# Problem Set 1

**Problem 1-1.** [0 points] **Instructions**

**All parts are due on February 25, 2020 at 10PM**. Aim for concise solutions; convoluted and obtuse descriptions might receive low marks, even when they are correct. Solutions should be submitted on Gradescope, and any code should be submitted for automated checking on `https://alg.mit.edu`.

Please abide by the collaboration policy as stated in the course information handout. Note that the first problem is an *Individual* problem and collaboration on this problem is not allowed.

**(1) Asymptotic behavior**

For each of the following sets of five or six functions, order them so that if $f_a$ appears before $f_b$ in your sequence, then $f_a = O(f_b)$. If $f_a = O(f_b)$ and $f_b = O(f_a)$ (meaning $f_a$ and $f_b$ could appear in either order), indicate this by enclosing $f_a$ and $f_b$ in a set with curly braces. For example, if the functions are:

$$f_1 = n, \qquad f_2 = \sqrt{n}, \qquad f_3 = n + \sqrt{n},$$

then the correct answers are $(f_2, \{f_1, f_3\})$ and $(f_2, \{f_3, f_1\})$.

Note: Recall that $a^{b^c}$ means $a^{(b^c)}$, not $(a^b)^c$, and that $\log$ means $\log_2$ unless a different base is specified explicitly. Stirling's approximation may help for part (d).

a ) [5 points]
$f_1 = (\log n)^{6006}$
$f_2 = 6006^n$
$f_3 = \log(6006n)$
$f_4 = 6006n^{1.001}$
$f_5 = n \log(n^{6006})$
**Solution:** $(f_3, f_1, f_5, f_4, f_2)$. This order follows from knowing that any power of $\log(n)$ grows slower than $n^a$ for all $a > 0$ and that $n^a$ grows slower than $n^b$ for $0 < a < b$.

b ) [5 points]
$f_1 = 2^{n^3}$
$f_2 = 3^{2^n}$
$f_3 = 2^{2^{n+1}}$
$f_4 = 2^{2n}$
$f_5 = 4^n$
**Solution:** $(\{f_4, f_5\}, f_1, f_2, f_3)$. This order follows from converting all of the exponent bases to 2 as shown below and noting that multiplicative constants in the exponents cannot be neglected.
$f_1 = 2^{n^3}$
$f_2 = 2^{\log(3) \cdot 2^n}$
$f_3 = 2^{2 \cdot 2^n}$
$f_4 = 2^{2n}$
$f_5 = 2^{2n}$

c ) [5 points]
$f_1 = \log\left((\log n)^3\right)$
$f_2 = n^{3 \log n}$
$f_3 = (\log \log n)^3$
$f_4 = (\log n)^{\log(n^3)}$
$f_5 = \log\left(3^{n^3}\right)$
$f_6 = \log\left(\log(n^3)\right)$

**Solution:** $(\{f_6, f_1\}, f_3, f_5, f_4, f_2)$. This order follows from elementary exponentiation and logarithm rules and converting some of the exponent bases to 2 as shown below.
$f_1 = 3 \log \log n$
$f_2 = 2^{3(\log n)^2}$
$f_3 = (\log \log n)^3$
$f_4 = 2^{3 \cdot (\log n) \cdot (\log \log n)}$
$f_5 = \log(3) \cdot n^3 = \log(3) \cdot 2^{3 \log n}$
$f_6 = \log(3 \log n) = \log 3 + \log \log n$

