in this class, you must justify your answer: show how you arrived at it and why it is correct. If there are assumptions you need to make along the way, state those clearly.

Advice 2: Verbal reasoning is typically insufficient for full credit. Instead, write a logical argument, in the style of a mathematical proof.

Instructions for submitting your solution:

- The solutions **should be typed**, we cannot accept hand-written solutions. Here's a short intro to **Latex**.
- In this homework we denote the asymptomatic *Big-O* notation by $\mathcal{O}$ and *Small-O* notation is represented as o.
- We recommend using online Latex editor **Overleaf**. Download the **.tex** file from Canvas and upload it on overleaf to edit.
- You should submit your work through **Gradescope** only.
- If you don't have an account on it, sign up for one using your CU email. You should have gotten an email to sign up. If your name based CU email doesn't work, try the identikey@colorado.edu version.
- Gradescope will only accept **.pdf** files (except for code files that should be submitted separately on Canvas if a problem set has them) and **try to fit your work in the box provided**.
- You cannot submit a pdf which has less pages than what we provided you as Gradescope won't allow it.

Name:	Qiuyang Fu
ID:	108667420

CSCI 3104, Algorithms **Escobedo & Jahagirdar**
Homework 2A (45 points) **Summer 2020, CU-Boulder**

Piazza threads for hints and further discussion

Piazza Threads
Question 1
Question 2
Question 3
Question 4
Question 5

Recommended reading: Divide & Conquer; Recurrence Relations: Ch. 2 2.3; Ch. 4 4.1, 4.2, 4.3, 4.4, 4.5. The three methods for solving recurrences from **4.3 - 4.5** is important.

Name: Qiuyang Fu

ID: 108667420

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

Homework 2A (45 points) **Summer 2020, CU-Boulder**

1. *(5 × 4 = 20 pts) Solve the following recurrence relations. For each case, show your work.*

   (a) $T(n) = 3T(n-1) + 1$ *if* $n > 1$ *and* $T(1) = \Theta(1)$

   It is obvious that the recurrence relations can be showed as below:

$$\begin{aligned} T(n) &= 3T(n-1) + 1 \\ &= 9T(n-2) + 4 \\ &= 27T(n-3) + 13 \\ &= 81T(n-4) + 40 \\ &= ... \\ &= 3^{n-1}T(1) + \frac{3^{n-1} - 1}{2} \end{aligned} \tag{1}$$

   Since $T(1) = \Theta(1)$, so $T(n) = \Theta(3^n)$.

Name: Qiuyang Fu

ID: 108667420

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

Homework 2A (45 points) **Summer 2020, CU-Boulder**

(b) $T(n) = 2T(\frac{n}{3}) + \Theta(n)$ *if* $n > 1$ *and* $T(1) = \Theta(1)$

Using master theorem, it is obvious that $a = 2$, $b = 3$, $f(n) = \Theta(n)$, since $f(n)$ is larger then $n^{log_b a} = n^{log_3 2}$, so $T(n) = \Theta(n)$.

Name: Qiuyang Fu

ID: 108667420

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

Homework 2A (45 points) **Summer 2020, CU-Boulder**

(c) $T(n) = 3T(\frac{n}{2}) + \Theta(n)$ *if* $n > 1$ *and* $T(1) = \Theta(1)$

Using master theorem, it is obvious that $a = 3$, $b = 2$, $f(n) = \Theta(n)$, since $f(n)$ is smaller then $n^{log_b a} = n^{log_2 3}$, so $T(n) = \Theta(n^{log_2 3})$.

(d) $T(n) = 4T(\frac{n}{2}) + \Theta(n^2)$ *if* $n > 1$ *and* $T(1) = \Theta(1)$

Using master theorem, it is obvious that $a = 4$, $b = 2$, $f(n) = \Theta(n^2)$, since $f(n) = \Theta(n^2)$ then $T(n) = \Theta(n^2 \log n)$.

Name: Qiuyang Fu

ID: 108667420

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

Homework 2A (45 points) **Summer 2020, CU-Boulder**

2. *(5 pts) Can master's theorem be applied to solve the following recurrence* $T(n) = 8T(\frac{n}{2}) + n^3 \log(n)$*. If* $n > 1$ *and* $T(1) = \Theta(1)$

   Prove that the above recurrence cannot be solved using Master's theorem by showing that none of the three conditions required to apply the theorem are satisfied by the recurrence.

The recurrence cannot be solved using Master's theorem since it is obvious that $a = 8$, $b = 2$, $f(n) = n^3 \log(n)$. Here we can get $n^{\log_b^a} = n^3$, but the relation between $f(n)$ and $n^{\log_b^a}$ is not polynomially smaller or larger which cannot fit into the first and third cases in Master's theorem. The ratio $f(n)/n^{\log_b^a} = \log(n)$ is asymptotically less than n^ϵ for any positive constant ϵ. Consequently, the recurrence falls to solve using Master's theorem.

