# Assignment 1 Answers

## COT 6405 - Introduction to Theory of Algorithms

1. Exercise 2.3-1: Using Figure 2.4 as a model, illustrate the operation of merge sort on the array A = 3, 41, 52, 26, 38, 57, 9, 49.

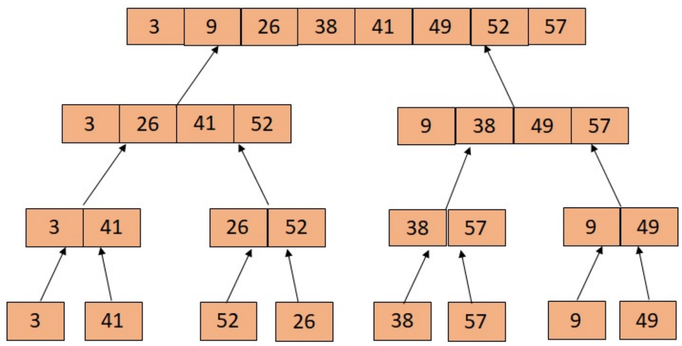


Figure 1: Mergesort question 1

2. Exercise 2.3-6: Observe that the while loop of lines 5-7 of the INSERTION-SORT procedure in Section 2.1 uses a linear search to scan (backward) through the sorted subarray A[1...j-1]. Can we use a binary search instead of a linear search to improve the overall worst-case running time of insertion sort to $\Theta(n\lg n)$?

   **Answer:** No. Although we can search a sorted array using binary search for the position of a key element in $\Theta(\lg n)$ time, we still need to move elements A[k..j] before we insert A[k]. This step takes $\Theta(n)$ time. For n elements, the algorithm is $\Theta(n^2)$.

3. For the MERGE function, the sizes of the L and R arrays are one element longer than $n_1$ and $n_2$, respectively. Can you rewrite the merge function with the size of L and R exactly equal to $n_1$ and $n_2$?

   **Answer:** Yes, we can add conditions to handle the scenarios where $i > n_1$ and $j > n_2$. When $i > n_1$, we take elements from the right array. When $j > n_2$, we take elements from the left array.

**Algorithm 1** MERGE(A,p,q,r)

```
n_1 = q - p + 1
n_2 = r - p
Create new arrays L[1, ··· , n_1], R[1, ··· , n_2]
for i = 1 to n_1
    L[i] = A[p + i − 1]
for j = 1 to n_2
    R[j] = A[q + j]
i = 1; j =1
for k = p to r
    if i > n_1 then
        A[k] = R[j]; j = j+1
    else if j > n_2 then
        A[k] = L[i]; i = i+1
    else if L[i] <= R[j] then
        A[k] = L[i]; i = i+1
    else A[k] = R[j]; j = j+1
```

4. Prove that $e^{\frac{1}{n}} \in O(n^t)$ when $t > 0$.

   **Answer:** *Proof:* Assume $e^{\frac{1}{n}} \in O(n^t)$ when $t > 0$. By the definition of O, there exist some $c > 0$ and $n_0 > 0$ such that $e^{\frac{1}{n}} \leq cn^t$ for all $n \geq n_0$.

$$\begin{aligned} e^{\frac{1}{n}} &\leq cn^t \\ ln(e^{\frac{1}{n}}) &\leq ln(cn^t) \\ \frac{1}{n} &\leq ln(c) + ln(n^t) \\ \frac{1}{n} &\leq t\ ln(n) + c \\ 1 &\leq nt\ ln(n) + c \end{aligned}$$

   Our inequality holds when $n_0 = 1$ and $c = 1$. The minimum value of the right side is 1, and only grows. Therefore, $e^{\frac{1}{n}} \in O(n^t)$ when $t > 0$.

5. Express the function $\frac{n^3}{100} - 50n - 100lgn$ in terms of $\Theta$ notation

   **Answer:** First, we must prove the upper bound, that $\frac{n^3}{100} - 50n - 100lgn \in O(n^3)$. There must exist some $n_0 > 0$ and $c > 0$ such that

$$\frac{n^3}{100} - 50n - 100lgn \leq cn^3$$

   Let's say that $c = \frac{1}{100}$:

$$\frac{n^3}{100} - 50n - 100lgn \le \frac{n^3}{100}$$
$$-50n - 100lgn \le 0$$
$$0 \le 50n + 100lgn$$

When $n_0 = 1$, the right side is greater than 0 and only grows. Therefore, the inequality holds and $\frac{n^3}{100} - 50n - 100lgn \in O(n^3)$ when $c = \frac{1}{100}$ and $n_0 = 1$.

Second, we must prove lower bound, that $\frac{n^3}{100} - 50n - 100lgn \in \Omega(n^3)$. There must exist some $n_0 > 0$ and $c > 0$ such that

$$\frac{n^3}{100} - 50n - 100lgn \ge cn^3$$

Let's say that $c = \frac{1}{200}$:

$$\frac{n^3}{100} - 50n - 100lgn \ge \frac{n^3}{200}$$
$$\frac{n^3}{100} \ge \frac{n^3}{200} + 50n + 100lgn$$
$$\frac{n^3}{100} - \frac{n^3}{200} \ge 50n + 100lgn$$
$$\frac{2n^3}{200} - \frac{n^3}{200} \ge 50n + 100lgn$$
$$\frac{n^3}{200} \ge 50n + 100lgn$$
$$n^3 \ge 10000n + 20000lgn$$

For sufficiently large $n$, for example $n_0 = 200$, $n^3$ is a much higher growth class than $n$ or $lgn$ and will always grow much faster than their sum. The inequality holds. Therefore, $\frac{n^3}{100} - 50n - 100lgn \in \Omega(n^3)$ when $c = \frac{1}{200}$ and $n_0 = 200$.