d ) [5 points]
Hint: Think about Stirling's approximation for $f_5$.
$f_1 = 2^n$
$f_2 = 2(n!)$
$f_3 = n^2$
$f_4 = \binom{n}{2}$
$f_5 = \binom{n}{n/2}$
$f_6 = \binom{n}{3}$
**Solution:** $(\{f_3, f_4\}, f_6, f_5, f_1, f_2)$. This order follows from the definition of the binomial coefficient and Stirling's approximation as shown below.
$f_1 = 2^n$
$f_2 = 2(n!) = 2 \cdot 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n \geq 2 \cdot 1 \cdot 2 \cdot 2 \cdot \ldots \cdot 2 \cdot n = n2^{n-1}$
$f_3 = n^2$
$f_4 = \binom{n}{2} = n(n-1)/2 = \Theta(n^2)$
$f_5 = \binom{n}{n/2} = \Theta\left(\frac{\sqrt{n}(n/e)^n}{\left(\sqrt{n}(n/(2e))^{n/2}\right)^2}\right) = \Theta\left(\frac{\sqrt{n}(n/e)^n}{n(n/(2e))^n}\right) = \Theta(2^n/\sqrt{n})$
$f_6 = \binom{n}{3} = \Theta(n^3)$

**Rubric:**

- 5 points per set for a correct order
- $-1$ point per inversion
- $-1$ point per grouping mistake, e.g., $(\{f_1, f_2, f_3\})$ instead of $(f_2, f_1, f_3)$ is $-2$ points because they differ by two splits.
- 0 points minimum

**(2) Recurrences**

Solve each of the following recurrences for $T(n)$. (Use $\Theta$ notation to give the asymptotic runtime.) Please justify your answers.

a ) [5 points] $T(n) = 9T(n/3) + n^2$
**Solution:** We see that $n^{\log_3 9} = n^2$, so we are in case 2 of the Master Theorem and the runtime is $\Theta(n^2 \log n)$.

b ) [5 points] $T(n) = 6006T(n/1.001) + 2^n$
**Solution:**

We see that $n^{\log_{1.001} 6006}$ grows polynomially while $f(n) = 2^n$ grows exponentially, so we are in case 3 of the Master Theorem and the runtime is $\Theta(2^n)$. We also have to check the regularity condition, which is that $af(n/b) \leq kf(n)$ for some constant $k < 1$ and sufficiently large $n$, and we see that $af(n/b) = 6006 \cdot (2^{1/1.001})^n$, which is smaller than $k \cdot 2^n$ for any $k > 0$ and for sufficiently large $n$, as $2^{1/1.001}$ is a smaller base than 2.

c ) [5 points] $T(n) = 2T(3n/16) + n$
**Solution:** Use case 3 of the Master Theorem to get that total work is $\Theta(n)$. We have that $n^{\log_{16/3} 2}$ grows slower than $n$ and can do the rest of the case analysis to see that case 3 applies.
Alternative solution: At each row of the work tree, we have $2 \cdot \frac{3}{16} = \frac{3}{8}$ of the work we had at the previous level. The argument in each node is at most $\frac{3}{16}$ of the argument of its parent, so there are at most $\log_{16/3} n$ levels. Thus, the total work we have is $\sum_{i=0}^{\log_{16/3} n} (\frac{3}{8})^i n \leq \sum_{i=0}^{\infty} (\frac{3}{8})^i n$ where the last inequality comes from the fact that all terms are non-negative. Using the formula for a geometric series, this is at most $n/(1-3/8) = O(n)$. We also know that we do $\Omega(n)$ work because we must do $n$ work at the first level, so we do $\Theta(n)$ work total.

d ) [5 points] $T(n) = T(n/7) + T(11n/14) + \Theta(n)$
Hint: One way to think about this problem is by considering a work tree. First, show that it takes $T(n) = O(n)$ and then that $T(n) = \Omega(n)$.
**Solution:** We analyze this as a work tree. We know that at the next level, we have at most $1/7 + 11/14 = 13/14$ of the work we had at the previous level. (You can think about doing 13/14 of the work done in each node at the previous level, which can be grouped out of the sum of the work across all nodes at that level.) This means that our total work is at most $\sum_{i=0}^{L} (13/14)^i n \leq \sum_{i=0}^{\infty} (13/14)^i n$ where $L$ is the number of levels in the tree. (The last inequality is due to the fact that all terms are non-negative.) This is at most $n/(1 - 13/14) = 14n = O(n)$ work overall. We also do $\Theta(n)$ work in the first level, meaning we do $\Omega(n)$ work overall and thus our total work is $\Theta(n)$.

