CS: 3330 EXAM 1 SOLUTIONS

1(a) (i) $2^{3\log_2 n} = \left(2^{\log_2 n}\right)^3 = n^3$.

(ii) $T(n) = \Theta(n^2)$. (This was discussed in class, motivated by the question of the run-time of a modified version of mergeSort in which one subarray has size 1 and the other has size $n-1$.)

(iii) Running time is $\Omega(2^{n/2})$. This is because a n-bit integer can be as large as 2^n and the naive primality algorithm evaluates numbers in the range 2 through $\sqrt{2^n} = (2^n)^{1/2} = 2^{n/2}$ to check if they're factors.

(iv) $\sum_{i=1}^{n} \Theta(\frac{1}{2}i) = \Theta(1)$. So we see that

$$(\log_2 n)^2 \sum_{i=1}^{n} \Theta(\tfrac{1}{2}i) = \Theta(\log^2 n).$$

(v) $n^{2.5}/(\log_2 n)^4 = \Theta(n^{2.5}/\log^2 n)$.

(vi) $T(n) = \Theta(\log_4 n) = \Theta(\log n)$. This can be seen by using the Master Theorem ($a=1$, $b=4$, $d=0$) or by unrolling the recurrence.

Ordering:

(vi)	(iv)	(ii)	(v)	(i)	(iii)
$\Theta(\log n)$	$\Theta(\log^2 n)$	$\Theta(n^2)$	$\Theta\left(\frac{n^{2.5}}{\log^2 n}\right)$	$\Theta(n^3)$	$\Omega(2^{n/2})$

1(b) (i) FALSE.

$100n^3 + 10n^2 + 15 = O(n^3 \log_2 n)$, but

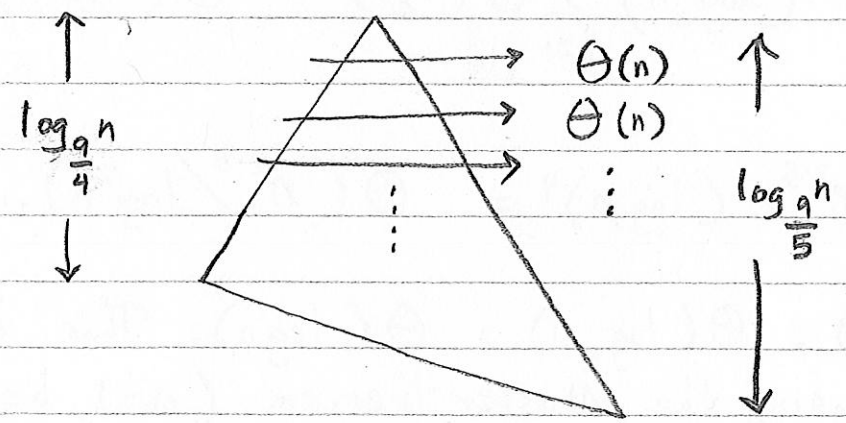
$100n^3 + 10n^2 + 15 \neq \Omega(n^3 \log_2 n)$.

(ii) FALSE.

$\Theta(\sqrt{2^n}) = \Theta(2^{n/2})$ is an exponential function in n, whereas $\Theta(4^{\log_2 n}) = \Theta(n^{\log_2 4}) = \Theta(n^2)$ is a polynomial function and we prefer algorithms with polynomial running time to algorithms with exponential running time.

(iii) TRUE.

As discussed in class, this can be easily seen using the visual method.



$\therefore$ Running time is $O(n \log n)$ and $\Omega(n \log n)$ and $\therefore \Theta(n \log n)$. Since the base of the logarithm does not matter inside asymptotic notation, $\Theta(n \log n) = \Theta(n \log_2 n)$.

(iv) FALSE.

Binary Search recurrence is $T(n) = T(n/2) + O(1)$, not $T(n) = 2T(n/2) + O(1)$. This is because in Binary search we solve only one of the two subproblem

2. (a) $\Theta(n \log_3 n) = \Theta(n \log n)$.

Outer loop executes $\Theta(n)$ times and inner loop executes $\Theta(\log_3 n)$ times. Furthermore, every statement in the code fragment executes in $\Theta(1)$ time.

(b) $\Theta\left(\frac{n^{3/2}}{k}\right)$.

Outer loop executes $\Theta(n)$ times and inner loop executes $\Theta\left(\frac{n^{1/2}}{k}\right)$ times. That the inner loop executes $\Theta\left(\frac{n^{1/2}}{k}\right)$ times, can be seen as follows. Suppose the inner loop executes t times. Then $1 + 2k \cdot t \sim \sqrt{n}$ $\Rightarrow t \simeq \frac{\sqrt{n}}{2k}$. Furthermore, every statement in the code fragment executes in $O(1)$ time.

(c) $\Theta(\log n)$.

This is just binary search except that the two subarrays produced by the Divide step have sizes $\frac{3n}{4}$ and $\frac{n}{4}$. Assuming the worst case, i.e.,

the algorithm repeatedly searches the subarray of size $\frac{3n}{4}$, we get the recurrence

$$T(n) = T(3n/4) + \Theta(1).$$

Using the Master Theorem ($a=1$, $b=4/3$, $d=0$) we see that $T(n) = \Theta(\log_{4/3} n) = \Theta(\log n)$.

<u>Note</u>: The code has a "bug" in it and the assignment to mid in Line 4 should be

$$\text{mid} \leftarrow \text{left} + \frac{(\text{right} - \text{left})}{4}.$$

Some students noticed the "bug".

