



Algorithm Theory

Sample Solution Exercise Sheet 1

Due: Friday, 24th of October 2025, 10:00 am

Exercise 1: General Instructions

(0 Points)

- Work on this exercise sheet in your group.
- **Everyone** has to submit the solution separately via Daphne.
⇒ For that, create a new folder named `exercise-01` and place your file(s) inside.
- Make sure each group members name is visible on your solution sheet (preferably Name and RZ-Account).
- Please submit the solution as PDF file (no .tex, .txt or .doc needed). Handwritten scans are okay, but make sure they are readable. The recommendation is to use L^AT_EX for submissions.
- If you have any kind of question regarding the exercise sheet, use the Zulip forum to ask.

Exercise 2: Recurrence Relation

(10 Points)

You are given the following recurrence relation:

$$T(n) \leq 2 \cdot T\left(\frac{n}{2}\right) + \sqrt{2 \cdot n}, \quad T(1) = 1.$$

- a) *Guess* a (closed-form) function $f(n)$, such that for all $n \geq 1$ it holds that $T(n) \leq f(n)$. Try to choose $f(n)$ as (asymptotically) *tight* upper bound of $T(n)$ as possible. (4 Points)
Remark: You should not just guess out of the box, instead, the idea is to find a sophisticated guess by repeatedly substituting the recurrence relation into itself.
- b) Use *induction* to show that your guess is correct. (4 Points)
Remark: You can assume that n equals 2^j for some $j \in \mathbb{N}$.
- c) So what is $T(n)$ in \mathcal{O} -notation due to your result? Can you achieve the same result using the Master Theorem? If so, explain which of the 3 cases you use, otherwise explain why it does not work. (2 Points)

Sample Solution

a)

$$\begin{aligned} T(n) &\leq 2 \cdot T\left(\frac{n}{2}\right) + \sqrt{2 \cdot n} \\ &\leq 2 \cdot \left(2 \cdot T\left(\frac{n}{2^2}\right) + \sqrt{2 \cdot n/2}\right) + \sqrt{2 \cdot n} \\ &= 2^2 \cdot T\left(\frac{n}{2^2}\right) + \sqrt{n} \cdot (2 + \sqrt{2}) \\ &\leq 2^2 \cdot \left(2 \cdot T\left(\frac{n}{2^3}\right) + \sqrt{2 \cdot n/2^2}\right) + \sqrt{n} \cdot (\sqrt{2^2} + \sqrt{2}) \\ &= 2^3 \cdot T\left(\frac{n}{2^3}\right) + \sqrt{n} \cdot (\sqrt{2^3} + \sqrt{2^2} + \sqrt{2}) \end{aligned}$$

from here we can make the educated guess, that after i steps, the recursion look as follows:

$$T(n) \leq 2^i \cdot T\left(\frac{n}{2^i}\right) + \sqrt{n} \cdot \sum_{j=1}^i \sqrt{2^j}$$

Furthermore, the recursion stops when $i = \log_2(n)$. So we will plug this in:

$$\begin{aligned} T(n) &\leq 2^{\log_2(n)} \cdot T\left(\frac{n}{2^{\log_2(n)}}\right) + \sqrt{n} \cdot \sum_{j=1}^{\log_2(n)} \sqrt{2^j} \\ &= n \cdot T(1) + \sqrt{2n} \left(\sum_{j=0}^{\log_2(n)-1} \sqrt{2^j} \right) = n + \sqrt{2n} \left(\frac{\sqrt{2}^{\log_2(n)} - 1}{\sqrt{2} - 1} \right) \\ &= n + \sqrt{2n} \left(\frac{\sqrt{n} - 1}{\sqrt{2} - 1} \right) = n + \frac{\sqrt{2}}{\sqrt{2} - 1} \cdot (n - \sqrt{n}) \end{aligned}$$

Thus, our guess is

$$f(n) := n + \frac{\sqrt{2}}{\sqrt{2} - 1} \cdot (n - \sqrt{n})$$

Note that this is just an unproven upper bound at this moment. The actual proof this upper bound works for all n is part of the next task.

b) Here we use induction to prove our guess achieved by recursion.

