Name: Daniel Kim
ID: 102353420

CSCI 3104, Algorithms **Profs. Chen & Grochow**
Problem Set 3 – Due Thurs Feb 6 11:55pm **Spring 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: Informal reasoning is typically insufficient for full credit. Instead, write a logical argument, in the style of a mathematical proof.

Instructions for submitting your solutions:

- All submissions must be easily legible.
- You should submit your work through the **class Canvas page** only.
- You may not need a full page for your solutions; pagebreaks are there to help Gradescope automatically find where each problem is. Even if you do not attempt every problem, please allot at least as many pages per problem (or subproblem) as are allotted in this template.

Quicklinks: 1a 1b 2a 2b 3 4

---

1. Solve the following recurrence relations. For each case, show your work.

   (a) $T(n) = 2T(n-1) + 1$ if $n > 1$, and $T(1) = 2$.

   Using substitution:
   1. $T(n) = 2T(n-1) + 1$
   2. $T(n-1) = 2T(n-2) + 1$
   3. $T(n) = 2(2T(n-2) + 1) + 1$ (Plug in 2 to 1)
   3. $T(n) = 4T(n-2) + 2^1 + 2^0$
   4. $T(n-2) = 2T(n-3) + 1$ 5. $T(n) = 4(2T(n-3) + 1) + 2^1 + 2^0$ (Plug in 4 to 3) 5. $T(n) = 8T(n-3) + 2^2 + 2^1 + 2^0$
   Pattern: $T(n) = 2^k T(n-k) + \sum_{j=0}^{k-1} 2^j$

   Using $\sum_{i=0}^{k-1} ar^i = \frac{a(r^k-1)}{r-1}$
   $2^k T(n-k) + \frac{2^k-1}{2-1}$
   Using base case to solve k $T(1) = 2 >> n - k = 1 >> k = n - 1$

Plug in k
$T(n) = 2^{n-1}T(n-(n-1)) + (2^{n-1}-1)$
$T(n) = 2^{n-1}T(1) + (2^{n-1}-1)$
$T(n) = 2^n + (2^{n-1}-1)$

We can can conclude that $T(n) = \Theta(2^n)$ because 2^n is the only n term and it's the highest function.

Name: Daniel Kim
ID: 102353420

(b) $T(n) = 3T(\frac{n}{2})+\Theta(n)$ if $n > 1$, and $T(1) = \Theta(1)$. Use the plug-in/substitution/unrolling method.
Let $\Theta(n) = Cn$
1. $T(n) = 3T(\frac{n}{2}) + Cn$
2. $T(\frac{n}{2}) = 3T(\frac{n}{4}) + C\frac{n}{2}$
3. $T(n) = 3(3T(\frac{n}{4}) + C\frac{n}{2}) + Cn$ (Plug 2 to 1)
3. $T(n) = 9T(\frac{n}{4}) + Cn\frac{3}{2} + Cn$
4. $T(\frac{n}{4}) = 3T(\frac{n}{8}) + C\frac{n}{4}$
5. $T(n) = 9(3T(\frac{n}{8}) + C\frac{n}{4}) + Cn\frac{3}{2} + Cn$ (Plug 4 to 3)
5. $T(n) = 27T(\frac{n}{8}) + Cn\frac{9}{4} + Cn\frac{3}{2} + Cn$

Pattern: $T(n) = 3^kT(\frac{n}{2^k}) + \sum_{j=0}^{k-1} Cn(\frac{3}{2})^j$
Using $\sum_{i=0}^{k-1} ar^i = \frac{a(r^k-1)}{r-1}$
$3^kT(\frac{n}{2^k}) + Cn\frac{(\frac{3}{2})^k-1}{\frac{3}{2}-1} = 3^kT(\frac{n}{2^k}) + Cn(2(\frac{3}{2})^k - 1)$

