The Design and Analysis of Algorithms HW1

B12508026 戴偉璿

October 24, 2025

Collaboration

This homework is done by myself. I discussed with ChatGPT for Exercise 6 and Exercise 8 (and other exercises were done independently). Besides, to make my expression more precise, I also used ChatGPT to check my grammar and Latex formatting.

Answers

1. (a) It is known that$(\log n)^k \ll n^a \ll a^{\sqrt{n}} \ll b^n \quad (k, a, b > 0,\ b > 1)$
   $\therefore\ (\log n)^{10} \ll \frac{n}{(\log n)^3} \ll n \ll n \log n \ll n \log^2 n \ll n^{3/2} \ll 64^{\sqrt{n}} \ll 2^n.$

   (b) i. $\boxed{\textbf{Disprove.}}$ To prove or disprove $n^{\frac{1}{2}} = O(n^{\frac{1}{3}})$, consider the definition of Big-O notation. $n^{\frac{1}{2}} \le cn^{\frac{1}{3}}$ where c is a constant. Dividing both sides by $n^{\frac{1}{3}}$ gives $n^{\frac{1}{6}} \le c$. As n approaches infinity, $n^{\frac{1}{6}}$ also approaches infinity, so there is no constant c that satisfies the inequality. Therefore, $n^{\frac{1}{2}} \ne O(n^{\frac{1}{3}})$.

   ii. $\boxed{\textbf{Prove.}}$ To prove or disprove $3^n = \Omega(27^{\sqrt{n}})$, consider the definition of Big-Omega notation. $3^n \ge c27^{\sqrt{n}}$ where c is a constant. Dividing both sides by $27^{\sqrt{n}}$ gives $\dfrac{3^n}{27^{\sqrt{n}}} = 3^{n-3\sqrt{n}} \ge c$. As n approaches infinity, $n - 3\sqrt{n}$ also approaches infinity, so there exists a constant c that satisfies the inequality. Therefore, $3^n = \Omega(27^{\sqrt{n}})$.

2. (a) The outer loop runs from 1 to n, and the inner loop runs from 1 to $\sqrt{i}$. So the total number of iterations is:$\displaystyle\sum_{k=1}^{n} \sqrt{k}$. To analysis this, we can use the integral method:$\displaystyle\sum_{k=1}^{n} \sqrt{k} \approx \int_1^n \sqrt{x}\,dx = \left[\frac{2}{3}x^{3/2}\right]_1^n = \frac{2}{3}(n^{3/2} - 1) = \Theta(n^{3/2}).$

   (b) The outer loop runs n times (for $i = n, \ldots, 1$). In the inner loop, starting from $j = i$ and repeatedly setting $j \leftarrow \sqrt{j}$ until $j < 2$, and it takes $\min\{k :\ i^{1/2^k} < 2\} = \min\{k :\ i < 2^{2^k}\} = \Theta(\log \log i)$ iterations.

   Hence the total work is $T(n) = \displaystyle\sum_{i=1}^{n} \Theta(\log \log i)$. For an upper bound, $T(n) \le$

$\sum_{i=1}^{n} \log\log n = n\log\log n = O(n\log\log n)$. For a lower bound, for all $i \in [\frac{n}{2}, n]$ we have $\log\log i \geq \log\log\frac{n}{2}$, thus $T(n) \geq \frac{n}{2}\log\log\frac{n}{2} = \Omega(n\log\log n)$. Therefore, $T(n) = \Theta(n\log\log n)$.

3. (a) We have $T(n) = \begin{cases} T\left(\dfrac{n}{6}\right) + T\left(\dfrac{n}{4}\right) + \dfrac{n}{2}, & n > 1 \\ 1, & n = 1 \end{cases}$

   $2T(\frac{n}{6}) + \frac{n}{2} \leq T(n) \leq 2T(\frac{n}{4}) + \frac{n}{2}$

   Solve $T(n) = 2T(\frac{n}{4}) + \frac{n}{2}$ by Master Theorem, where $a = 2, b = 4, f(n) = \frac{n}{2}$

   Therefore, $n^{\log_b a} = n^{\log_4 2} = n^{1/2}$

   Since $f(n) = \Omega(n^{\log_b a + \epsilon})$ for $\epsilon = \frac{1}{2}$ and $af(\frac{n}{b}) \leq cf(n)$ for $c = \frac{3}{4}$, by case 3 of Master Theorem, we have $T(n) = \Theta(f(n)) = \Theta(n)$.