Induction base: $T(1) = 1 \leq 1 + \frac{\sqrt{2}}{\sqrt{2}-1} \cdot 0 = f(1) \checkmark$

Induction hypothesis: We assume that the statement holds for $n/2$.

Induction step: We will now show that based on our hypothesis, the statement also holds for n .

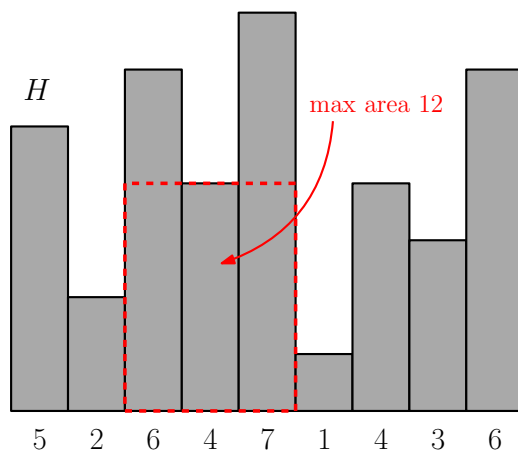
$$\begin{aligned} T(n) &\leq 2 \cdot T\left(\frac{n}{2}\right) + \sqrt{2 \cdot n} \\ &\leq 2 \cdot \left(\frac{n}{2} + \frac{\sqrt{2}}{\sqrt{2}-1} \cdot \left(\frac{n}{2} - \sqrt{\frac{n}{2}} \right) \right) + \sqrt{2 \cdot n} \\ &= n + \frac{\sqrt{2}}{\sqrt{2}-1} \cdot n - \frac{\sqrt{2}}{\sqrt{2}-1} \cdot \sqrt{2n} + \sqrt{2n} \\ &= n + \frac{\sqrt{2}}{\sqrt{2}-1} \cdot n - \sqrt{n} \cdot \left(\frac{2}{\sqrt{2}-1} - \frac{\sqrt{2}(\sqrt{2}-1)}{\sqrt{2}-1} \right) \\ &= n + \frac{\sqrt{2}}{\sqrt{2}-1} \cdot n - \sqrt{n} \cdot \left(\frac{\sqrt{2}}{\sqrt{2}-1} \right) \\ &= f(n) \checkmark \end{aligned}$$

c) Our result is $T(n) = \mathcal{O}(n)$. The case 1 in the Master Theorem leads to the same statement as $\sqrt{n} = n^{1/2}$ we have $c = 1/2 \leq 1 = \log_2(2)$. Hence, due to the theorem, the runtime is $T(n) = \mathcal{O}(n^{\log_2(2)}) = \mathcal{O}(n)$.

Exercise 3: Maximum Rectangle in a Histogram

(10 Points)

Consider a sequence h_1, \dots, h_n of positive, integer numbers. This sequence represents a histogram H consisting of n horizontally aligned bars each of width 1, where h_i represents the height of the i^{th} bar. The goal is to find a rectangle of maximum area completely within H (i.e., within the union of subsequent bars).



(a) Describe an algorithm that computes a maximum area rectangle in H in time $\mathcal{O}(n^2)$. (2 Points)

(b) Describe an algorithm that computes a maximum area rectangle that is within H and also intersects the i^{th} bar in time $\mathcal{O}(n)$ and prove the running time. Also argue why your algorithm is correct, i.e., why there can not exist a rectangle intersecting i with a strictly larger area than the outcome of your algorithm. (5 Points)

Remark: Correct solutions in $o(n^2)$ grant partial points.

(c) Give an algorithm that uses the *divide and conquer* principle to compute a maximum area rectangle in H in time $\mathcal{O}(n \log n)$ and prove the running time. (3 Points)

Remark: You can use part (b), even if you did not succeed.

Sample Solution

(a) There are multiple possible solutions. One idea is to compute for any pair i, j the largest rectangle that starts at the i^{th} bar and ends with the j^{th} bar, respectively.

Algorithm 1 $\text{max-area}(h_1, \dots, h_n)$

```
max-area  $\leftarrow$  0
for  $i := 1$  to  $n$  do
  min-height  $\leftarrow$   $h_i$ 
  for  $j := i$  to  $n$  do
    min-height  $\leftarrow$   $\min(\text{min-height}, h_j)$ 
    max-area  $\leftarrow$   $\max(\text{max-area}, \text{min-height} \cdot (j - i + 1))$ 
return max-area
```

As the computations within both for-loops is constant, the overall runtime is $\mathcal{O}(n^2)$.

