Data Structures and Introduction to Algorithms - 20229

Maman 11

Yehonatan Simian
206584021

Question 1 (25%)

Section 1

Let's compare insertion sort and merge sort assuming that, for input of size n, insertion sort execution takes $8n^2$ steps, whilst merge sort execution takes $64n \log n$ steps. For which values of n does insertion sort run faster than merge sort?

Solution Let's find n for which $8n^2 < 64n \lg n$:

$$8n^2 < 64n \lg n$$
$$n < 8 \lg n$$

Using desmos we can find that the solutions to this equations are $\{1.1, 43.559\}$, hence insertion sort is faster for $1 \leq n \leq 43$. ■

Section 2

This time, we are required to find n for which $100n^2 < 2^n$:

$$100n^2 < 2^n$$
$$n^2 < 0.02 \cdot 2^n$$

Let's use our friend desmos again to find out that the solution to the equation is $\{14.325\}$, hence the smallest value of n for which $100n^2$ is faster than 2^n is $n = 15$. ■

Question 2 (25%)

Section 1

Let's arrange the given functions in an descending growth rate. I will use $f(x) > g(x)$ notation to describe $f(x) = \Omega(g(x))$ and $f(x) = g(x)$ notation to describe $f(x) = \Theta(g(x))$ in order to maintain readability.

Solution Below are the relationships between the functions' growth rates. I tried adding the equation number that proves each relation, but it make everything much less readable and I decided to remove it, so... trust me, I guess:

$$
\begin{array}{ccccccccccc}
 & 2^{2^{n+1}} & > & 2^{2^n} & > & (n+1)! & > & n! & > & n2^n & > & e^n \\
> & 2^n & > & 1.5^n & > & (\lg n)! & > & n^{\lg(\lg(n))} & = & \lg(n)^{\lg(n)} & > & n^3 \\
> & n^2 & = & 4^{\lg(n)} & > & n\lg(n) & = & \lg(n!) & > & 2^{\lg(n)} & = & n \\
> & (\sqrt{2})^{\lg(n)} & > & 2^{\sqrt{2\lg(n)}} & > & \lg^2(n) & > & \ln(n) & > & \sqrt{\lg(n)} & > & \ln(\ln(n)) \\
> & 2^{\lg^*(n)} & > & \lg^*(n) & = & \lg^*(\lg(n)) & > & \lg(\lg^*(n)) & > & n^{1/\lg(n)} & = & 1
\end{array}
$$

Section 2

We need a non-negative function $f(n)$ whose growth rate is not less than $2^{2^{n+1}}$ and no more than 1.

Solution First of all, let's find a function whose growth rate is greater than $2^{2^{n+1}}$. That would be easy: $2^{2^{2^{n+1}}}$. Now, let's find a function whose growth rate is less than 1. It may sound tricky, but all we need is a descending function whose asymptote is 0, e.g. $\frac{1}{n}$.

We reached the fun part - let's fuse these two functions together (illustration below). In return we get a function that alternates between growing super-fast and decreasing super-saiyan-2-fast, therefore there is *no* asymptotic behavior to this function and it fulfils the requirements of not grow/decrease super-saiyan-ultra-instinct fast:

$$
f(n) = \begin{cases} 2^{2^{2^{n+1}}} & n \text{ is even} \\ \frac{1}{n} & n \text{ is odd} \end{cases}
$$

Then, for each $1 \le i \le 30$, we find out that:

$$
\lim_{n\to\infty} \frac{f(n)}{g_i(n)} \begin{cases} \ge \lim_{n\to\infty} \frac{f(n)}{g_1(n)} = \lim_{n\to\infty} \frac{2^{2^{2^{n+1}}}}{2^{2^{n+1}}} = \infty & n \text{ is even} \\ \le \lim_{n\to\infty} \frac{f(n)}{g_{30}(n)} = \lim_{n\to\infty} \frac{1}{n} = 0 & n \text{ is odd} \end{cases}
$$

Therefore $f(n)$ is not $O(g_i(n))$ and also not $\Omega(g_i(n))$ for each $1 \le i \le 30$ ■

Figure 1: Fusion Illustration of $2^{2^{2^{n+1}}}$ and $\frac{1}{n}$

Question 3 (25%)

Let's discuss a sorting algorithm that, for an input list of size n, sorts n/k sub-lists of size k using insertion sort ($\Theta(n \lg n)$), and then merges them to one list using merge-sort's merge procedure.

