

2025 Algorithm Midterm Reference Answers

1. (5%)

- Answer: 3%
- Reason: 2%

2. (5% + 5%)

- $T(n) = T(n/5) + T(3n/4) + O(n)$
 - Assume: $T(n) = O(n) \leq an$
 - $T(n) \leq an/5 + 3an/4 + cn$
 - $= (c + 19a/20)n$
 - Let $a = 20c$
 - $T(n) \leq (c + 19a/20)n = 20cn = an$
 - $T(n) = O(n)$
- $T(n) = T(n/3) + T(3n/4) + O(n)$
 - Use recursion tree method we can see:
 - $T(n) \leq n + 13/12n + (13/12)^2n + \dots$
 - There are at most $\log_{\frac{4}{3}} n$ terms
 - $T(n) \leq \frac{1 - \frac{13}{12}^{\log_{\frac{4}{3}} n}}{1 - \frac{13}{12}} = O(n^{\log_{\frac{4}{3}} \frac{13}{12}}) = O(n^p)$
 - where $1 < p < 2$

3. (15%)

We analysis the case in which the i_{th} operation is TABLE-DELETE.

In this case, $num_i = num_{i-1} - 1$, and

Case 1: (4%)

$$\alpha_{i-1} > \frac{1}{2} \text{ and } \alpha_i \geq \frac{1}{2}$$

- We have $c_i = 1$, $size_i = size_{i-1}$

- Therefore,

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + (2 \cdot num_i - size_i) - (2 \cdot num_{i-1} - size_{i-1}) \\
&= 1 + (2 \cdot (num_{i-1} - 1) - size_{i-1}) - (2 \cdot num_{i-1} - size_{i-1}) \\
&= -1
\end{aligned}$$

Case 2: (2%)

$$\alpha_{i-1} = \frac{1}{2} \text{ and } \alpha_i < \frac{1}{2}$$

- We have $c_i = 1$, $size_i = size_{i-1}$

- Therefore,

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + \left(\frac{size_i}{2} - num_i \right) - (2 \cdot num_{i-1} - size_{i-1}) \\
&= 1 + \left(\frac{size_{i-1}}{2} - (num_{i-1} - 1) \right) - (2 \cdot num_{i-1} - size_{i-1}) \\
&= 2 + \frac{3}{2} size_{i-1} - 3 \cdot num_{i-1} \\
&= 2 + \frac{3}{2} size_{i-1} - 3 \cdot \alpha_{i-1} \cdot size_{i-1} \\
&\leq 2 + \frac{3}{2} size_{i-1} - \frac{3}{2} \cdot size_{i-1} \\
&= 2
\end{aligned}$$

Case 3: (4%)

$$\frac{1}{4} < \alpha_{i-1} < \frac{1}{2} \text{ and } \frac{1}{4} \leq \alpha_i < \frac{1}{2}$$

- We have $c_i = 1$, $size_i = size_{i-1}$

- Therefore,

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + \left(\frac{size_i}{2} - num_i \right) - \left(\frac{size_{i-1}}{2} - num_{i-1} \right) \\
&= 1 + \left(\frac{size_i}{2} - num_i \right) - \left(\frac{size_i}{2} - (num_{i-1} + 1) \right) \\
&= 2
\end{aligned}$$

Case 4: (5%)

$$\frac{1}{4} = \alpha_{i-1} \text{ and } \alpha_i < \frac{1}{4}$$

- Then we have $c_i = num_i + 1$, for 1 deletion and num_i moving, and $size_i = \frac{1}{2}size_{i-1}$
- And furthermore: $\frac{size_i}{2} = \frac{size_{i-1}}{4} = num_{i-1} = num_i + 1$
- Therefore,

$$\hat{c}_i = c_i + \Phi_i - \Phi_{i-1}$$

$$\begin{aligned} &= (num_i + 1) + \left(\frac{size_i}{2} - num_i\right) - \left(\frac{size_{i-1}}{2} - num_{i-1}\right) \\ &= (num_i + 1) + ((num_i + 1) - num_i) - (2 \cdot (num_i + 1) - (num_i + 1)) \\ &= 1 \end{aligned}$$

4. (5% + 5%)

The proof need to be done with induction, which means a base case and an induction step.