**Rubric:**

- 2 points per correct runtime bound
- 3 point per correct justification

**Problem 1-2.** [30 points] **Agent B travels through time.**

Stepbrothers Fineas and Pherb recently fixed up an old time machine they found in the Vandille museum, but before they get a chance to use it, it's stolen by their pet beaver Berry (aka Agent B). Fortunately, he takes it for a good purpose though - stopping the evil Dr. Deinz Hoofenshmirtz.

Agent B knows that, at some point during his life, Dr. Hoofenshmirtz will create a fancy new machine, the Rule-inator, that will allow him to rule the entire tri-state area! He wants to stop Dr. Hoofenshmirtz from using the machine, so he plans to travel through time to destroy it once it is built but before he uses it. Unfortunately, he doesn't know when Dr. Hoofenshmirtz will invent the machine so he's not sure what day he should travel to. However, he does know that the machine takes at least one day to charge, so his goal is to find the machine the day *before* it is used. This is his "target date."

You can assume that whenever Agent B arrives at a new day, he can tell if the machine has been used yet or not. Also assume that the machine will not be used today and that Agent B only needs to destroy the machine once.

**(1)** [10 points] Agent B assumes that Dr. Hoofenshmirtz has a finite lifespan of at most $n$ more days. Describe and analyze an algorithm that allows Agent B to save the world by using the time machine at most $O(\log n)$ times. Be sure to analyze the correctness and runtime of your solution.

**Solution:** For this problem, we will do a binary search on the $n$ possible days on which the event occurs. In order to do binary search, we have to be able to determine if a given day, $i$, is the target day or if the target occurs before $i$ or after $i$. If the machine has been used by day $i$, then we know the target date occurs before $i$. If the machine has not been used, we also check day $i + 1$ and if the machine has not been used on either of these days, we know the target date occurs after day $i$. Otherwise, day $i$ is the target.

**Correctness:** In this case, we are using binary search (an algorithm known from prerequisites 6.0001/6.042) so we are able to cite the correctness of that algorithm.

**Time analysis:** Binary search takes $O(\log n)$ trials/checked indices. In order to check if a given date $i$ is the target date, we use the machine at most twice (checking day $i$ and day $i + 1$), so we have a constant number of uses per index checked and thus $O(\log n)$ total uses.

**Rubric:**

- 6 points for considering binary search
- 2 points for correctness (including citing a prerequisite)
- 2 points for correct time analysis

- -2 points for not explaining how the comparison is done (how you determine if you are at the correct date) or explaining incorrectly
- Partial credit may be awarded

**(2)** [20 points] It turns out that Dr. Hoofenshmirtz also figured out a way to create a life-extend-inator so now he can live forever! Agent B still needs to find the day he invents the Rule-inator, but he can no longer assume that the Rule-inator will be used within $n$ days from today. Give a way for Agent B to save the world using the time machine at most $O(\log x)$ times, where $x$ is Agent B's target date (the day before the machine is used).

Recall that $x$ is not known to Agent B, so while the runtime of your solution will depend on $x$, the method Agent B uses cannot depend on knowledge of $x$. Be sure to analyze the correctness of your solution and justify the number of time-machine uses it involves.

**Solution:** For this problem, we use a variant on binary search also known as *exponential search*. We first work to determine a good upper bound, $u$, on $x$ and then we run binary search on the $u$ days following today. We will call "today" day 0 and begin by checking day 1 to see if the machine has been used yet, then day 2, then day 4, and so on such that, on our $i^{th}$ check, we try day $2^i$. We take the upper bound $u$ to be the first day we check that we find the machine has been used. The crucial point is that, with this stopping rule, $u \leq 2x$. In the second phase, we run binary search on the interval $(0, u)$ as we did in part (1).

**Correctness**: The machine is used a finite number of days from now, so if we take enough trial steps we will eventually find a day that occurs after the Rule-inator has been used. We know day $x$ must occur before this, so we are in the same situation as in part (1) and from here we are able to rely on the correctness of that part to determine $x$.