Section 1

Show that it is possible to sort n/k sub-lists of size k each using insertion sort in worst-case time complexity of $\Theta(nk)$.

Proof Sorting a list of size k using insertion sort takes time $\Theta(k^2)$ at worst case, hence sorting n/k lists will take a worst-case time of $\Theta(\frac{n}{k}) \cdot k^2 = \Theta(nk)$. ■

Section 2

Show that it is possible to merge the n/k sub-lists is worst-case time complexity of $\Theta(n \lg(\frac{n}{k}))$.

Proof The merge-sort algorithm generally takes time $\Theta(n \lg n)$ because the depths of the merge tree is $\lg(n)$, and merging each pair of sub-lists takes linear time. However, in our scenario we already have sorted sub-lists of size k, hence we can skip the first $\lg(k)$ levels of the tree, leaving us with only $\lg(n) - \lg(k) = \lg(\frac{n}{k})$ levels of the tree (each still takes linear time to merge), therefore the overall merging time complexity is $\Theta(n \lg(\frac{n}{k}))$. ■

Section 3

Given that the worst case time complexity of this algorithm is $\Theta(nk + n \lg(\frac{n}{k}))$, what is the largest asymptotic value of k as a function of n, for which the asymptotic time complexity of our algorithm is identical to the original merge-sort algorithm?

Proof The expression inside the Θ is made of two terms: nk and $n \lg(\frac{n}{k})$. Let's notice that the former has a larger value as k increases, whilst the latter has smaller values as k increases; e.g. for $k = 1$ we get n in the first term and $n \lg(n)$ in the second term, whilst for $k = n$ we get n^2 in the first term and n in the second term. Thus, the largest value of k as a function of n that will keep the first term $\Theta(n \lg(n))$ is $k = \Theta(\lg(n))$. ■

Section 4

How should k be chosen in practice?

Solution Let c_1, c_2 be the coefficients of the two terms nk and $n\lg(\frac{n}{k})$ respectively within the asymptotic notation of Θ. In order to optimize the time complexity, we need k values such that $c_1 n = c_2 \cdot \frac{n}{k}$. Kal lir'ot that any constant k fulfils this requirement, hence a constant k should be chosen in practice.