3. (a) $\Theta(\log n) + \Theta(\log p) = \Theta(\log n) + \Theta(1) = \Theta(\log n)$.

(b) $\Theta(\log(n!)) = \Theta(n \log n)$.

(c) $\Theta(m^3 \log^3 n)$.

To see this note that m can be represented using $O(n \log n)$ bits and i using $O(\log n)$ bits. So using the grade-school multiplication algorithm, we can compute $m * i$ in $O(n \log^2 n)$ time. The assignment $m \leftarrow m * i$ takes an additional $O(n \log n)$ time. Since all this is repeated n times, the for-loop takes $O(n^2 \log^2 n)$ time. By just considering $i \leftarrow n/2$ to n, we can show that $m \leftarrow m * i$ takes $\Omega(n \log n)$ time for each of these values of i. So $\Omega(n^2 \log n)$ is an understimate

of the running time of the loop. So we see that the for-loop takes $\Theta(n^2 \log^2 n)$ time.

Then we consider the time it takes to evaluate m <u>mod</u> p. In class, we discussed an algorithm that took $\Theta(x^3)$ time to compute a mod b where a is represented using x bits and $b \leq a$. Since m is represented using $\Theta(n \log n)$ bits, it takes $\Theta(n^3 \log^3 n)$ time to compute m mod p.

Let $s = \Theta(\log n)$. Then $\Theta(n^3 \log^3 n) = \Theta(2^{3\log_2 n} \cdot \log^3 n$

$= \Theta(2^{\Theta(s)} \cdot s^3)$. This is exponential in s.

(d) No.

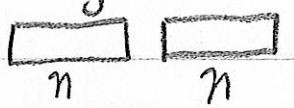
(e) Replace $m \leftarrow m * i$ by $m \leftarrow (m * i) \bmod p$.

(f) No.
Unfortunately, the for-loop still executes n times and so the running time is $\Omega(n)$, which is exponential is s, the input size.

4 (a) $\Theta(nk^2)$.

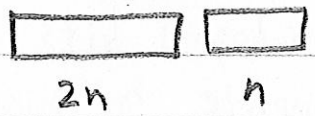
Reasoning

Merge 1 — Running Time



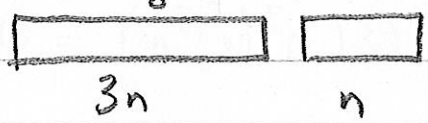
$\Theta(2n)$

Merge 2



$\Theta(3n)$

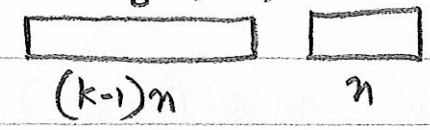
Merge 3



$\Theta(4n)$

$\vdots$ $\qquad$ $\vdots$

Merge $(k-1)$



$\Theta(kn)$

$$\text{Total time} = \sum_{i=2}^{k} \Theta(in) = \Theta(n) \cdot \sum_{i=2}^{k} i = \Theta(nk^2).$$

(b) $T(nk) = 2T\left(\frac{nk}{2}\right) + \Theta(nk)$

(c) Let us use x to denote nk. So our recurrence is

$$T(x) = 2T(x/2) + \Theta(x)$$

Unrolling it once:

$$T(x) = 2\left(2T(x/4) + \Theta(x/2)\right) + \Theta(x)$$

$$= 2^2 T\left(x/2^2\right) + 2^1 \Theta\left(x/2^1\right) + 2^0 \Theta\left(x/2^0\right)$$

$$= 2^2 T\left(x/2^2\right) + 2 \cdot \Theta(x)$$

Unrolling it a second time:

$$T(x) = 2^2\left(2T\left(x/2^3\right) + \Theta\left(x/2^2\right)\right) + 2 \cdot \Theta(x)$$

$$= 2^3 T\left(x/2^3\right) + 2^2 \cdot \Theta\left(x/2^2\right) + 2 \cdot \Theta(x)$$

$$= 2^3 T\left(x/2^3\right) + 3 \cdot \Theta(x).$$

Guess:

$$T(x) = 2^t \cdot T\left(x/2^t\right) + t \cdot \Theta(x)$$

We want $\frac{x}{2^t} = 2n$, because $T(2n) = \Theta(n)$ is a base case. So we solve $\frac{x}{2^t} = 2n$.

$$\frac{nk}{2^t} = 2n \Rightarrow k = 2^{t+1} \Rightarrow t+1 = \log_2 k.$$

$$\Rightarrow t = \log_2 k - 1$$

So

$$T(nk) = \frac{k}{2} \cdot T(2n) + \left(\log_2 k - 1\right) \Theta(nk)$$

$$= \frac{k}{2} \cdot \Theta(n) + \Theta(nk \log k) = \Theta(nk) + \Theta(nk \log k)$$

$$= \Theta(nk \log k).$$

5. (a) $\Theta(n)$

(b) Consider an array with elements 0's and 1's. Assume that 0 is a majority element, but for each of the k "trials" p ends up being the index of a 1. Then the algorithm returns "no majority element" even though the input contains a majority.

(c) $< \left(\frac{1}{2}\right)^k = \frac{1}{2^k} = \frac{1}{8}$.

(d) With $k=3$, the prob. of error $< \frac{1}{8}$. If we pick $k=7$, we get that the prob. of error is $< \frac{1}{2^7} = \frac{1}{128} < 1\%$. Note that this is the smallest value of k that yields a less than 1% error probability.