(b) The idea is to start at bar i and extend the rectangle in the direction with the higher bar.

Algorithm 2 max-area-bar(i, h_1, \dots, h_n)

```
max-area  $\leftarrow$  0
min-height  $\leftarrow$   $h_i$ 
 $\ell, r \leftarrow i$ 
 $h_0, h_{n+1} \leftarrow -\infty$  ▷ Sentinel elements
while  $\ell > 1$  or  $r < n$  do
  if  $h_{\ell-1} \geq h_{r+1}$  then
     $\ell \leftarrow \ell - 1$ 
    min-height  $\leftarrow$  min(min-height,  $h_\ell$ )
  else
     $r \leftarrow r + 1$ 
    min-height  $\leftarrow$  min(min-height,  $h_r$ )
  max-area  $\leftarrow$  max(max-area, min-height  $\cdot$  ( $r - \ell + 1$ ))
return max-area
```

Running time: The operations inside the while loop have constant running time. The while loop has at most n iterations since in each iteration we either increase r or decrease ℓ and both variables together can be increased or decrease at most n times, until they hit 1 or n . Therefore, the running time is $\mathcal{O}(n)$.

Correctness: For completeness's sake we give an argument why the algorithm produces the correct result, i.e., the maximum rectangle in H that intersects bar i . This is not so clear (as in part (a)), since we are not checking every possibility, i.e., the area of every rectangle extending from some bar $\ell' \leq i$ to some bar $r' \geq i$, so we can not simply argue that we get the correct result by "brute force".

For the sake of contradiction assume that our algorithm computes a rectangle R_{alg} , but there is a rectangle R_{opt} that has the largest area among all rectangles intersecting i . Assume the leftmost column, of R_{opt} is ℓ_{opt} and the rightmost is r_{opt} . Since our algorithm iteratively increases the current rectangle, there is an iteration where we add ℓ_{opt} and there is an iteration where we add r_{opt} . Without loss of generalization assume that our algorithm added ℓ_{opt} before r_{opt} (the other case is symmetric). We will now argue that in the next iterations after adding ℓ_{opt} our rectangle will extend itself towards r_{opt} before extending further to the left. The argument is simple, if our algorithm would consider the next leftmost bar, it must have a larger height than the right counterpart, we could thus extend R_{opt} by this column and achieve a new rectangle with a larger area than R_{opt} . Contradiction. Hence, our algorithm would not consider any bar from the left before adding r_{opt} . At that point the current rectangle is exactly R_{opt} . Thus, R_{alg} will necessarily have the same area as R_{opt} (since we only update the area if we find a larger one) and thus is optimal as well.

- (c) The idea is to split the Histogram H at the middle bar $m := \lfloor \frac{n}{2} \rfloor$ and take the maximum area of three partial results. The first is the maximum area of a rectangle in h_1, \dots, h_{m-1} the second is the maximum area intersecting bar m and the third is the maximum area rectangle in h_{m+1}, \dots, h_n . As the partial results cover all possible positions of rectangles, the overall result is correct if the partial results are. The first and third result are obtained recursively, the second with our result from part (b).

Algorithm 3 $\text{max-area-dc}(h_1, \dots, h_n)$

```
if  $n = 0$  then
  return 0 ▷ check base cases
if  $n = 1$  then
  return  $h_1$ 
 $m \leftarrow \lfloor \frac{n}{2} \rfloor$ 
area1  $\leftarrow \text{max-area-dc}(h_1, \dots, h_{m-1})$ 
area2  $\leftarrow \text{max-area-bar}(m, h_1, \dots, h_n)$ 
area3  $\leftarrow \text{max-area-dc}(h_{m+1}, \dots, h_n)$ 
return  $\max(\text{area1}, \text{area2}, \text{area3})$ 
```

Running time: Our recursive function for the running time is $T(n) \leq 2T(\frac{n}{2}) + \mathcal{O}(n)$. Applying the Master Theorem given in the lecture shows that $T(n) \in \mathcal{O}(n \log n)$.