Using base case to solve k: $T(1) = Cn >> \frac{n}{2^k} = 1 >> \log(1) = \log(\frac{n}{2^k})$
$>> 0 = \log(n) - k\log(2) >> k\log(2) = \log(n) >> k = \frac{log(n)}{\log(2)} >>$
$k = \log_2(n)$

Plug in k
$T(n) = 3^{\log_2(n)}T(\frac{n}{2^{\log_2(n)}}) + Cn(2(\frac{3}{2})^{\log_2(n)} - 1)$
$T(n) = 3^{\log_2(n)}T(1) + Cn(2(\frac{3}{2})^{\log_2(n)} - 1)$
$T(1) = \Theta(1)$, $\Theta(1)$ means that they are constants and the same as 1
$T(n) = 3^{\log_2(n)} * 1 + Cn(2(\frac{3}{2})^{\log_2(n)} - 1)$
$T(n) = 3^{\log_2(n)} + Cn(2(\frac{3}{2})^{\log_2(n)} - 1)$
$T(n) = 3^{\log_2(n)} + 2Cn(\frac{3}{2})^{\log_2(n)} - 2Cn$
$T(n) = 3^{\log_2(n)} + 2C3^{\log_2(n)} - 2Cn$
$T(n) = 3C3^{\log_2(n)} - 2Cn$

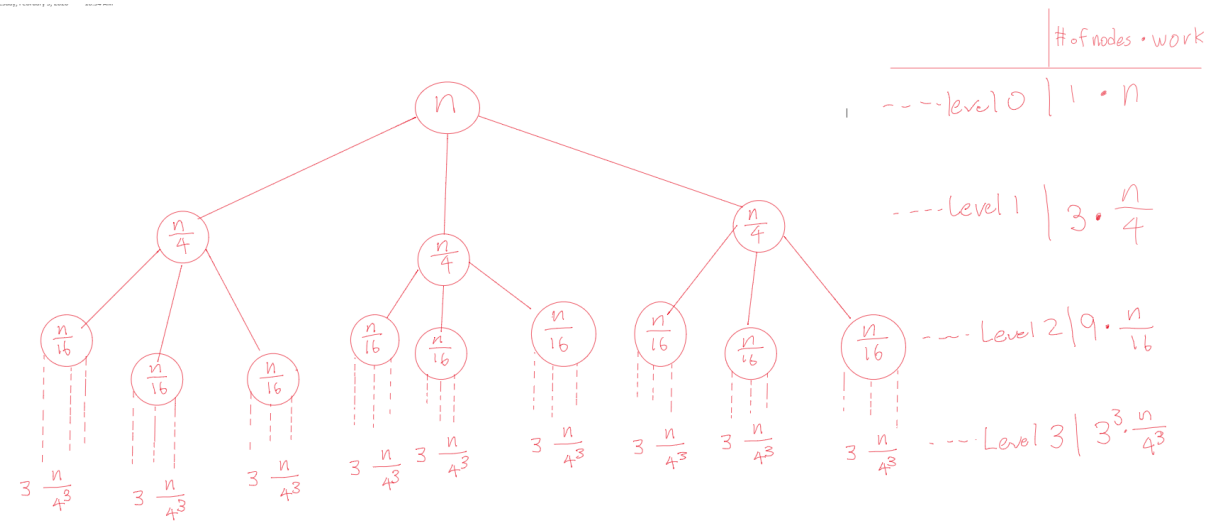
We can conclude that $T(n) \leq O(3^{\log_2(n)})$ because $O(3^{\log_2(n)})$ is growing the fastest and faster than n variable

Name: Daniel Kim
ID: 102353420

CSCI 3104, Algorithms **Profs. Chen & Grochow**
Problem Set 3 – Due Thurs Feb 6 11:55pm **Spring 2020, CU-Boulder**