Name: Qiuyang Fu

ID: 108667420

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

Homework 2A (45 points) **Summer 2020, CU-Boulder**

3. *($5 \times 2 = 10$ pts) Consider the following functions. For each of them, determine how many times is 'hi' printed in terms of the input n (i.e in Asymptotic Notation of n). You should first write down a recurrence and then solve it using the* ***recursion tree method****. That means you should write down the first few levels of the recursion tree, specify the pattern, and then solve.*

(a)
```
def fun(n) {
    if (n > 1) {
        print( 'hi' 'hi' 'hi' )
        fun(n/3)
        fun(n/3)
    }
}
```

In this problem, the recurrence relation is $T(n) = 2T(\frac{n}{3}) + 3$, using recursion tree method, the recurrence tree is showed as below. From the tree, we can get that the subproblem size for a node at depth i is $n/3^i$. Thus, the tree has $\log_3 n + 1$ levels and $2^{\log_3 n}$ leaf. The total number of 'hi' printed at depth i, for $i = 0, 1, 2, ..., \log_3 n - 1$, is $2^i \times 3$. Finally, the total number of 'hi' printed is $\sum_{i=0}^{\log_3 n-1} 2^i \times 3 = 3 \times (2^{\log_3^n} - 1)$.

Name: Qiuyang Fu

ID: 108667420

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

Homework 2A (45 points) **Summer 2020, CU-Boulder**

(b)
```
def fun(n) {
    if (n > 1) {
        for i=1 to n {
            print( ‘hi’ ‘hi’ )
        }
        fun(n/3)
        fun(n/3)
    }
}
```

In this problem, the recurrence relation is $T(n) = 2T(\frac{n}{3}) + 2n$, using recursion tree method, the recurrence tree is showed as below. From the tree, we can get that the subproblem size for a node at depth i is $n/3^i$. Thus, the tree has $\log_3 n + 1$ levels and $2^{\log_3 n}$ leaf. The total number of 'hi' printed at depth i, for $i = 0, 1, 2, ..., \log_3 n - 1$, is $2^i(n/3^i) \times 2 = (2/3)^i n \times 2$. Finally, the total number of 'hi' printed is $\sum_{i=0}^{\log_3 n-1}(2/3)^i n \times 2 = 6n \times (1 - (2/3)^{\log_3^n})$.

Name: Qiuyang Fu

ID: 108667420

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

Homework 2A (45 points) **Summer 2020, CU-Boulder**

4. *(5 pts) Let $T(n) = 3T(\frac{n}{3}) + n$, where $T(n)$ is constant when $n \leq 3$. Using unrolling, determine tight asymptotic bounds for $T(n)$. That is, find a function $g(n)$ such that $T(n) \in \Theta(g(n))$. Clearly show all your work*

After unrolling the recurrence, the relation can be showed as below:

$$\begin{aligned} T(n) &= 3T(n/3) + n \\ &= 9T(n/9) + 2n \\ &= 27T(n/27) + 3n \\ &= 81T(n/81) + 4n \\ &= ... \\ &= nT(1) + n\log_3 n \end{aligned} \tag{2}$$

From the unrolling result above, it is obvious that $g(n) = n\log_3 n$.

Name: Qiuyang Fu

ID: 108667420

CSCI 3104, Algorithms **Escobedo & Jahagirdar**
Homework 2A (45 points) **Summer 2020, CU-Boulder**

5. *(5 pts) Consider the following pseudo-code.*

```
def fun(A[1,2,3....n]) {
    if A.length == 0:
        return 0
    count = 1
    for i=1 to n:
        for j=1 to n:
            count++
    return 1 + fun(A[1,2,3.....n-2])
}
```

Find a recurrence for the worst-case runtime complexity of this algorithm. Then solve your recurrence and get a tight bound on the worst-case runtime complexity.

From the code it is obvious that the worst-case recurrence relation is showed as $T(n) = T(n-2) + n^2$. Using unrolling method, the worst-case runtime complexity is $T(n) = T(1) + 1^2 + 3^2 + 5^2 + ... + n^2 = O(n^3)$.

Name: Qiuyang Fu

ID: 108667420

CSCI 3104, Algorithms
Homework 2A (45 points)

Escobedo & Jahagirdar
Summer 2020, CU-Boulder

6. ***Extra Credit (5% of total homework grade)*** *For this extra credit question, please refer the leetcode link provided below or click here. Multiple solutions exist to this question ranging from brute force to the most optimal one. Points will be provided based on Time and Space Complexities relative to that of the most optimal solution.*

   Please provide your solution with proper comments which carries points as well.

   https://leetcode.com/problems/maximum-subarray/

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        if(nums.size() == 0){
            return 0;
        }
        int start = 1;
        int maxSum = nums[0];
        int states[nums.size()];
        // using dp to record the previous states avoiding unnecessary calculation
        states[0] = nums[0];
        while(start < nums.size()){
           // record the state of current position
            states[start] = states[start-1] > 0 ?
            (states[start-1] + nums[start]):(nums[start]);
            // update the max sum if necessary
            if(states[start] > maxSum){
                maxSum = states[start];
            }
            start++;
        }
        return maxSum;
    }
};
```