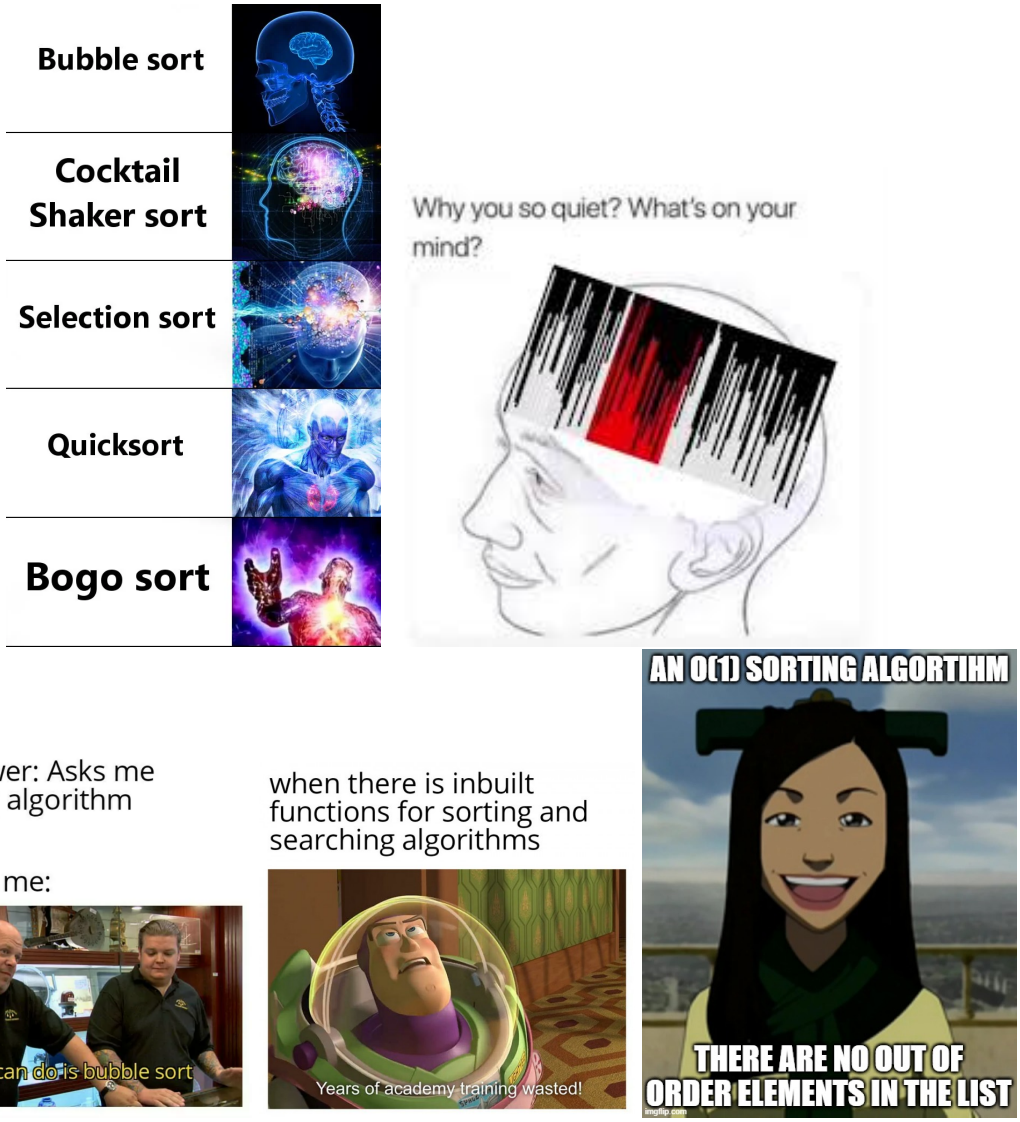

Figure 2: Sorting Memes

Question 4 (25%)

Find upper and lower asymptotic bounds for $T(n)$ in each of the following recursive formulas:

Section 1

$T(n) = 3T(n/2) + n \lg n$

Solution According to the master theorem (case 1), given $a = 3$, $b = 2$, $f(n) = n \lg n$, we can chose $\varepsilon = 0.5$ and get $f(n) = n \lg n = O(n^{\log_2 3 - 0.5 \approx 1.08})$, hence $T(n) = \Theta(n^{\lg 3})$. ■

Section 2

$T(n) = 5T(n/5) + n/\lg n$

Solution Let's assume that $\exists c > 0$ s.t. $T(n) \leq cn \lg n$, then

$$\begin{aligned} T(n) &= 5T(n/5) + n \lg n \\ &\leq cn \lg n - cn \lg 5 + n/\lg n \\ &= cn \lg n + n(\frac{1}{\lg n} - c \lg 5) \\ &\leq cn \lg n \end{aligned}$$

The equation above proves that $T(n) = O(n \lg n)$. Now let's look at a lower bound of $T(n)$, assuming that $T(n) \geq cn^{1-\varepsilon}$ for arbitrary $c > 0$ and $\varepsilon > 0$:

$$\begin{aligned} T(n) &= 5T(n/5) + n/\lg n \\ &\geq 5c/5^{1-\varepsilon} n^{1-\varepsilon} + n/\lg n \\ &= 3^{\varepsilon} cn^{1-\varepsilon} + n/\lg n \end{aligned}$$

The equation above is no greater than $cn^{1-\varepsilon}$ if $3^{\varepsilon} + n^{\varepsilon}/(c \lg n) \geq 1$ which is true since $\lg n \in o(n^{\varepsilon})$, hence $T(n) = \Omega(n^{1-\varepsilon})$.

We found out that $T(n) = O(n \lg n)$ and $T(n) = \Omega(n^{1-\varepsilon})$, therefore $T(n) = \Theta(n)$. ■

Section 3

$T(n) = 4T(n/2) + n^2\sqrt{n}$

Solution According to the master theorem (case 3), given $a = 4$, $b = 2$, $f(n) = n^{5/2}$, we can chose $\varepsilon = 0.5$ and get $f(n) = n^{5/2} = \Omega(n^{\log_2 4 + 0.5 = 2.5})$. Since $af(\frac{n}{b}) = 4 \cdot (\frac{n}{2})^{2.5} = \frac{n^{2.5}}{\sqrt{2}}$, we can choose $c = \frac{1}{\sqrt{2}} < 1$, hence $T(n) = \Theta(n^{2.5})$. ■

