Quiz 1B

CS/CE 412/471 Algorithms: Design and Analysis, Spring 2025

27 Jan, 2025. 3 questions, 35 points, 5 printed sides

Instructions:

1. Enter your name and ID below and at the top of every side.
2. You have 60 minutes to complete this quiz.
3. You may keep a calculator with you. Submit all other devices with your bag at the front of the classroom.
4. Solve the problems by hand in clear and legible handwriting on your own paper.
5. Provide precise and concise solutions.
6. Consulting or copying from another student constitutes a violation of academic integrity. Any infractions will result in disciplinary action, including a failing grade for the quiz.
7. For your solution to be graded, submit this problem sheet along with your solution.

Student Name: ______________________________

Student ID: ______________________________

viel Spaß!

1. Algorithm Analysis and Correctness

(a) $\boxed{\text{10 points}}$ Consider the following pseudocode for finding the minimum element in an array:

```
MinElement(A, n)
1. min = A[1]
2. for i = 2 to n
3.     if A[i] < min
4.         min = A[i]
5. return min
```

Prove the correctness of this algorithm using the loop invariant method.

Solution: We use the following loop invariant:

> At the start of each iteration of the loop, `min` contains the smallest value among the first $i-1$ elements of the array.

Proof. The algorithm is correct.

1. **Initialization:** Before the first iteration, `min` is initialized to $A[1]$. Since only one element has been considered at this point, the invariant holds.
2. **Maintenance:** At each iteration, the algorithm compares the current element $A[i]$ with `min`. If $A[i] <$ `min`, `min` is updated to $A[i]$; otherwise, `min` remains unchanged. Thus, the invariant continues to hold.
3. **Termination:** At the end of the loop, all n elements have been examined. By the invariant, `min` contains the smallest element in $A[1 \dots n]$.

□

(b) $\boxed{\text{5 points}}$ Analyze its running time in terms of n, where n is the size of the array. Provide the best-case, worst-case, and average-case time complexities.

Solution:

Proof. The time complexity of each case is $\Theta(n)$.

1. The number of iterations, $n-1$, is independent of the data.
2. A constant amount of work takes place in each iteration.
3. Only Line 4 depends on the data. This affects the amount of constant work that takes place in an iteration.

□

2. Asymptotic Notation

(a) $\boxed{\text{5 points}}$ Arrange the following functions in increasing order of growth rate. Justify your answer.

$$f_1(n) = \log n,\ f_2(n) = n,\ f_3(n) = 2^n,\ f_4(n) = n^2.$$

Solution: Order: $f_1(n) = \log n < f_2(n) = n < f_4(n) = n^2 < f_3(n) = 2^n$.

Proof. To compare the growth rates of these functions, we analyze the ratio of their growth rates as $n \to \infty$ using the following limit test:

$$\lim_{n\to\infty} \frac{f_i(n)}{f_j(n)}.$$

1. Compare $f_1(n) = \log n$ and $f_2(n) = n$:

$$\lim_{n\to\infty} \frac{\log n}{n} = 0.$$

   Since the limit is 0, $f_1(n) = \log n$ grows slower than $f_2(n) = n$.

2. Compare $f_2(n) = n$ and $f_4(n) = n^2$:

$$\lim_{n\to\infty} \frac{n}{n^2} = \lim_{n\to\infty} \frac{1}{n} = 0.$$

   Since the limit is 0, $f_2(n) = n$ grows slower than $f_4(n) = n^2$.

3. Compare $f_4(n) = n^2$ and $f_3(n) = 2^n$:

$$\lim_{n\to\infty} \frac{n^2}{2^n}.$$

   Using L'Hôpital's Rule:

$$\lim_{n\to\infty} \frac{n^2}{2^n} = \lim_{n\to\infty} \frac{2n}{2^n \ln 2} = \lim_{n\to\infty} \frac{2}{2^n (\ln 2)^2} = 0.$$

   Since the limit is 0, $f_4(n) = n^2$ grows slower than $f_3(n) = 2^n$.

□

(b) $\boxed{\text{5 points}}$ Prove or disprove: $n^2 + \log n \in O(n^2)$.

Solution:

Proof. $n^2 + \log n \in O(n^2)$.

1. We need constants $c > 0$ and $n_0 > 0$ such that:

$$n^2 + \log n \le c \cdot n^2, \quad \forall n \ge n_0.$$

2. Dividing both sides by n^2:

$$1 + \frac{\log n}{n^2} \le c.$$

3. As $n \to \infty$, $\frac{\log n}{n^2} \to 0$ (since quadratic growth dominates logarithmic growth).

4. One choice for n_0 and c is $n_0 = 10, c = 2$.

$$1 + \frac{\log 10}{10^2} = 1 + \frac{1}{100} \leq 2$$

5. Because quadratic growth dominates logarithmic growth, the value of $\frac{\log n}{n^2}$ will only decrease as n increases.

□

3. Recurrence Relations and Lower Bounds

(a) $\boxed{\text{5 points}}$ Solve the following recurrence relation using the unfolding method:

$$T(n) = 3T\left(\frac{n}{3}\right) + n.$$

Show all steps and derive a closed-form solution.

Solution:

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

$$\begin{aligned} T(n) &= 3T\left(\frac{n}{3}\right) + n \\ &= 3\left(3T\left(\frac{n}{9}\right) + \frac{n}{3}\right) + n = 9T\left(\frac{n}{9}\right) + 2n \\ &= \dots \\ &= 3^k T\left(\frac{n}{3^k}\right) + k \cdot n \end{aligned}$$

Base case: $T(1) = \Theta(1) = c$. When $\frac{n}{3^k} = 1 \implies k = \log_3 n$:

$$T(n) = cn + n \cdot \log_3 n = \Theta(n \log n)$$

□

(b) $\boxed{\text{5 points}}$ Explain why any comparison-based sorting algorithm must make at least $\lceil \log_2(n!) \rceil$ comparisons in the worst case.

Solution:

Proof.

1. In the worst case, an *optimal* comparison-based algorithm to sort n elements must make h comparisons where h is the height of the corresponding decision tree.
2. The decision tree has at least $n!$ leaves.
3. The decision tree has at most 2^h leaves.

4. Therefore, $n! \leq 2^h \implies h \geq \log_2 n!$. So the smallest integer value of h is $\lceil \log_2 n! \rceil$.
5. A general comparison-based algorithm, optimal or otherwise, to sort n numbers therefore makes at least $\lceil \log_2 n! \rceil$ comparisons in the worst case.

$\square$