2. Consider the following functions. For each of them, determine how many times is 'hi' printed in terms of the input 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/4)
         fun(n/4)
         fun(n/4)
}}
```



From the recursion tree diagram, you can see in level 1, there 3 nodes with $\frac{n}{4}$, in level 2, 9 nodes with $\frac{n}{16}$, in level 3, 27 nodes with $\frac{n}{4^3}$. In k term or k-th level, $3^k \frac{n}{4^k}$
So this is the recurrence relations taken from the diagram.
Recurrence relations: $T(n) = 3T(\frac{n}{4^k}) + 1; n > 1, T(1) = 0$
Summation: $T(n) = 1 + 3 + 3^2 + 3^3 + 3^4 ... 3^k = \sum_{j=0}^{k} 3^j$

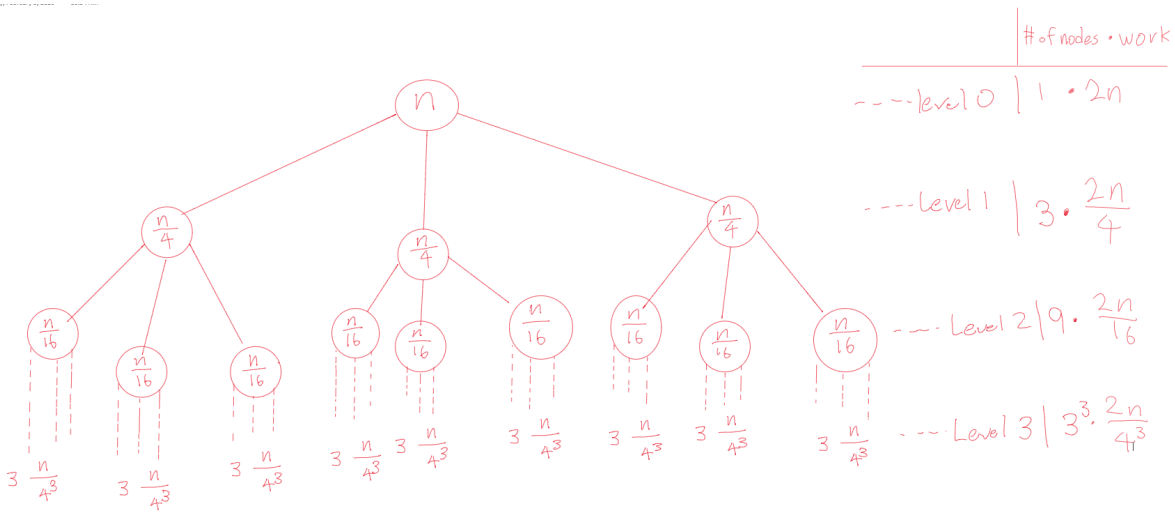
Solving for k, using the base case $T(1) = 0$, $\frac{n}{4^k} = 1 >> n = 4^k >> k = \log_4 n$
Using $\sum_{i=0}^{k-1} ar^i = \frac{a(r^k-1)}{r-1}$

$\sum_{j=0}^{k} 3^j = \frac{3^{k+1}-1}{3-1} = \frac{3^{k+1}-1}{2}$

Plug in k
$= \frac{1}{2}(3^{\log_4 n+1} - 1) = \frac{1}{2}(3 * 3^{\log_4 n} - 1)$
$T(n) = \frac{1}{2}(3 * n^{\log_4 3} - 1)$

We can conclude that $T(n) = \theta(n^{\log_4 3})$ because $n^{\log_4 3}$ is only n variable and the fastest growing function

(b)
```
def fun(n) {
      if (n > 1) {
         for i=1 to n {
           print( 'hi' 'hi' )
         }
         fun(n/4)
         fun(n/4)
         fun(n/4)
}}
```



From the recursion tree diagram, it is similar to the diagram from problem 2 a) but there there is $2n$ in work because of the for loop and printing two 'hi's. So you can see in level 1, there 3 nodes with $\frac{2n}{4}$, in level 2, 9 nodes with $\frac{2n}{16}$, in level 3, 27 nodes with $\frac{2n}{4^3}$. In k term or k-th level, $3^k\frac{2n}{4^k}$. In addition, $\frac{2n}{4^k}$ is included in summation because of the for loop. So this is the recurrence relations taken from the diagram.
Recurrence relations: $T(n) = 2n + 3T(\frac{2n}{4^k}); n > 1, T(1) = 0$
Summation: $T(n) = (1 + 3 + 3^2 + 3^3 + 3^4 ... 3^k) * \frac{2n}{4^k} = \sum_{j=0}^{k} 3^j \frac{2n}{4^j}$

Solving for k, using the base case $T(1) = 0$, $\frac{n}{4^k} = 1 >> n = 4^k >> k = \log_4 n$
Using $\sum_{i=0}^{k-1} ar^i = \frac{a(1-r^k)}{1-r}$

$\sum_{j=0}^{k} 3^j \frac{2n}{4^j} = \sum_{j=0}^{k} \frac{3^j}{4^j} * 2n = \sum_{j=0}^{k} 2n(\frac{3}{4})^j = \frac{2n(1-(\frac{3}{4})^{k+1})}{1-\frac{3}{4}} = \frac{2n(1-\frac{3}{4}*(\frac{3}{4})^k)}{\frac{1}{4}}$

$= 8n(1 - \frac{3}{4} * (\frac{3}{4})^k)$

Plug in k

$8n(1 - \frac{3}{4} * (\frac{3}{4})^{\log_4 n}) = 8n(1 - n^{\log_4 \frac{3}{4}}) = 8n - 6n * n^{\log_4 \frac{3}{4}}$

By comparing with $n^{log_4 \frac{3}{4}}$ and n, We can conclude that $T(n) \leq O(n)$ because n is grows faster than $n^{\log_4 \frac{3}{4}}$

Name: Daniel Kim
ID: 102353420

3. Consider the following algorithm

