Prof. Pietro S. Oliveto

CS217 Data Structures and Algorithm Analysis
Fall term 2025

Exercise Sheet 3

Handout: Sept 27th

Deadline: October 4th - 4pm

Question 3.1 (0.1 marks)

Consider the following input for MERGESORT:

12	10	4	2	9	6	5	25	8

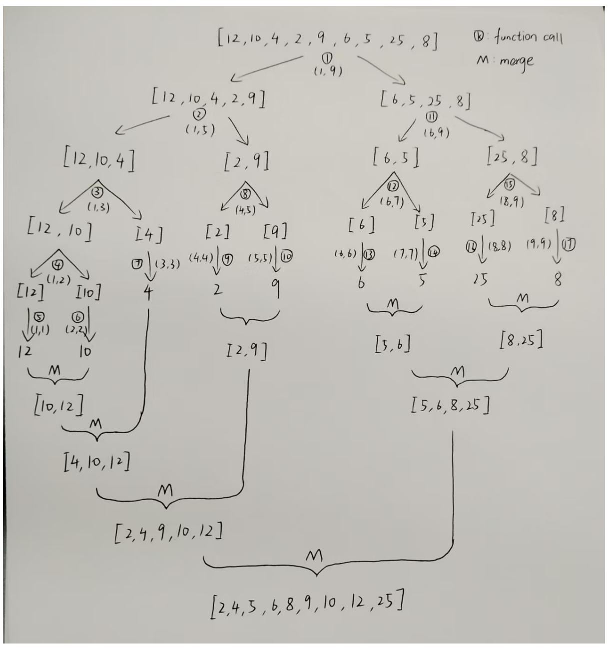
Illustrate the operation of the algorithm (follow how it was done in the lecture notes).

The algorithm's pseudocode looks like below:

```
MERGESORT(A, p, r)
1: if p < r then
2:     q = ⌊(p + r)/2⌋
3:     MERGESORT(A, p, q)
4:     MERGESORT(A, q + 1, r)
5:     MERGE(A, p, q, r)
```

We assume the MERGE algorithm in line 5 can correctly sort two arrays which are already sorted.

The operation flow is illustrated as below:



Question 3.2 (0.45 marks)

Prove using the substitution method the runtime of the MERGESORT Algorithm on an input of length n, as follows. Let n be an exact power of 2, $n = 2^k$ to avoid using floors and ceilings. Use mathematical induction over k to show that the solution of the recurrence involving positive constants $c, d > 0$

$$T(n) = \begin{cases} d & \text{if } n = 2^0 = 1 \\ 2T(n/2) + cn & \text{if } n = 2^k \text{ and } k \geq 1 \end{cases}$$

is $T(n) = dn + cn \log n$ (we always use log to denote the logarithm of base 2, so $\log = \log_2$). **Hint:** you may want to rewrite the above by replacing n with 2^k. Then the task is to prove that $T(2^k) = d2^k + c2^k \cdot k$ using the recurrence

$$T(2^k) = \begin{cases} d & \text{if } k = 0 \\ 2T(2^{k-1}) + c2^k & \text{if } k \geq 1 \end{cases}$$

Here's the proof to the runtime $T(2^k) = d2^k + c2^k \cdot k$ is correct runtime of recurrence equantion.

Base case:$k = 0, T(2^0) = d2^0 + c2^0 \cdot 0 = d$

Step: if $T(2^k) = d2^k + c2^k \cdot k$, then $T(2^{k+1}) = 2T(2^k) + c2^{k+1} = 2(d2^k + c2^k \cdot k) + c2^{k+1} = d2^{k+1} + c2^{k+1} \cdot (k+1)$

Conclusion: $\forall k \geq 0, T(2^k) = d2^k + c2^k \cdot k$

i.e. $T(n) = dn + cn \log n$ is the solution for the recurrence equation.

Question 3.3 (0.4 marks)

Use the Master Theorem to give asymptotic tight bounds for the following recurrences. Justify your answers.

1. $T(n) = 2T(n/4) + 1$
2. $T(n) = 2T(n/4) + \sqrt{n}$
3. $T(n) = 2T(n/4) + \sqrt{n} \log^2 n$
4. $T(n) = 2T(n/4) + n$

1. $T(n) = 2T(n/4) + 1$

   Here, $a = 2$, $b = 4$, and $f(n) = 1$. We have $n^{\log_b a} = n^{\log_4 2} = n^{1/2}$.

   Since $f(n) = O(n^{1/2-\epsilon})$ for $\epsilon = 1/2$, by Case 1 of the Master Theorem,

   $$T(n) = \Theta(n^{1/2}).$$

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

   Here, $a = 2$, $b = 4$, and $f(n) = n^{1/2}$. Again, $n^{\log_b a} = n^{1/2}$.

   Since $f(n) = \Theta(n^{\log_b a})$, by Case 2 of the Master Theorem,

   $$T(n) = \Theta(n^{1/2} \log n).$$

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

   Here, $a = 2$, $b = 4$, and $f(n) = n^{1/2}\log^2 n$. We have $n^{\log_b a} = n^{1/2}$.

   Since $f(n) = \Theta(n^{\log_b a}\log^2 n)$, where $k = 2$:

$$T(n) = \Theta(n^{1/2}\log^{k+1} n) = \Theta(n^{1/2}\log^3 n).$$

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

   Here, $a = 2$, $b = 4$, and $f(n) = n$. We have $n^{\log_b a} = n^{1/2}$.

   Since $f(n) = \Omega(n^{1/2+\epsilon})$ with $\epsilon = 1/2$, and the regularity condition $2f(n/4) \leq cf(n)$ for some $c < 1$ holds, by Case 3 of the Master Theorem,

$$T(n) = \Theta(f(n)) = \Theta(n).$$

Question 3.4 (0.45 marks)

Write the pseudo-code of the recursive `BINARYSEARCH(A, x, low, high)` algorithm discussed during the lecture to find whether a number x is present in an increasingly sorted array of length n. Write down its recurrence equation and prove that its runtime is $\Theta(\log n)$ using the Master Theorem.

Pseudocode:

Input: Sorted array $A[1\ldots n]$, target value x, smallest index low and largest index high
Output: Index i such that $A[i] = x$, or -1 if not found
Function `RecBinarySearch`$(A, x, low, high)$**:**

```
    if low > high then
        return −1 ;
    end
    m ← ⌊(low + high)/2⌋ ;
    if A[m] = x then
        return m;
    end
    else if A[m] < x then
        return RecBinarySearch(A, x, m + 1, high);
    end
    else
        return RecBinarySearch(A, x, low, m − 1);
    end
```

Algorithm 1: BINARY SEARCH

Recurrence equation:

$$T(n) = \begin{cases} \Theta(1) & \text{if } x = A[m] \\ T(n/2) + \Theta(1) & \text{if } x \neq A[m] \end{cases}$$

Runtime:
$a = 1, b = 2, n^{log_b a} = n^0 = 1, f(n) = \Theta(1)$
so $f(n) = \Theta(1) = \Theta(n^{log_b a} log^k n) = \Theta(log^k n)$ is true for k=0
then$T(n) = \Theta(logn)$