Section 5

$T(n) = 2T(n/2) + n/\lg n$

Solution Same as Section 3, we just replace 2 with 5 and find out that $T(n) = O(n \lg n)$ and $T(n) = \Omega(n^{1-\varepsilon})$ hence $T(n) = \Theta(n)$. ■

Section 6

$T(n) = T(n/2) + T(n/4) + T(n/8) + n$

Solution Let's guess using the substitution method; My senses tell me that $T(n) = \Theta(n)$. Let's find $c > 0$ for which $T(n) \leq cn$:

$$T(n) = T(n/2) + T(n/4) + T(n/8) + n = \frac{cn}{2} + \frac{cn}{4} + \frac{cn}{8} + n = \frac{7cn}{8} + n$$

The final term $\frac{7cn}{8} + n$ is smaller than cn for $c \geq 8$. Now, proving that $T(n) = \Omega(n)$ is almost trivial:

$$T(n) = T(n/2) + T(n/4) + T(n/8) + n \geq n$$

We found that $T(n) = O(n)$ and $T(n) = \Omega(n)$ therefore $T(n) = \Theta(n)$. ■

Section 7

$T(n) = T(n-1) + 1/n$

Solution This is an interesting one. The recurrence relation above is none else but the harmonic series! Let's prove this by induction: let $H_n = \sum_{i=1}^{n} \frac{1}{i}$. For the base case, $T(1) = 1 = H_1$. Let's assume that $T(n) = H_n$; then $T(n+1) = H^n + \frac{1}{n+1} = H_{n+1}$, and the induction is complete. The harmonic series sums to $\lg n$, thus $T(n) = \Theta(\lg n)$. ■

Section 8

$T(n) = T(n-1) + \lg n$

Solution Similarly to the last section, we'll guess $T(n) = \Theta(n \lg n)$. Assuming that $\exists c$ s.t. $T(n) \leq cn \lg n$, we'll find that

$$\begin{aligned} T(n) &= T(n-1) + \lg n \\ &\leq c(n-1)\lg(n-1) + \lg n \\ &= cn\lg(n-1) - c\lg(n-1) + \lg n \\ &\leq cn\lg(n-1) - c\lg(n/2) + \lg n \\ &= cn\lg(n-1) - c\lg n + c + \lg n \\ &< cn\lg n - c\lg n + c + \lg n \end{aligned}$$

For the last expression to not be greater than $cn \lg n$, we need:

$$\begin{aligned} -c\lg n + c + \lg n &\leq 0 \\ c &\leq (c-1)\lg n \\ \frac{c}{c-1} &\leq \lg n \end{aligned}$$

The final equation is correct for $c = 2, n \geq 4$, hence $T(n) = O(n \lg n)$.

Similarly, given $T(n) \geq cn \lg n + dn$, we can find that

$$T(n) \geq cn\lg n - cn - c\lg(n-1) + dn - d + \lg n$$

which is not less than $cn \lg n$ when

$$\begin{aligned} -cn - c\lg(n-1) + dn - d + \underbrace{\lg(n-1)}_{<\lg(n)} &\geq 0 \\ (d-c)n &> (c-1)\lg(n-1) + d \end{aligned}$$

The final equation is correct for $c = 1, d = 2, n \geq 2$, hence $T(n) = \Omega(n \lg n)$. We found that $T(n) = O(n)$ and $T(n) = \Omega(n)$ therefore $T(n) = \Theta(n)$. ■

Section 9

$T(n) = T(n-2) + 2\lg n$

Solution Similarly to the last section, we'll guess $T(n) = \Theta(n \lg n)$. Assuming that $\exists c$ s.t. $T(n) \leq cn \lg n$, we'll find that

$$\begin{aligned} T(n) &= T(n-2) + 2\lg n \\ &\leq c(n-2)\lg(n-2) + 2\lg n \\ &\leq c(n-2)\lg n + 2\lg n \\ &= cn\lg n + (2-2c)\lg n \end{aligned}$$

Choose $c > 1$ and the last expression proves that $T(n) = O(n \lg n)$.

Similarly, given $T(n) \geq cn \lg n + dn$, we can find that

$$T(n) \geq cn \lg n - cn - 2(c-1) \lg n + dn - 2d$$

For any $c > 0.5$ and $d \geq 2(2c-1)$ we'll get that $T(n) \geq cn \lg n$, and we have proven once again that $T(n) = \Theta(n \lg n)$. ■

Figure 3: Time Complexity Memes