```
fun(A[1, ..., 4n]):
    if A.length == 0:
        return 0
    return 1 + fun(A[3, ..., 4n-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.

We know that the size of array A is $4n$ from line 1. In the return statement in line 4, we know that the size of the array is $4n-4$ because it is return from 3 to $4n-2$ so subtracting 4n by 4 total. $(A[1], A[2], A[4n-1], A[4n-2])$. For the base case, we know that something happens when A.length is 0 which concludes that $T(0) = C$ or $\theta(1), C$ for some constant. So using these facts, we can say this for recurrence.
Recurrence relations: $T(4n) = T(4n-4) + C$
Using substitution:
Let $a = 4n$
1. $T(a) = T(a-4) + C_0$
2. $T(a-4) = T(a-8) + C_1$
3. $T(a) = T(a-8) + C_1 + C_2$ (Plug 2 to 1)
4. $T(a-8) = T(a-12) + C_2$
5. $T(a) = T(a-12) + C_2 + C_1 + C_0$ (Plug 4 to 3)

Pattern: $T(a) = T(a-4k) + C * k$
Using the base case to solve k: $T(0) = C$ or $\theta(1) >> a - 4k = 0 >> -4k = -a >> k = \frac{a}{4}$

Plug in k
$T(a) = T(a - 4(\frac{a}{4})) + C * \frac{a}{4}$
$T(a) = T(0) + C * \frac{a}{4}$

Plug in a
$T(4n) = T(0) + C * n$
We can conclude that the tight bound on the worst-case runtime complexity is $T(4n) = \theta(n)$ because n is the only n - term there is besides the constants

Name: Daniel Kim
ID: 102353420

4. (Recall Problem 4 in Problem Set 1) Given an array $A = [a_1, a_2, \cdots, a_n]$, a reverse is a pair (a_i, a_j) such that $i < j$ but $a_i > a_j$. Design a divide-and-conquer algorithm with a runtime of $O(n \log n)$ for computing the number of reverses in the array. Your solution to this question needs to include both a written explanation and an implementation of your algorithm, including:

   (a) Your algorithm has to be a divide and conquer algorithm that is modified from the Merge Sort algorithm. Explain how your algorithm works, including pseudocode.

