Problem Set 1

All parts are due Tuesday, September 13 at 11PM.

Name: Kevin Jiang

Collaborators: Jason Lu, Suraj Srinivasan, Stephanie Zhang, Hieu Nguyen

Problem 1-1.

(a)

$$\begin{aligned}
f_1 &= 20n + 18 = \Theta(n) \\
f_2 &= 20n \cdot 18 = \Theta(n) \\
f_3 &= 20n^{18} = \Theta(n^{18}) \\
f_4 &= \log_{20}(n^{18}) = 18\log_{20}(n) = \Theta(\log(n)) \\
f_5 &= (\log_{18}(n))^{20} = \Theta(\log(n)^{20})
\end{aligned}$$

$$\Rightarrow \boxed{(f_4, f_5, \{f_1, f_2\}, f_3)}$$

(b)

$$\begin{aligned}
f_1 &= n^{2\log n} = \Theta(n^{\log n^2}) \\
f_2 &= 2^{2^{\log n}} = 2^n = \Theta(2^n) \\
f_3 &= 2^{(\log n)^2} = (2^{\log n})^{\log n} = n^{\log n} = \Theta(n^{\log n}) \\
f_4 &= \Theta(n^{\log n}) \\
f_5 &= \Theta(2^{\log(\log n)}) = \Theta(\log n)
\end{aligned}$$

$$\Rightarrow \boxed{(f_5, \{f_3, f_4\}, f_1, f_2)}$$

(c)

$$\begin{aligned}
f_1 &= \Theta(2^{n^3}) \\
f_2 &= \Theta(2^{(n+1)^3}) \\
f_3 &= \Theta(n^{n^2}) \\
f_4 &= \Theta(4^{2^n}) \\
f_5 &= \Theta(3^{2^n})
\end{aligned}$$

$$\Rightarrow \boxed{(f_3, f_1, f_2, f_5, f_4)}$$

(d)

$$
\begin{aligned}
f_1 &= (2n)! \approx \sqrt{4\pi n}\left(\frac{2n}{e}\right)^{2n} = \Theta\left(\left(\frac{2}{e}\right)^{2n} \cdot \sqrt{n} \cdot n^{2n}\right) \\
f_2 &= \frac{(2n)!}{n! \cdot n!} \approx \frac{\sqrt{4\pi n}\left(\frac{2n}{e}\right)^{2n}}{2\pi n \cdot \left(\frac{n}{e}\right)^{2n}} = \Theta\left(\frac{1}{\sqrt{n}} \cdot 4^n\right) \\
f_3 &= \Theta(4^n) \\
f_4 &= \Theta(2^n \cdot n^n) \\
f_5 &= \Theta((n^n)^2)
\end{aligned}
$$

$$\Rightarrow \boxed{(f_2, f_3, f_4, f_1, f_5)}$$

Problem 1-2.

(a) i. Let $T(n)$ be the run time for finding the maximum sub-array sum of an array of length n. The first two cases of Derek's strategy each require $T\left(\frac{2n}{3}\right)$ work since they are analyzing a set of $\frac{2n}{3}$ elements. The amount of work needed for the third case of Derek's strategy is $O(n)$ since we can scan forwards to find the largest sub-array beginning at the $\frac{2}{3}$rd mark, and scan backward to find the largest sub-array ending at the $\frac{2}{3}$rd mark. Hence, the recurrence for the amount of work Dereks strategy will take is $\boxed{T(n) = 2T\left(\frac{2n}{3}\right) + O(n)}$.

ii. We have the general case of the Master Theorem:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n).$$

For the recurrence in the first part, we have $a = 2$, $b = \frac{3}{2}$, and $f(n) = O(n)$. Since $\log_b a = \log_{\frac{3}{2}} 2 \approx 1.7$, we are in the first case of the Master Theorem ($f(n) = O(n^{\log_b a - \epsilon}) = O(n^{1.7})$. Therefore, we have $\boxed{T(n) = \Theta(n^{\log_{\frac{3}{2}} 2})}$, which is not $\Theta(n \log n)$.

