CSCI 174
Assignment 2
Master Method
Questions

Name: **Abhishek Gupta**

ID Number:

Master Method Rules:

$$T(n) = aT(\frac{n}{b}) + f(n) \quad a \geq 1 \quad b > 1$$

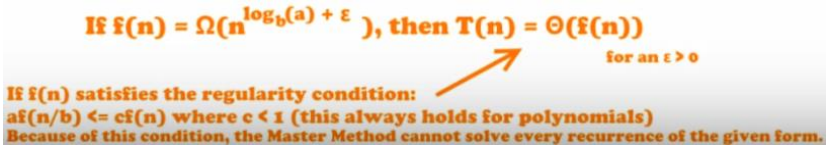
- n = size of current problem.
- a = number of subproblems in the recursion.
- n/b = size of each subproblem.
- f(n) = cost of the work that has to be done outside the recursive calls (cost of dividing + merging).

3 Cases for f(n):

1. Runtime is dominated by cost at the leaves.

2. Runtime is evenly distributed throughout tree.

3. Runtime is dominated by the cost at the root.



(1) Use the Master Method to solve the recurrences below.

(a) $T(N) = 7T(\frac{N}{7}) + N$

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

a = 7, b = 7, f(N) = N

STEP 2: Compute $N^{\log_b a}$

$$N^{\log_b a} = N^{\log_7 7} = N^1$$

STEP 3: Compare f(N) with $N^{\log_b a}$

N = N, therefore we look at CASE 2

STEP 4: If $f(N) = \Theta(N^{log_b a})$

then $T(N) = \Theta(N^{log_b a} logN)$

$$T(N) = \Theta(N^{log_7 7} logN)$$

$$\boxed{T(N) = \Theta(NlogN)}$$

(b) $T(N) = 7T(\frac{N}{27}) + N$

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

a = 7, b = 27, f(N) = N

STEP 2: Compute $N^{\log_b a}$

$N^{log_b a} = N^{log_{27} 7}$

STEP 3: Compare f(N) with $N^{\log_b a}$

$N > N^{log_{27} 7}$ therefore we look at CASE 3

STEP 4: If $f(N) = \Omega(N^{\log_b a - \epsilon})$

then $T(N) = \Theta(f(N))$

$$\boxed{T(N) = \Theta(N)}$$

(c) $T(N) = 25T(\frac{N}{2}) + N^3$

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

$a = 25, b = 2, f(N) = N^3$

STEP 2: Compute $N^{\log_b a}$

$N^{\log_b a} = N^{\log_2 25} = N^{\log_2 5^2} = N^{2 \times \log_2 5}$

STEP 3: Compare f(N) with $N^{\log_b a}$

$N^3 < N^{2 \times \log_2 5}$ therefore CASE 1

STEP 4: If $f(N) = O(N^{\log_b a - \epsilon})$

then $T(N) = \Theta(N^{\log_b a})$

$$\boxed{T(N) = \Theta(N^{2 \times \log_2 5})}$$

(2) Consider the following pseudo-code:

```
MAX-HEAPIFY(A, i)
 1  l = LEFT(i)
 2  r = RIGHT(i)
 3  if l ≤ A.heap-size and A[l] > A[i]
 4       largest = l
 5  else largest = i
 6  if r ≤ A.heap-size and A[r] > A[largest]
 7       largest = r
 8  if largest ≠ i
 9       exchange A[i] with A[largest]
10       MAX-HEAPIFY(A, largest)
```

With the utility functions:

```
PARENT(i)
1  return ⌊i/2⌋

LEFT(i)
1  return 2i

RIGHT(i)
1  return 2i + 1
```

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

$$T(N) = T(\frac{2N}{3}) + \Theta(1)$$

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

$$a = 1, b = 3/2, f(N) = \Theta(1)$$

STEP 2: Compute $N^{\log_b a}$

$$N^{\log_b a} = N^{\log_{1.5} 1}$$

STEP 3: Compare f(N) with $N^{\log_b a}$

Since $\log_{1.5} 1 = 0$, we say that $N^0 = 1 = \Theta(1)$ therefore CASE 2.

