University of Zurich

Department of Informatics
Database Technology Group
Prof. Dr. M. Böhlen

Informatics II
Exercise 4

Spring Semester 2022
Week 5
Divide and Conquer, Recurrences

Task 1. Short Questions
Solve all of the following subtasks without executing any code.

(a) Is the following statement true or false?
«Divide and conquer technique means that an algorithm is divided into subroutines (functions) which solve similar tasks with the goal that the algorithm is easier understood and easier to debug.»

Answer: ☐ True ☐ False

(b) Consider an arbitrary array A of $n = 543$ integers which is sorted in ascending order using the merge sort algorithm as outlined in the lecture. What will be the maximum recursion depth reached during the algorithm?

Answer:

(c) Consider the following function in C. Give the recurrence which models the running time of this function with respect to `n`.

```c
int alpha(int n) {
    if (n <= 2) {
        return 9;
    }
    else {
        return 5 + 7 * alpha(n / 3);
    }
    int t = 0;
    for (int i = 0; i < n; i++) {
        t = t + n;
    }
    printf("%d", t);
}
```

Answer:

(d) What is the asymptotic tight bound of the C function `alpha` given in subtask (c) with respect to `n`?

Answer:

Task 2. Substitution Method
Use the substitution method to prove or disprove the following assertions regarding the given recurrences:

(a) For $T(n) = 3T(2n/3) + 5n^2 + 4n$, where $T(1) = 1$, it holds that $T(n) \in O(n^2)$.

(b) For $T(n) = T(n/2) + 3T(3n/7) + 2n$, where $T(1) = 1$, it holds that $T(n) \in O(\log(n))$.

Task 3. Repeated Backward Substitution
Solve the following recurrences using the repeated substitution method:

(a) $T(n) = 5T(n/5) + 7$, where $T(1) = 1$

(b) $T(n) = 3T(n-2) + n$, where $T(1) = 1$

(c) $T(n) = T(\sqrt{n}) + \log n$, where $T(1) = 1$

Task 4. Recursion Tree Method
Consider the following recurrence:

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2T(n/2) + T(3n/4) + n^2 & \text{if } n > 1 \end{cases}$$

(a) Draw a recursion tree and use it to estimate the asymptotic upper bound of $T(n)$. Show the tree-based computations that led to your estimate.

(b) Use the substitution method to prove that your estimate in the previous subtask is correct.

Task 5. Master Method
Decide for each of the given recurrences, whether the master method is applicable (according to the rules given in the lecture). If it is not applicable, shortly explain the reason for this. If it is applicable, write down a, b, $f(n)$ and the case (1 to 3). In this case, use the master method to derive the asymptotic tight bound of the respective recurrence. (Assume $T(1) \in \Theta(1)$ for all given recurrences.)

(a) $T(n) = 32T(n/4) + \sqrt{n^5} + 2n + 7$

(b) $T(n) = 2T(n/3) \cdot \log n + 5n^4 + 1$

(c) $T(n) = 36T(n/6) + 2\sqrt{n}\log(n)$

(d) $T(n) = 2T(5n/4) + n$

(e) $T(n) = \frac{\sqrt{3}}{2}T(n/7) + 2^n$

(f) $T(n) = 21n + 7T(3n/11) \cdot 7 + 8\log(n!) + \sqrt{\log(n)} + n^2$

Task 6. Sorting Coloured Circles

Consider a matrix $M[0..m-1][0..m-1]$ ($m = 2^i$, $i \geq 1$) with dimensions a power of two. Each of the $n = m^2$ cells of M contains a coloured circle in one of exactly four colours from $C = \{red, blue, black, violet\}$. In every 2×2 submatrix which upper left cell has an even row and an even column index, all four colours are present exactly once. Apart from this condition, the circles initially are filled into the matrix in random order.

Design an algorithm in pseudo code using the *divide and conquer technique* to rearrange the circles in the matrix such that all red circles are in the left upper corner, all blue circles are in the right upper corner, all black circles are in the left lower corner and all violet circles are in the right lower corner. Your algorithm must work in place (i.e. without defining a helper matrix of identical size) by only using exchanges of circles between two cells of the matrix (you may assume that a subroutine $exchange(x, y)$ already exists). See Figure 1 for an example.

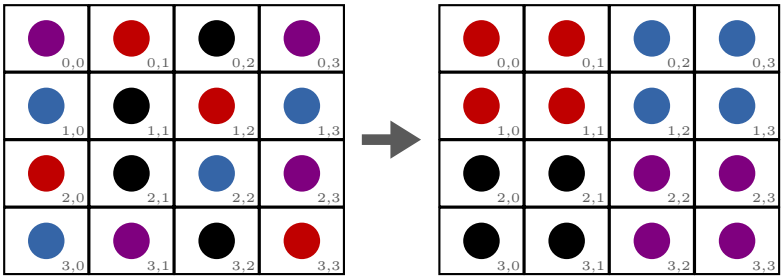

Figure 1: Example of sorting a 4×4 matrix

Task 7. Modified Merge Sort Algorithm

(a) Implement in C the merge sort algorithm as given in the lecture slides and modify the algorithm as follows:

(i) It shall be possible to choose the order of sorting (ascending or descending) through a parameter.

(ii) Any input array should first be checked whether it is already sorted in the required order using a linear search. If the former is the case, an early abort of the algorithm shall be performed.

(iii) Once the divide stage of the merge sort algorithm has reached a subproblem size (partial array length) smaller or equal to 6, the recursion should be aborted and the subproblem should be solved using the insertion sort algorithm instead.

(b) Give the best case, average case and worst case time complexity of the algorithm implemented in the previous subtask with regard to the size of the input array.