Having proven that $\frac{n^3}{100} - 50n - 100lgn \in O(n^3)$ and $\frac{n^3}{100} - 50n - 100lgn \in \Omega(n^3)$, we have met the definition of, and proven, that $\frac{n^3}{100} - 50n - 100lgn \in \Theta(n^3)$.

6. Exercise 3.1-6 Prove that the running time of an algorithm is $\Theta(g(n))$ if and only if its worst-case running time is $O(g(n))$ and its best-case running time is $\Omega(g(n))$.

   **Answer:** Note that you need to prove two directions for the "if and only if" statement.

   If the worst-case running time of the algorithm is $O(g(n))$ and best-case running time is $\Omega(g(n))$, let the worst running time be $T_w(n)$, and the best running time be $T_b(n)$. By definition, we know that:

$$T_w(n) \le c_1 g(n)$$
$$T_b(n) \ge c_2 g(n)$$

for some values $c_1 > 0$, $c_2 > 0$ and $0 < n_0 < n$.

In addition, $T_b(n) \le T(n) \le T_w(n)$ because the worst running time should be greater than or equal to the best running time, and the overall running time $T(n)$ should be between both of or equal to either or both of the worst and best running times.

Thus $c_2 g(n) \le T(n) \le c_1 g(n)$. By definition, this proves that $T(n) \in \Theta(g(n))$.

From the other direction, if the running time of the algorithm is $\Theta(g(n))$, we know $c_2 g(n) \le T(n) \le c_1 g(n)$ for some values $c_1 > 0$ and $c_2 > 0$ and $0 < n_0 < n$.

Both the worst running time and the best running time fit this definition because they are just bound cases of $T(n)$, like so:

$$\begin{aligned} T_w(n) &\le c_1 g(n), \quad \text{thus } T_w(n) \in O(g(n)) \\ T_b(n) &\ge c_2 g(n), \quad \text{thus } T_b(n) \in \Omega(g(n)) \end{aligned}$$

Therefore, we have shown an algorithm is $\Theta(g(n))$ if and only if its worst-case running time is $O(g(n))$ and its best-case running time is $\Omega(g(n))$.

7. Which is asymptotically larger: $\lg n$ or $\sqrt{n}$ ? Please explain your reason.

   **Answer:** First, assume that either $\sqrt{n} \in O(\lg n)$ or $\lg n \in O(\sqrt{n})$. We will assume the first. There must exist some $c > 0$ and $n_0 > 0$ such that:

$$\begin{aligned} \sqrt{n} &\le c \lg n \\ 2^{\sqrt{n}} &\le 2^c n \\ \frac{2^{\sqrt{n}}}{n} &\le 2^c \end{aligned}$$

   Because $c$ is a constant, $2^c$ is a constant term. In addition, $2^{\sqrt{n}}$ grows faster than $n$, so the fraction will grow. For large $n$, the constant term will eventually be surpassed by the fraction. This inequality is false, so our initial assumption is false.

   Therefore, $\sqrt{n}$ must be asymptotically larger.

8. Prove that $n^{\lg c} \in \Omega(c^{\lg n})$, where $c$ is a constant and $c > 1$.

   **Answer:** First, assume that $n^{\lg c} \in \Omega(c^{\lg n})$. There must exist some $d > 0$ and $n_0 > 0$ such that:

$$n^{\lg c} \le d c^{\lg n}$$

   From the equality $x^{\log_a y} = y^{\log_a x}$, we know that

$$n^{\lg c} = c^{\lg n}$$

   Thus, when $n_0 = 1$ and $d_0 = 1$, the assumed inequality holds true. Therefore, $n^{\lg c} \in \Omega(c^{\lg n})$ for some constant $c$ where $c > 1$.

9. Use the definition of limits at infinity to prove $(\lg x)^p \in o(x^p)$.

Definition (limits at infinity): Let $f(x)$ be a function defined on $x > K$ for some K. Then we say that $\lim_{x\to\infty} f(x) = L$ if for every number $\epsilon > 0$ there is some number $M > 0$ such that $|f(x) - L| < \epsilon$ whenever $x > M$.

**Answer:** According to the definition of little o, $f(n) \in O(g(n))$ if and only if, for all $c > 0$, there exists $n_0 > 0$ such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0$.

Thus, to prove $(\lg x)^p \in o(x^p)$, we need to prove that for all $c > 0$, there exists $n_0 > 0$ such that $0 \le (\lg x)^p \le cx^p$ for all $n \ge n_0$.

We notice that

$$lim_{x\to\infty} \frac{(\lg x)^p}{cx^p} = 0$$

According to the definition of limits, this means that "For every number $\epsilon > 0$, there is some number $M > 0$ such that $\frac{(\lg x)^p}{cx^p} < \epsilon$ whenever $x > M$."

Here we assume $M \ge 1$ and thus $\frac{(\lg x)^p}{cx^p} > 0$ and we can omit the absolute value norm.

Substituting $\epsilon$ with $c$ and $M$ with $n_0$, this statement becomes "For every number $c > 0$, there is some number $n_0 > 0$ such that $\frac{(\lg x)^p}{cx^p} < c$ whenever $x > n_0$."

Therefore, $(\lg x)^p \in o(x^p)$.