(b) i. $T(n) = 2T(\frac{n}{4}) + O(\log n)$

$$\Rightarrow a = 2,\ b = 4,\ f(n) = O(\log n) \Rightarrow \log_b a = \log_4 2 = \frac{1}{2}.$$

Since $f(n) = O(\sqrt{n})$, the leaves dominate the run-time. Therefore, $\boxed{T(n) = \Theta(\sqrt{n})}$.

ii. $T(n) = 17T(\frac{n}{3}) + \Theta(n!)$

$$\Rightarrow a = 3,\ b = 17,\ f(n) = \Theta(n!) \Rightarrow \log_b a = \log_3 17 \approx 2.57.$$

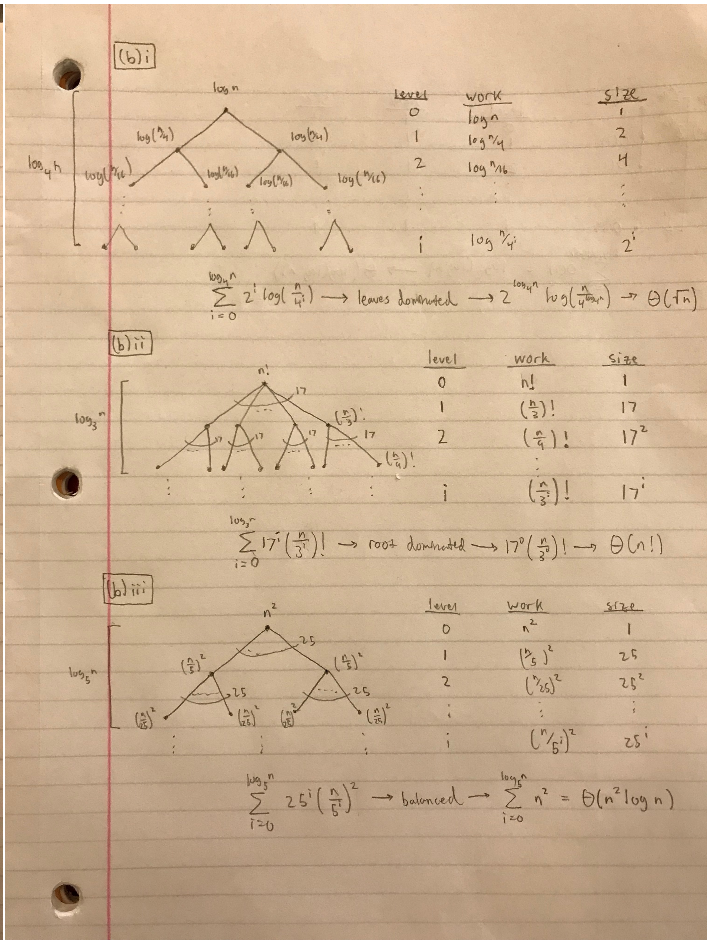
Since $f(n) = \Omega(n^{2.57})$, the root dominates the run-time. Therefore, $\boxed{T(n) = \Theta(n!)}$.

iii. $T(n) = 25T(\frac{n}{5}) + n^2 + 2n + \log n$

$$\Rightarrow a = 25,\ b = 5,\ f(n) = O(n^2) \Rightarrow \log_b a = \log_5 25 = 2.$$

Since $f(n) = \Theta(n^2)$, we are in the balanced case of the Master Theorem. Therefore, $\boxed{T(n) = \Theta(n^2 \log n)}$.

iv. $T(n) = T(\sqrt{n}) + O(1)$



$\boxed{(b)i}$

$\log n$
$\log(n/4)$
$\log(n/4)$
$\log(n/16)$
$\log(n/16)$
$\log(n/16)$
$\log(n/16)$
$\log_4 n$

Level	work	size
0	$\log n$	1
1	$\log n/4$	2
2	$\log n/16$	4
$\vdots$	$\vdots$	$\vdots$
i	$\log n/4^i$	2^i

$$\sum_{i=0}^{\log_4 n} 2^i \log\left(\frac{n}{4^i}\right) \longrightarrow \text{leaves dominated} \longrightarrow 2^{\log_4 n} \log\left(\frac{n}{4^{\log_4 n}}\right) \rightarrow \Theta(\sqrt{n})$$

