**CSCI 174**

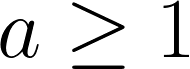
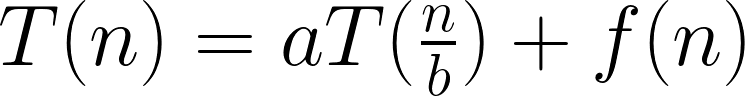
**Assignment 2**

**Master Method**

**Questions**

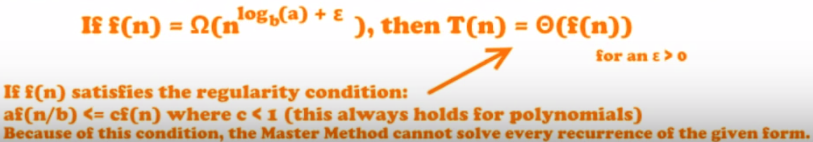
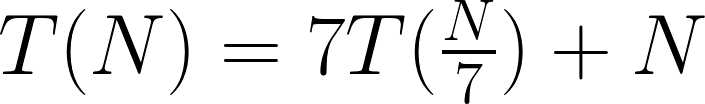
**Name: Abhishek Gupta**

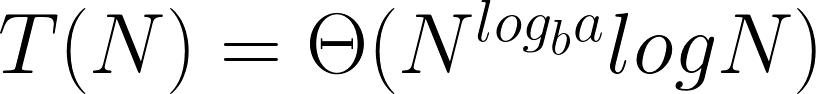
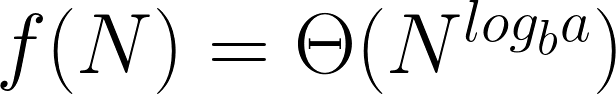
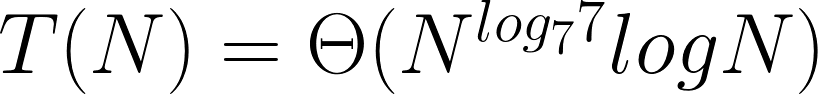
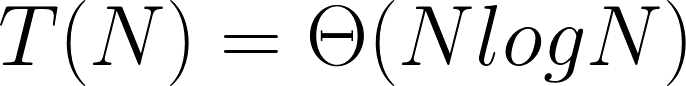
**ID Number:**

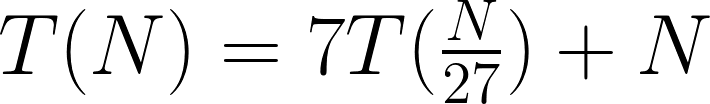
**C:\Users\joopj\AppData\Local\Microsoft\Windows\INetCache\Content.Word\CodeCogsEqn (2).pngMaster Method Rules:**

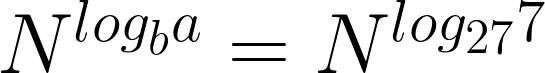
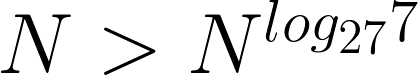
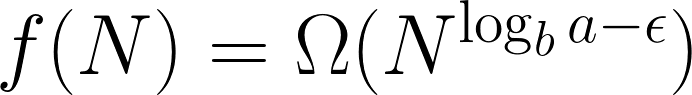
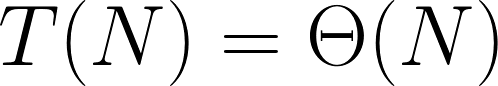
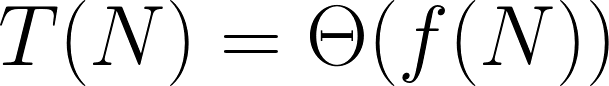
* n = size of current problem.
* a = number of subproblems in the recursion.
* n/b = size of each subproblem.
* f(n) = cost of the work that has to be done outside the recursive calls (cost of dividing + merging).