**Time analysis**: Let $x$ be the time the Rule-inator is used. Phase 1 terminates after $\lceil \log x \rceil + 1 = O(\log x)$ steps. Phase 2 is a binary search inside the interval $(0, u)$ which takes $O(\log u) = O(\log x)$ steps.

**Rubric:**

- 10 points for description of a correct algorithm
- 4 points for correctness
- 6 points for running time analysis
- Partial credit may be awarded

**Problem 1-3.** [30 points] **Oh, mamma mia!**

Dophie was rummaging around her attic when she found an old diary that her mother Sonna used to keep. She was so excited to read it that she rushed down the stairs and accidentally dropped the diary causing some of the pages to fall out! Dophie knows that the first date in the diary was $d$, and she wants to figure out the date $x$ of the first missing page. Thankfully, Sonna is known to be organized, so all of her $n$ entries were dated with non-negative integers and sorted by date in ascending order. Also, Sonna wrote exactly one diary entry per day every day, so the dates in her diary are distinct and with no gaps. So, for example, if Sonna's diary entries were originally dated as $[5, 6, 7, 8, 9, 10, 11]$, after Dophie's fall, the diary entries could be $[5, 6, 8, 9, 11]$ where in this case we have $d = 5$ given to us and we see that $x = 7$. Note also that the first entry could be one of the missing ones so, after the fall, the diary entries could be $[6, 8, 10]$ with $d = 5$. In this case, we would have that $x = 5$.

For notation, let $D$ be an array with the current diary entries and $d$ be the date of the first entry to the original diary. As above $n$ is the number of entries in the original diary.

**(1)** [5 points] Given $D, d$ and $j$, describe an $O(1)$ time method to determine whether the first date missing is after date $j$. Be sure to analyze correctness and runtime of your solution.

**Solution:** Let $x$ be the missing date. We compare $D[j]$ to $j$ and say that $D[j] > x$ if $D[j] - d > j$. Otherwise $D[j] < x$.

**Correctness:** This follows as $x$ is defined as the first missing date so, assuming that $D[0] = d$, the index of $x$, if it had been in the diary, would have been $x - d$. Similarly, any date $y > d$ prior to $x$ is in page $y - d$. On the other hand, if a date $y$ is not in page $y - d$, that means that there is a prior date missing so $y > x$.

**Time analysis:** We just have to make one comparison, $D[j] - d$ to $j$ and this is $O(1)$ time.

**Rubric:**

- 5 points if correct solution and time analysis
- 4 points if correct solution and justification but incorrect time analysis
- 3 points if correct solution and incorrect justification
- 0 points if incorrect solution

**(2)** [12 points] Describe an algorithm to find the date $x$ of the first missing page in $\log(n)$ time.

**Solution:** Given part (1), given a page $j$, we can query, in $O(1)$ time, whether the missing page dated $x$ comes before or after the date of page $j$. This means that we can perform binary search to look for $x$.

As an edge case, we need to see whether the missing page was the first page in the diary. To do this, we simply compare $D[0]$ to $d$. If $D[0] > d$, that means that the missing page was dated $d$.

**Correctness:** Follows immediately from the correctness of the query described in Part (1) together with the correctness of the Binary Search algorithm.

**Time analysis:** See next part for recurrence and runtime.

**Rubric:**

- 12 points for description of a correct algorithm
- -2 points for incorrectly handling edge case where the first page is missing.
- Partial credit may be awarded

**(3)** [3 points] Write and solve the recurrence of the algorithm you proposed.

**Solution:**

$$T(n) = T(n/2) + O(1)$$

Here, we have $n^{\log_2 1} = n^0 = \Theta(1 \log^0 n)$, so we are in case 2 of Master's Theorem. Hence, the runtime of the recurrence is $\Theta(\log n)$

**Rubric:**

- 3 points for correct recurrence and runtime

**(4)** [10 points] Code and submit.

**Rubric:**

- This part is automatically graded at `alg.mit.edu`.