$\boxed{(b)ii}$

$n!$
17
$(\frac{n}{3})!$
17
17
17
17
$(\frac{n}{9})!$
$\log_3 n$

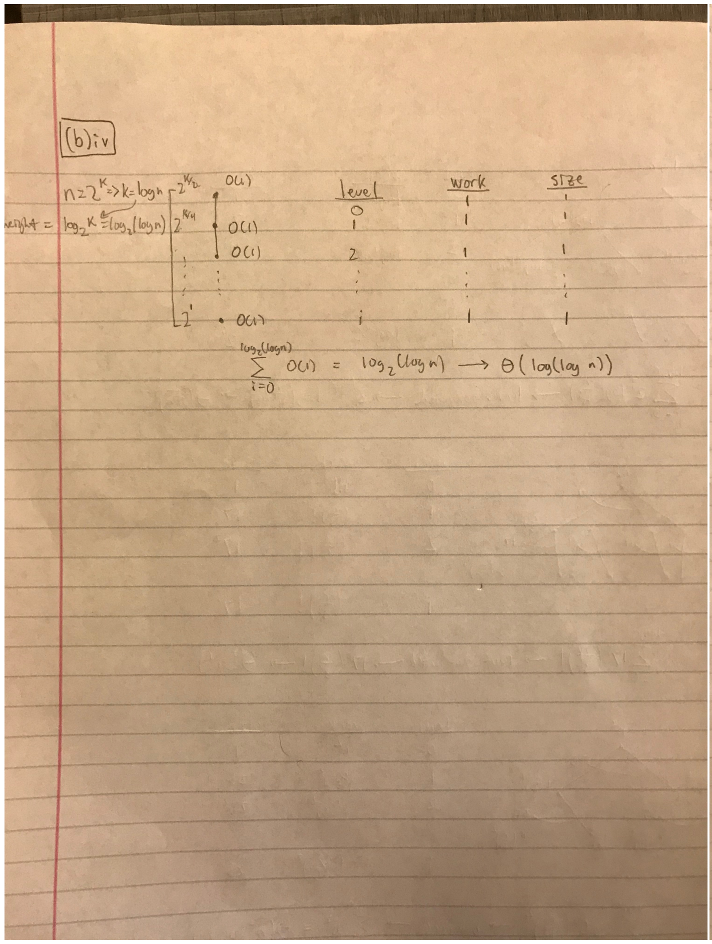
level	work	size
0	$n!$	1
1	$(\frac{n}{3})!$	17
2	$(\frac{n}{9})!$	17^2
	$\vdots$	
i	$(\frac{n}{3^i})!$	17^i

$$\sum_{i=0}^{\log_3 n} 17^i \left(\frac{n}{3^i}\right)! \rightarrow \text{root dominated} \longrightarrow 17^0\left(\frac{n}{3^0}\right)! \longrightarrow \Theta(n!)$$

$\boxed{(b)iii}$

level	work	size
0	n^2	1
1	$(n/5)^2$	25
2	$(n/25)^2$	25^2
$\vdots$	$\vdots$	$\vdots$
i	$(n/5^i)^2$	25^i

$$\sum_{i=0}^{\log_5 n} 25^i \left(\frac{n}{5^i}\right)^2 \longrightarrow \text{balanced} \longrightarrow \sum_{i=0}^{\log_5 n} n^2 = \Theta(n^2 \log n)$$



$\boxed{(b)iv}$

$n = 2^k \Rightarrow k = \log n$

height $= \log_2 k = \log_2(\log n)$

$2^{k/2}$
$2^{k/4}$
2^1
$O(1)$
$O(1)$
$O(1)$
$O(1)$

level	work	size
	1	1
0	1	1
1		
2	1	1
$\vdots$	$\vdots$	$\vdots$
i	1	1

$$\sum_{i=0}^{\log_2(\log n)} O(1) = \log_2(\log n) \longrightarrow \Theta(\log(\log n))$$

Problem 1-3.

(a) For each of the n rows, we apply binary search to see if v is present in the row. Since there are m elements in each row, the running time for each of the n rows will be $\log m$. Hence, in the worst case scenario where v is not present in A , the algorithm is $O(n \log m)$

(b) By the definition of a two-dimensionally sorted matrix, the bottom left element will always be the largest and smallest element in the 0th column and $(n - 1)$th row, respectively. Therefore, if $v > s$, then v is larger than any element in the 1st column, reducing the search area to an $n \times (m - 1)$ matrix. On the other hand, if $v < s$, then v is smaller than any element in the nth row, so the search area becomes an $(n - 1) \times m$ matrix. In either case, we are eliminating one row containing m elements or one column containing n elements. Hence, we reduce the number of elements from A by at least $\min(n, m)$.

(c) We claim that the following greedy algorithm will find the integer v in a two-dimensionally sorted matrix A in at most $O(n + m)$ time:

We always look at the bottom left corner of the matrix. As described in part (b), we can delete either one row or one column in the matrix. Next, we compare the "new" bottom left corner with v and delete another row or column. We can continue this process of comparing the bottom left corner to v and eliminating either one row or one column from the matrix until we find v or operate through the entire matrix.

At every step of our algorithm, we are deleting either a row or a column from the matrix. This is equivalent to subtracting one from either n or m in a $n \times m$ matrix. Since every element in the matrix can be accessed by subtracting one a non-negative number of times from both n and m, every element has a chance to be the "bottom left corner". Hence, this algorithm is correct.

The worst case scenario is when v is not in the matrix. In this case, we have to delete all the rows and also delete all the columns. Since there are n rows and m columns in total, the maximum work needed is $n + m$, or $O(n + m)$.

(d) Submit your implementation to `alg.mit.edu`.