Exercise III, Algorithms 2024-2025

These exercises are for your own benefit. Feel free to collaborate and share your answers with other students. There are many problems on this set, solve as many as you can and ask for help if you get stuck for too long. Problems marked * are more difficult but also more fun :).

These problems are taken from various sources at EPFL and on the Internet, too numerous to cite individually.

1 Asymptotics and recursions

1 *(Previous exam question)* Give tight asymptotic bounds for the following recurrences (assuming that $T(1) = \Theta(1)$):

(i) $T(n) = 3T(n/3) + \Theta(1)$

(ii) $T(n) = 3T(n/3) + \Theta(n)$

(iii) $T(n) = 3T(n/3) + \Theta(n^2)$

(iv) $T(n) = T(n/5) + 2T(2n/5) + \Theta(n)$

(v) $T(n) = 25T(n/5) + \Theta(n^2)$

2 *(previous exam question)*

Suppose you are choosing between the following five Divide-and-Conquer algorithms:

Algorithm A solves problems of size n by dividing (in constant time) them into two subproblems each of size $n/2$, recursively solving each subproblem, and then combining the solutions in $\Theta(n^3)$ time.

Algorithm B solves problems of size n by dividing (in constant time) them into nine subproblems each of size $n/3$, recursively solving each subproblem, and then combining the solutions in $\Theta(n^2)$ time.

Algorithm C solves problems of size n by dividing (in constant time) them into ten subproblems each of size $n/3$, recursively solving each subproblem, and then combining the solutions in $\Theta(n)$ time.

Algorithm D solves problems of size n by dividing (in constant time) them into eight subproblems each of size $n/2$, recursively solving each subproblem, and then combining the solutions in constant time.

Algorithm E solves problems of size n by dividing (in constant time) them into two subproblems each of size $n - 1$, recursively solving each subproblem, and then combining the solutions in constant time.

What are the running times of each of these algorithms (in Θ notation), and which would you choose?

Maximum-Subarray Problem

3 In class we saw a divide-and-conquer algorithm for MAXIMUM SUBARRAY that runs in time $O(n \log n)$. Illustrate how it works by showing/explaining the divide, conquer and merge step of each subproblem when the algorithm is given the following input

-1	4	-1	-1	2	-5	2	1

Matrix Multiplication

4 (Exercise 4.2-1 in the book)
Use Strassen's algorithm to compute the matrix product

$$\begin{pmatrix} 1 & 3 \\ 7 & 5 \end{pmatrix} \begin{pmatrix} 6 & 8 \\ 4 & 2 \end{pmatrix}$$

Show your work.

5 (Exercise 4.2-3 in the book)
How would you modify Strassen's algorithm to multiply $n \times n$ matrices in which n is not an exact power of 2? Show that the resulting algorithm runs in time $\Theta(n^{\lg 7})$.

6 (half a *, Exercise 4.2-4 in the book)
What is the largest k such that if you can multiply 3×3 matrices using k multiplications (not assuming commutativity of multiplication), then you can multiply $n \times n$ matrices in time $o(n^{\lg 7})$? What would the running time of this algorithm be?
Note: $\lg x$ corresponds to the base 2 logarithm of x, i.e., $\lg x = \log_2 x$.

More general questions

7 *(previous exam question)* Consider the procedure Power that takes as input a number a, a non-negative integer n and returns a^n:

```
Power(a, n)
1. if $n = 0$
2.        return 1
3. if $n = 1$
4.        return a
5. $q = \lfloor \frac{n}{4} \rfloor + 1$
6. return Power$(a, q)\cdot$ Power$(a, n - q)$
```

7a *(10 pts)* Let $T(n)$ be the time it takes to invoke Power(a, n). Then the recurrence relation of $T(n)$ is

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 0 \text{ or } n = 1, \\ T(\lfloor n/4 \rfloor + 1) + T(n - \lfloor n/4 \rfloor - 1) + \Theta(1) & \text{if } n \geq 2. \end{cases}$$

Prove that $T(n) = O(n)$ **using the substitution method**.

In your proof you may ignore the floor function, i.e., you can replace $\lfloor n/4 \rfloor$ by $n/4$.

7b *(15 pts)* **Design** and **analyze** a modified procedure FastPower(a, n) that returns the same value a^n but runs in time $\Theta(\log n)$.

A solution that only works when n is a power of 2, i.e., $n = 2^k$ for some integer $k \geq 0$, gives partial credits.

(Note that a^n is *not* a basic instruction and should therefore not be used.)

8 ($*$, Exercise 4.1-5 in the book)
Use the following ideas to develop a nonrecursive, linear-time algorithm for the maximum-subarray problem. Start at the left end of the array, and progress toward the right, keeping track of the maximum subarray seen so far. Knowing a maximum subarray of $A[1 \,..\, j]$, extend the answer to find a maximum subarray ending at index $j + 1$ by using the following observation: a maximum subarray of $A[1 \,..\, j + 1]$ is either a maximum subarray of $A[1 \,..\, j]$ or a subarray of the form $A[i \,..\, j + 1]$, for some $1 \leq i \leq j + 1$. Determine a maximum subarray of the form $A[i \,..\, j + 1]$ in constant time based on knowing a maximum subarray ending at index j.