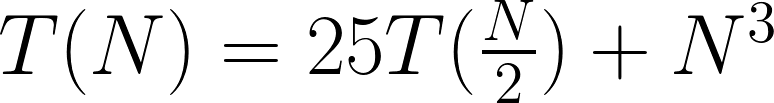
**3 Cases for f(n):**

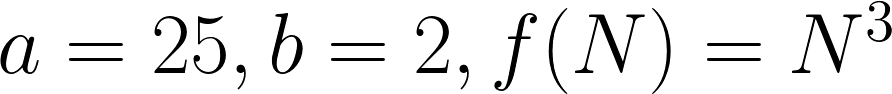
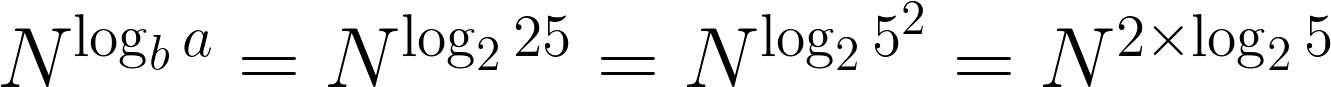
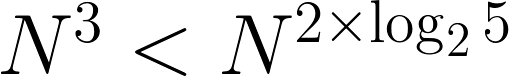
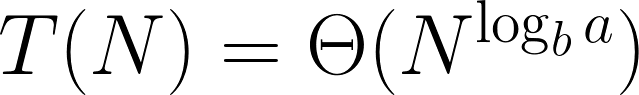
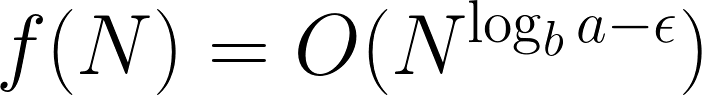
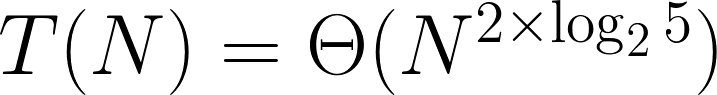
1. Runtime is dominated by cost at the leaves.  
   
2. Runtime is evenly distributed throughout tree.  
   C:\Users\joopj\AppData\Local\Microsoft\Windows\INetCache\Content.Word\2.png
3. Runtime is dominated by the cost at the root.   
   
4. Use the Master Method to solve the recurrences below.
5. 

STEP 1: Extract a, b, and f(N)  
 a = 7, b = 7, f(N) = N  
STEP 2: Compute   
  
   
  
STEP 3: Compare f(N) with   
  
 N = N, therefore we look at CASE 2  
STEP 4: If   
  
 then

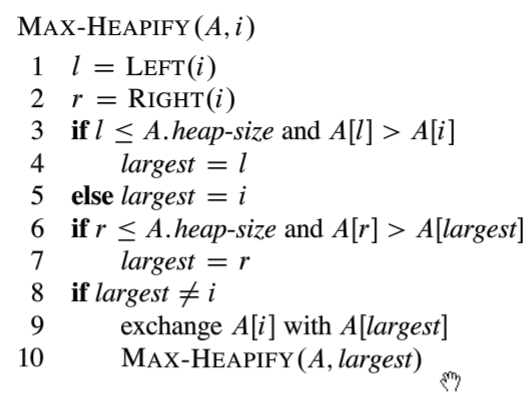
1. 

 STEP 1: Extract a, b, and f(N)  
 a = 7, b = 27, f(N) = N  
 STEP 2: Compute   
  
   
  
 STEP 3: Compare f(N) with   
  
   
therefore we look at CASE 3  
 STEP 4: If   
  
 then

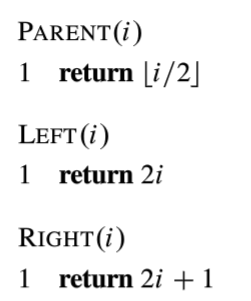
1. 

STEP 1: Extract a, b and f(N)  
   
  
STEP 2: Compute   
  
   
  
STEP 3: Compare f(N) with   
  
 therefore CASE 1   
STEP 4: If   
 then

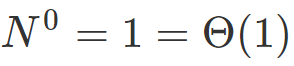
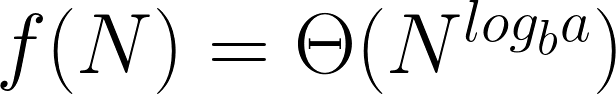
1. Consider the following pseudo-code:

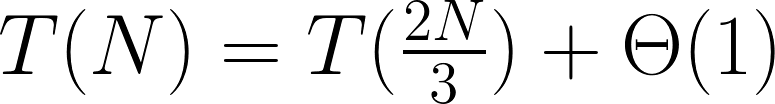
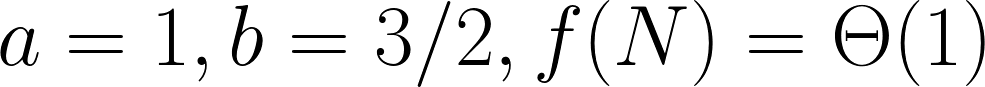
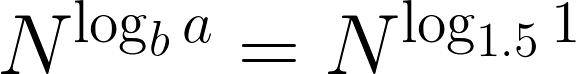


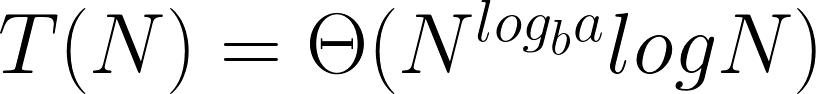
With the utility functions:



Model this algorithm with a recurrence relation, then solve it using master method.