   Similarly, solve $T(n) = 2T(\frac{n}{6}) + \frac{n}{2}$ by Master Theorem, where $a = 2, b = 6, f(n) = \frac{n}{2}$

   Therefore, $n^{\log_b a} = n^{\log_6 2}$

   Since $f(n) = \Omega(n^{\log_b a + \epsilon})$ for $\epsilon = 1 - \log_6 2$ and $af(\frac{n}{b}) \leq cf(n)$ for $c = \frac{3}{4}$, by case 3 of Master Theorem, we have $T(n) = \Theta(f(n)) = \Theta(n)$.

   Thus, by the squeeze theorem, we have $T(n) = \Theta(n)$.

   (b) We have $T(n) = \begin{cases} 2T\left(\dfrac{n}{2}\right) + n\log n, & n > 1 \\ 1, & n = 1 \end{cases}$

   By Master Theorem: $a = 2, b = 2, f(n) = n\log n, n^{\log_b a} = n^{\log_2 2} = n$.

   $f(n) = \Theta(n\log n) = \Theta\big(n^{\log_b a}\log^1 n\big)$.

   This matches Master Theorem case 2 with $k = 1$. $T(n) = \Theta\big(n\log^{k+1} n\big) = \Theta(n\log^2 n)$.

   $\therefore T(n) = \Theta(n\log^2 n)$.

   (c) We have $T(n) = \begin{cases} 4T\left(n^{1/4}\right) + \log n, & n > 4 \\ 2, & n \leq 4 \end{cases}$

   Set $m = \log_2 n$, so $n = 2^m$. Then $\log_2\left(n^{1/4}\right) = \frac{1}{4}\log_2 n = \frac{m}{4}$.

   Let $S(m) = T(2^m)$. The recurrence becomes $S(m) = \begin{cases} 4S\left(\frac{m}{4}\right) + m, & m > 2 \\ 1, & m \leq 2 \end{cases}$

   Apply Master Theorem: $a = 4, \quad b = 4, \quad f(m) = m, \quad m^{\log_b a} = m^{\log_4 4} = m$. $f(m) = \Theta(m) = \Theta\big(m^{\log_b a}\big)$.

   This is Case 2 of the Master Theorem: $S(m) = \Theta(m\log m)$. Since $m = \log n$, $T(n) = S(\log n) = \Theta(\log n\log\log n)$.

   $\therefore T(n) = \Theta(\log n\log\log n)$.

4. Set $E(n)$ be the expected times in n elements.

   - Case 1: pivot is the smallest, $\frac{1}{n} \times n - 1$.
   - Case 2: pivot is the j-th smallest.

   It is clear that $E(n) = 0$ if $n = 1$.

   If $n > 1$, then $E(n) = \dfrac{n-1}{n} + \dfrac{1}{n}\sum_{j=1}^{n-1}[n + E(j)] = \dfrac{n-1}{n} + \dfrac{1}{n}\left[n - 1 + \sum_{k=1}^{n-1} E(k)\right] = \dfrac{2(n-1)}{n} + \dfrac{1}{n}\sum_{k=1}^{n-1} E(k), k = j - 1$

   $$nE(n) = 2(n-1) + \sum_{j=1}^{n-1} E(k)$$

   $$\Rightarrow nE(n) = 2(n-1) + \sum_{j=1}^{n-1} E(k)$$

   $$nE(n) - (n-1)E(n-1) = 2 + E(n-1)$$

   $$nE(n) - nE(n-1) = 2, E(n) - E(n-1) = \frac{2}{n}, E(n) = \sum_{k=2}^{n} \frac{2}{k} = 2(H_n - 1)$$

   $$\therefore E(n) = O(\log n)$$

5. $S_i \in [0, n^{\log_2 \log_2 -1}]$, Applying radix sort with base $= n$.
   the number of digits is:$\log_n n^{\log_2 \log_2 n} = \log_2 \log_2 n$. For each digit, we use counting sort which runs in $O(n + k) = O(n + n) = O(n)$ time. Thus the total time is $n \log \log n = O(n \log n)$.