STEP 4: If $f(N) = \Theta(N^{log_b a})$

Then $T(N) = \Theta(N^{log_b a} logN)$

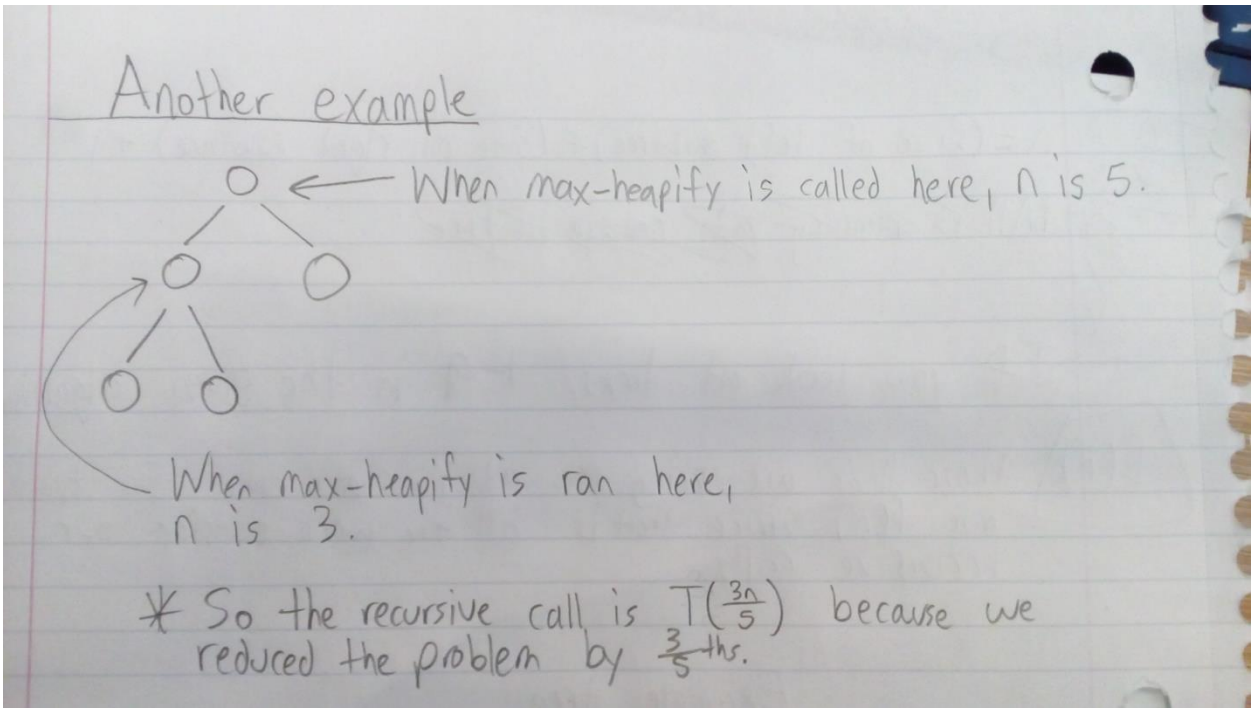
$$T(N) = \Theta(N^0 \log n)$$

$$\boxed{T(N) = \Theta(\log n)}$$

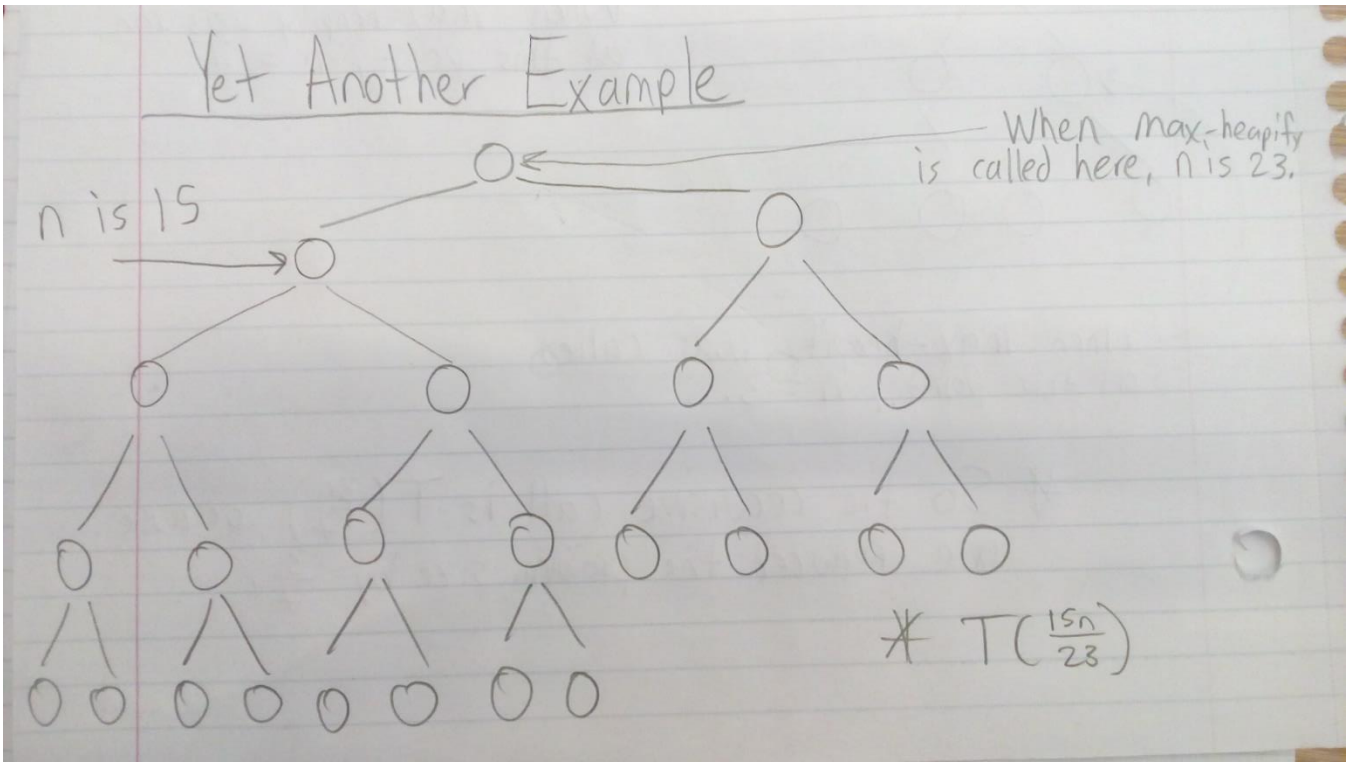
How do we get this recurrence? Let's set this up. Let T(N) represent our runtime. Let N represent the number of elements in our tree. We know that a recurrence is in the form T(n) = aT(n/b) + f(n) where "a" represents the number of subproblems in the recursion, "n" represents the current size of the 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

(3) 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:

$$T(N) = 3T(\frac{N}{2}) + \frac{N}{3}$$

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

$$a = 3, b = 2, f(N) = \frac{N}{3}$$

STEP 2: Compute $N^{\log_b a}$

$$N^{\log_b a} = N^{\log_2 3}$$

STEP 3: Compare f(N) with $N^{\log_b a}$

$\frac{N}{3} < N^{\log_2 3}$ therefore CASE 1

STEP 4: If $f(N) = O(N^{\log_b a - \epsilon})$

Then $T(N) = \Theta(N^{\log_b a})$

$$\boxed{T(N) = \Theta(N^{\log_2 3})}$$

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