



## Algorithm Theory

### Sample Solution Exercise Sheet 12

Due: Friday, 30th of January, 2026, 10:00 am

#### Exercise 1: $k$ -th smallest element

(10 Points)

Given a set  $S$  of  $n$  pairwise distinct numbers and a number  $k \in \{1, \dots, n\}$ , we want to define a function that returns the  $k^{\text{th}}$  smallest element of  $S$ .

Consider the following randomized divide and conquer algorithm.

---

**Algorithm 1** `select`( $S, k$ )

---

```
1: Choose a pivot  $x \in S$  uniformly at random
2:  $S^- = \emptyset$ 
3:  $S^+ = \emptyset$ 
4: for all  $y \in S \setminus \{x\}$  do
5:   if  $y < x$  then
6:      $S^- = S^- \cup \{y\}$ 
7:   else
8:      $S^+ = S^+ \cup \{y\}$ 
9: if  $|S^-| = k - 1$  then
10:  return  $x$ 
11: else
12:  if  $|S^-| \geq k$  then
13:    return select( $S^-, k$ )
14:  else
15:    if  $|S^-| < k - 1$  then
16:      return select( $S^+, k - |S^-| - 1$ )
```

---

- (a) Shortly explain why `select`( $S, k$ ) is correct, i.e., returns the  $k^{\text{th}}$  smallest element of  $S$ , and explain the worst-case runtime. (3 Points)
- (b) What is the probability that both  $|S^+| \leq \frac{3}{4}|S|$  and  $|S^-| \leq \frac{3}{4}|S|$  (after one iteration)? Explain your answer. For simplicity, you may assume that  $|S|$  is a multiple of 4. (1 Point)
- (c) Give an upper bound on the expected runtime of `select`( $S, k$ ). Explain your answer. (3 Points)  
*Hint: Use (b).*
- (d) Show that `select`( $S, k$ ) terminates in time  $O(n \log n)$  with probability at least  $1 - \frac{1}{n}$ . (3 Points)  
*Hint: Use the following Chernoff's Bound: If  $X_1, \dots, X_n$  is a sequence of independent 0-1 random variables,  $X = \sum X_i$  and  $\mu = E[X]$ , then for any  $0 < \delta < 1$  we have*

$$\Pr(X \leq (1 - \delta)\mu) \leq e^{-\frac{\delta^2}{2}\mu}.$$

## Sample Solution

- (a) **Correctness:** The algorithm picks a random pivot and constructs a set of elements smaller than the pivot and a set of elements larger than the pivot. If the pivot is the  $k^{\text{th}}$  smallest element, then the set of smaller elements is of size exactly  $k - 1$ . Thus, the algorithm is correct in this case. If otherwise, the  $k^{\text{th}}$  smallest element is smaller than the pivot, then the set of smaller elements is of size at least  $k$ . And hence finding the  $k^{\text{th}}$  smallest element is the same as finding it in this subset. Thus, in this case the recursion in line 13 solves the problem correctly. The last case is if the pivot is larger than the  $k^{\text{th}}$  smallest element. This implies that the  $k^{\text{th}}$  smallest must be in the set of elements that are larger than the pivot. If we assume that the set of smaller elements plus the pivot contain  $\ell := |S^-| + 1 < k$  elements, then the smallest element in  $S^+$  is the  $(\ell + 1)^{\text{th}}$  smallest element in  $S$  and the second smallest is the  $(\ell + 2)^{\text{th}}$  smallest in  $S$  and so on. Hence, the  $k$  smallest in  $S$  is the  $k - \ell$  smallest in  $S^+$ . The recursive call in line 16, is computing the  $k - |S^-| - 1 = k - \ell$  smallest in  $S^+$ .

**Worst-Case:** The worst case (similar to quicksort) is that the pivot is always the smallest (or largest) element of the array. Note that this means that in every recursive call the array shrinks by just one element. Thus, the worst case runtime is  $T(n) = T(n - 1) + O(n) \leq O(n^2)$ .

- (b) The statement translates to we do not want to pick a pivot that belongs to the smallest  $1/4$  nor to the largest  $1/4$  of the elements, i.e., we want to pick an element from the 'middle half'. Since this half contains half of the elements of  $S$ , we get such a good pivot with probability  $1/2$ .
- (c) With probability  $1/2$  we get a good pivot (i.e., the recursive call will be on a sublist of size  $\leq 3/4|S|$ ) and with probability  $1/2$  we get a bad pivot that barely shrinks the sublists in the recursive call:

$$E[T(n)] = \frac{1}{2} \cdot E\left[T\left(\frac{3n}{4}\right)\right] + \frac{1}{2} \cdot E[T(n)] + c \cdot n$$