   (b) Implement your algorithm in Python, C, C++, or Java. **You MUST submit a runnable source code file. You will not receive credit if we cannot compile your code. Do NOT simply copy/paste your code into the PDF.**

   (c) Randomly generate an array of 100 numbers and use it as input to run your code. Report on both the input to and the output of your code.

∗ ∗ ∗ Used Geeks for Geeks: Count Inversions in an array | Set 1 (Using Merge Sort)
link: $https://www.geeksforgeeks.org/counting-inversions/?fbclid=IwAR01fLWxIieTYTrmST-bf0Cm4$ as references ∗ ∗ ∗
a)

```
Pseudocode:
// This functions copies to temp array with input array and return the number of
   inversions happens in the array
int sortMerged (array, array.length) // Assign temporary array with
                                        array.length size and
                                        return mergeSort function

// Needs two helper functions to make sortMerged happen.
// Recursive function that sorts the sub arrays and keeps returning
// number of inversions
int mergeSort (array, temp array, left beginning, right end)
    if right index is bigger than left index, divide the array into two parts.
    Use recursive function mergeSort till right is less than left.
    Call helper function merge to merge back two parts
```

Name: Daniel Kim
ID: 102353420


```
// Helper function to merge back
int merge (array, temp array, first index of sub array 1, middle index of array,
second index of sub array 2)
    while loop to compare first indexes of sub array 1 and sub array 2 till
    end of the each sub array's size by incriminating by one on each iteration
```

With divide and conquer algorithm merge sort, the time complexity is $nlog(n)$ because there are no for loops and there are only constant being added to the work because of the while loops and recursions. So in mergeSort, it is a recursive function so that repeats till right most index is less than left most index in the array while counting the reverses of two unsorted sub arrays and merge (which is dividing). This function merges two unsorted lists into one merged list while counting the flips in the array by using the merge function. In addition, it will copy each element of the original array to temporary array. After temporary array is sorted, it will copy back to the original array and return the total count.
Merge function will sorts the unsorted left and right sub arrays and count the total number of flips. In detail, the count will be keep adding from the number of flips in both left and right array, and merging two arrays.

c)

```
80  int main()
81  {
82     int size = 100;
83     int array[size];
84     srand(time(0));
85     for(int i = 0; i < size; i++){
86        array[i] = rand() % 200 + 1;
87     }
88      cout<<"Array is\n";
89     for(int j = 0; j < size; j++){
90        cout<<array[j]<<' ';
91     }
92     cout << endl;
93     cout << endl;
94     int number = mergeSort(array, size);
95     cout << "Number of reverse are " << number << '\n';
96     cout << endl;
97     for(int j = 0; j < size; j++){
98        cout<<array[j]<<' ';
99     }
100    return 0;
101 }
102
```

input

```
Array is
189 33 162 161 64 163 137 91 42 128 148 40 109 3 44 54 181 175 59 183 8 58 102 23 56 12
8 39 10 138 198 125 126 30 38 87 45 200 23 87 194 103 34 33 163 37 28 16 169 3 74 151 1
62 131 52 185 187 126 175 196 63 124 72 189 153 109 27 150 60 49 36 53 151 70 86 65 58
65 80 26 67 153 176 181 84 28 117 70 105 91 17 168 14 88 156 118 148 134 67 7 182

Number of reverse are 2455

3 3 7 8 10 14 16 17 23 23 26 27 28 28 30 33 33 34 36 37 38 39 40 42 44 45 49 52 53 54 5
5 58 58 59 60 63 64 65 65 67 67 70 70 72 74 80 84 86 87 87 88 91 91 102 103 105 109 109
117 118 123 124 125 126 126 128 131 134 137 138 148 148 150 151 151 153 153 156 161 16
2 162 163 163 168 169 175 175 176 181 181 182 183 185 187 189 189 194 196 198 200
```