C:\Users\joopj\AppData\Local\Microsoft\Windows\INetCache\Content.Word\112.png  
STEP 1: Extract a, b and f(N)  
  
  
STEP 2: Compute   
  
  
   
STEP 3: Compare f(N) with   
  
 Since   
, we say that   
therefore CASE 2.

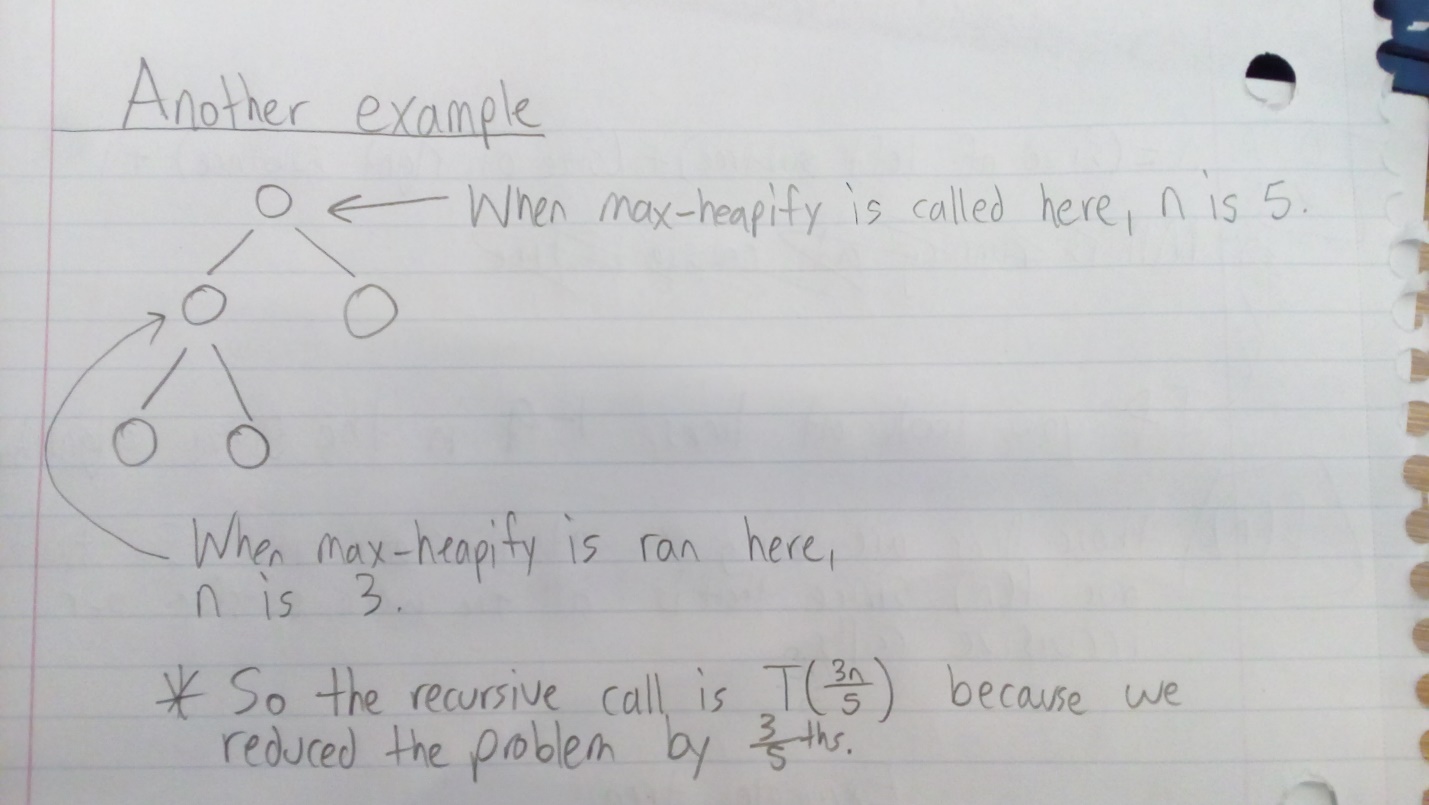


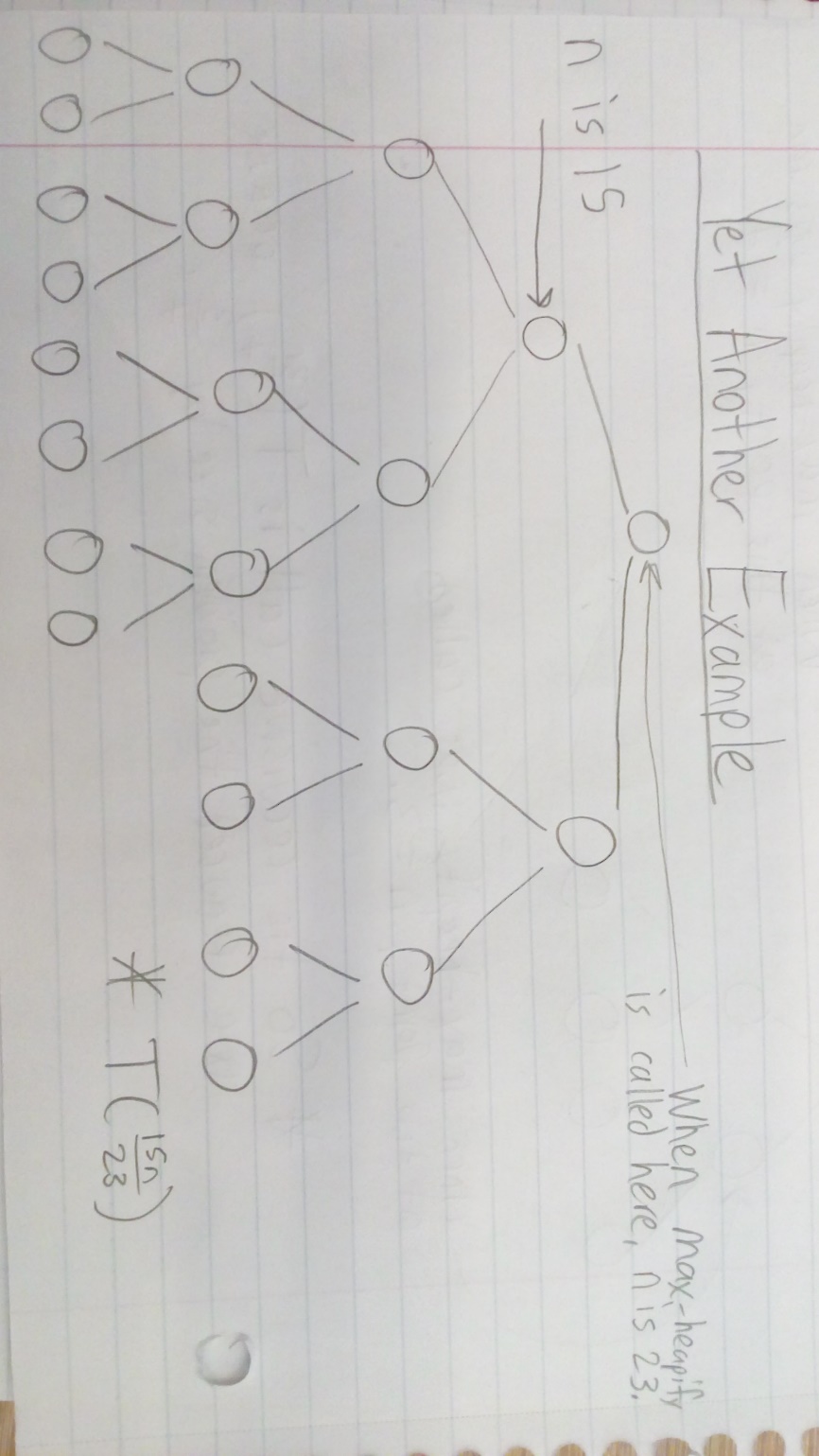
C:\Users\joopj\AppData\Local\Microsoft\Windows\INetCache\Content.Word\1.pngC:\Users\joopj\AppData\Local\Microsoft\Windows\INetCache\Content.Word\1.pngSTEP 4: If   
  
 Then

How do we get this recurrence? Let’s set this up. Let T(N) represent our runtime. Let N represent the number of elements in our tree. We know that a recurrence is in the form T(n) = aT(n/b) + f(n) where “a” represents the number of subproblems in the recursion, “n” represents the current size of the problem, “n/b” represents the size of each subproblem, and “f(n)” represents the cost of the work that had to be done outside the recursive calls.