6. Let X_{ij} be an indicator random variable, then

   $$X_{ij} = \begin{cases} 1 & \text{if elements } i \text{ and } j \text{ are compared during QuickSelect,} \\ 0 & \text{otherwise.} \end{cases}$$

   The total number of comparisons is:

   $T(n) = \sum_{1 \leq i < j \leq n} X_{ij}, \quad E[T(n)] = \sum_{1 \leq i < j \leq n} \Pr(i, j \text{ are compared})$.

   If $k \notin [i, j]$, $\Pr(i, j \text{ are compared}) \neq 0$ depending on pivot position. If $k \in [i, j]$, $\Pr(i, j \text{ are compared}) \approx \frac{2}{n}$ on average.

   Hence, the expected number of comparisons $E[T(n)]$ is derived from the recurrence relation:

   $E[T(n)] = (n-1) + \frac{1}{n}\sum_{s=0}^{n-1} E[T(s)]$,

   where $n - 1$ is the cost of pivot selection, and the sum represents the expected cost of the recursive subproblem based on the random pivot.

   Assume $E[T(n)] = c \cdot n + d$, Substitute $c \cdot n + d = (n-1) + \frac{1}{n}\left(c \cdot \frac{(n-1)n}{2} + d \cdot n\right)$, $c \cdot n + d = (n-1) + c \cdot \frac{n-1}{2} + d$, $c \cdot n = n - 1 + c \cdot \frac{n-1}{2}$

As $n \to \infty$, $c \approx 2$, thus $E[T(n)] = O(n)$.

Thus, the expected time complexity of QuickSelect is $E[T(n)] = O(n)$.

7. For sequences X (length n) and Y (length m), the SCS length is computed using dynamic programming.

   Define $dp[i][j]$ as the SCS length for $X[1..i]$ and $Y[1..j]$.

   - Initialize $dp[0][j] = j$ for $j = 0$ to m, and $dp[i][0] = i$ for $i = 0$ to n.
   - For $i = 1$ to n and $j = 1$ to m:
     - If $X[i] = Y[j]$, $dp[i][j] = dp[i-1][j-1] + 1$.
     - Otherwise, $dp[i][j] = \min(dp[i-1][j] + 1, dp[i][j-1] + 1)$.

   Time complexity: $O(n \cdot m)$, space complexity: $O(n \cdot m)$.

8. Consider an n-character sequence X and an m-character sequence Y. Define $dp[i][j]$ as the minimum cost to convert the prefix $X[1..i]$ into the prefix $Y[1..j]$.

   Initialize the dynamic programming table:

   - Set $dp[0][j] = 2 \cdot j$ for $j = 0$ to m, representing the cost of inserting all characters of $Y[1..j]$ into an empty string.
   - Set $dp[i][0] = 2 \cdot i$ for $i = 0$ to n, representing the cost of deleting all characters of $X[1..i]$.

   For each $i = 1$ to n and $j = 1$ to m, compute $dp[i][j]$ as follows:

   - If $X[i] = Y[j]$, set $dp[i][j] = dp[i-1][j-1]$, since no operation is needed.
   - Otherwise, set $dp[i][j] = \min(dp[i-1][j-1] + 3, dp[i-1][j] + 2, dp[i][j-1] + 2)$, where:
     - $dp[i-1][j-1] + 3$ is the cost of replacing $X[i]$ with $Y[j]$.
     - $dp[i-1][j] + 2$ is the cost of deleting $X[i]$.
     - $dp[i][j-1] + 2$ is the cost of inserting $Y[j]$.

   The final answer is $dp[n][m]$, which represents the minimum cost to convert X into Y.