(a)

- Base case: $i = 0$, there are indeed $2^0 = 1$ node in level 0.
- Assume the statement stay true for $i = n$.
- The next level $n + 1$ will at most have $2^n \times 2 = 2^{n+1}$ nodes from the n -th level.
Therefore, the statement stays true for $i = n + 1$.
- Proved by induction.

(b)

- Base case: $k = 1$, there are indeed $2^1 - 1 = 1$ node in the tree.
- Assume the statement is true for $k = n$.
- For a depth $k = n + 1$ tree, there are $2^n - 1 + 2^{n+1-1} = 2^{n+1} - 1$ nodes (from (a)). Therefore, the statement stays true for $k = n + 1$.
- Proved by induction.

5. (10%)

<https://hackmd.io/@KentLee/Bk7anZr01x> (<https://hackmd.io/@KentLee/Bk7anZr01x>).

6. (6% + 6%)

- Correctness - prove by induction

- Base case: n 1-digit integers can be obviously sorted by radix-sort(counting sort).
 - Assume $n k - 1$ digit integers can be sorted correctly.
 - When sorting $n k$ -digit integers, they are first sorted by the least $k - 1$ significant digits.
 - When 2 integers have different k -th digits, they will be correctly sorted by counting sort.
 - When 2 integers have the same k -th digit, the order remains the same since the counting sort is stable.
 - Therefore, $n k$ -digit integers can be correctly sorted by radix-sort.
- Linear time
 - The time complexity of the counting sort is $O(n)$.
 - For each digit, we do counting sort once.
 - $T(n) = k \times O(n) = O(kn) = O(n)$ since k is constant.

7. (8%)

Radix Sort is not comparison-based (4%) — it works in a completely different way:

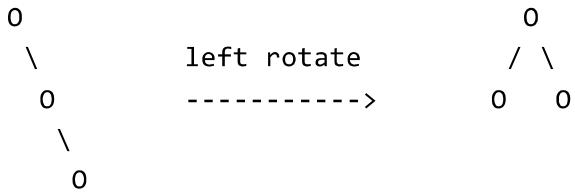
- It treats the input as sequences of digits over a finite alphabet (like base-10 or base-256).
- The algorithm sorts numbers digit by digit, under the assumption that the digit count is constant..
- It uses a linear-time stable sort (like Counting Sort) as a subroutine on each digit.

Notes: +2 points for any of the above mentioned.

8. (5% + 5%)

Rotation changes the depth of the left subtree and the right subtree.

For example, in this case



the left rotation reduces the depth by one.

9. (10%)

- No, using groups of 3 breaks the worst-case linear time guarantee.
- Suppose now that we use groups of size 3, because we will still know that the median of medians is less than at least 2 elements from half of the $\lceil \frac{n}{3} \rceil$ groups, so, it is greater than roughly $\frac{2n}{6}$ of the elements. Therefore, we are never calling it recursively on more than $\frac{4n}{6}$ elements.

So we have that the recurrence we are able to get is:

$$T(n) = T\left(\lceil \frac{n}{3} \rceil\right) + T\left(\frac{4n}{6}\right) + O(n) \geq T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + O(n)$$

Then, we can show $T(n) \geq cn \log n$ by substitution method:

$$T(n) \geq c\left(\frac{m}{3}\right) \log\left(\frac{m}{3}\right) + c\left(\frac{4m}{6}\right) \log\left(\frac{4m}{6}\right) + O(m) \geq cm \log m + O(m)$$

- Therefore, we have that it grows more quickly than linear.

Notes: If you cite formula $T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{3n}{4}\right) + O(n)$, you need to provide the complexity result to get full marks.

10. (10%)

Bottom up approach.

A[-, 10, 9, 5, 8, 3, 2, 4, 6, 7, 1]