Let’s start with figuring out what f(n). If we look at lines 1-9 in the algorithm, these lines are being ran at constant time or Theta(N). This is our f(n) because that is all the work outside our recursive calls.

Next, let’s look at “n/b.” Since this is representing each subproblem, it will represent the max number of nodes in a subtree. Let’s first look at a few examples:

In this example, we have a tree with 5 nodes. When max-heapify is called at the root, n is 5. When it is called at the left child node, n is 3. The recursive call is T(3n/5) because we reduced the problem by 3/5ths which is 0.60.



I this example, running max-heapify at root makes n be 23 and running it at the left child node makes it 15. The recurrence is T(15n/23) because we reduced the problem to 15/23rds which is about 0.65.

As you can see, as we increase n, we get closer to our bound of 0.66 which is 2/3rds. So our “n/b” is actually 2/3rds.

Therefore, our recurrence relation for max-heapify is T(N) = T(2n/3) + Theta(N).

Compute

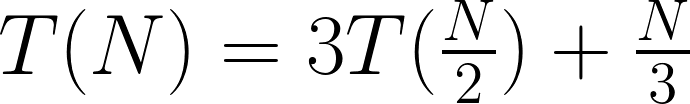
1. Consider an algorithm where we take in a problem and create 3 subproblems each half a large as the original problem. Then the subproblem solutions are combined with a single loop through a list 1/3 the size of the original problem.

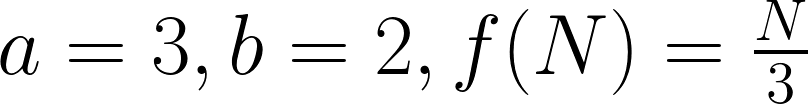
**Model this problem with a recurrence relation and solve using the master method.**

A recurrence relation is characterized as T(N) = aT(N/b) + f(N) where N represents the size of the problem, a represents the number of subproblems in the recursion, N/b is the size of each subproblem, and f(N) represents the cost of the work that has to be done outside the recursive calls.

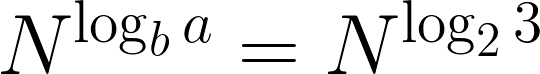
It is given that we have “3 subproblems each half as large as the original problem” which means that a = 3 and b = 2. Once that recursive call is executed we add the time to the work done outside the call which is a third of the size of the original problem with is f(N) = N / 3.

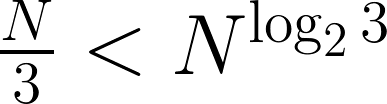
Therefore, our recurrence will be:

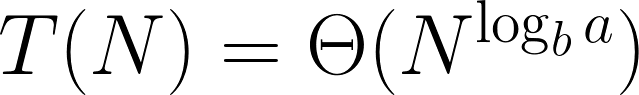
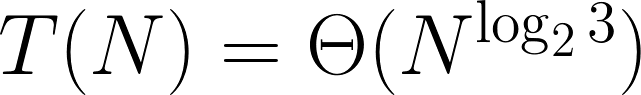
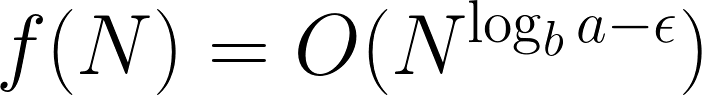


STEP 1: Extract a, b and f(N)

STEP 2: Compute



STEP 3: Compare f(N) with   
  
 therefore CASE 1

 STEP 4: If   
  
 Then

**Works Cited**

1. **VIDEO:** Master Method ( incl. Step-By-Step Guide and Examples ) – Analysis  
   **LINK:** <https://www.youtube.com/watch?v=6CX7s7JnXs0>
2. **VIDEO:** Algorithms lecture 12 -- Max heapify algorithm and complete binary tree  
   **LINK:** https://www.youtube.com/watch?v=2fA1FdxNqiE
3. **VIDEO:** Algorithms 6.2 – max heapify  
   **LINK:** <https://www.youtube.com/watch?v=tydfy_rLGmI>

<https://stackoverflow.com/questions/9099110/worst-case-in-max-heapify-how-do-you-get-2n-3>

<http://www.cems.uvm.edu/~rsnapp/teaching/cs124/notes/cs124notes_031914.pdf>

<https://www.quora.com/How-is-the-reccurence-relation-of-Max-Heapify-T-n-T-2n-3-+-theta-of-1>

<https://math.stackexchange.com/questions/181022/worst-case-analysis-of-max-heapify-procedure>