This can be simplified to

$$E[T(n)] = E\left[T\left(\frac{3n}{4}\right)\right] + c' \cdot n$$

By the master theorem we get that  $E[T(n)] = O(n)$ . Alternatively this can also be computed as follows:

$$E[T(n)] \leq E\left[T\left(\frac{3n}{4}\right)\right] + c' \cdot n \leq E\left[T\left(\frac{3^2 n}{4^2}\right)\right] + c' \cdot \frac{3n}{4} + c' \cdot n \leq n \cdot c' \cdot \sum_{i=0}^{\lceil \log_{4/3}(n) \rceil} \left(\frac{3}{4}\right)^i \leq 4 \cdot c' \cdot n$$

- (d) We want to show that the algorithm succeeds in  $N := O(\log n)$  iterations with prob  $1 - 1/n$  (since each iteration takes time at most  $O(n)$ , this is sufficient). We define  $X_i$  as the event that in iteration  $i$  we have a good split (that by (b) hold with probability  $1/2$ ). Furthermore, we know that after  $\lceil \log_{4/3}(n) \rceil$  good splits we are done. We say  $X = \sum_i X_i$  and hence  $\mu = E[X] = N/2$ . We will now show that the statement holds if we choose  $N = 16 \cdot \ln(n) \geq 4 \cdot \lceil \log_{4/3}(n) \rceil$ . To apply Chernoff, we will choose  $\delta = 1/2$  as this leads to the following equality  $(1 - \delta) \cdot \mu = 1/2 \cdot N/2 = N/4$ . Hence:

$$\begin{aligned} P(X \leq \lceil \log_{4/3}(n) \rceil) &= P\left(X \leq \frac{4 \lceil \log_{4/3}(n) \rceil}{4}\right) \leq P\left(X \leq \frac{16 \ln(n)}{4}\right) = P(X \leq N/4) \\ &= P(X \leq (1 - \delta)\mu) \\ &\leq e^{-\frac{(1/2)^2}{2} \cdot \frac{N}{2}} = e^{-\frac{N}{16}} \\ &= \frac{1}{e^{\ln(n)}} = \frac{1}{n}. \end{aligned}$$

## Exercise 2: Randomized Coloring

(10 Points)

Let  $G = (V, E)$  be a simple, undirected graph with maximum degree  $\Delta$ . A (node) coloring of the graph is an assignment of colors to the nodes in a way that no two adjacent nodes are assigned with the same color. More formal: A coloring is a mapping  $\phi : V \rightarrow C$  of nodes in  $V$  to some color space  $C$  s.t.  $\phi(u) \neq \phi(v)$  if  $\{u, v\} \in E$ .

Consider Algorithm 2 to assign colors from the colors space  $\{1, 2, \dots, \Delta + 1\}$  to the nodes. Let  $L_v$  be the lists of **available** colors of  $v$ , that initially is set to the color space.

---

### Algorithm 2 Randomized Coloring

---

**Ensure:**  $\phi$  is a proper  $\Delta + 1$  coloring

- 1: Let  $L_v := \{1, 2, \dots, \Delta + 1\}$
  - 2: **for** each uncolored node  $v \in V$  in parallel **do**
  - 3:      $v$  becomes active with probability  $p = \frac{1}{2}$
  - 4:     **if**  $v$  is active **then**
  - 5:         Let  $v$  choose a color  $x_v \in L_v$  uniformly at random
  - 6:         **if** no neighbor  $u$  picked  $x_v$  as well **then**
  - 7:              $\phi(v) := x_v$  ▷  $v$  is colored now!
  - 8:     **if**  $v$  is still uncolored **then**
  - 9:         delete  $\phi(u)$  from  $L_v$  for all colored neighbors  $u$ . ▷ Update  $L_v$
- 

Note that in every iteration,  $|L_v|$  is larger than the number of uncolored neighbors of  $v$ .

- (a) Show that a node  $v$  that is still uncolored will be colored in the next iteration with probability at least  $1/4$ . (5 Points)  
*Hint: Assume  $v$  is active and has  $k$  uncolored neighbors. What is the probability that  $v$  gets colored?*
- (b) After how many iterations is a node  $v \in V$  colored in expectation? (2 Points)
- (c) Show that Algorithm 2 terminates in  $O(\log n)$  iterations **with high probability**.  
That is for a given constant  $c > 0$ , all nodes are colored within  $O(\log n)$  iterations with probability at least  $1 - \frac{1}{n^c}$ . (3 Points)  
*Hint: Use the result of (a) for tasks (b) and (c) even if you didn't manage to come up with a solution.*

## Sample Solution

- (a) Fix an uncolored node  $v$  and assume  $v$  is active and has  $k$  uncolored neighbors. The probability that some neighbor  $u$  is active is  $p$ . Let  $u$  decides on a color  $x_u$ , the probability that  $v$  decides on the same color  $x_u = x_v$  is  $1/|L_v|$  (since a color from  $L_v$  is chosen uniformly at random). Note that by construction  $|L_v| \geq k + 1$ . It follows that  $v$  and  $u$  are in conflict with probability  $\frac{p}{|L_v|} \leq p/(k + 1)$ . Hence, the probability that  $v$  is in conflict with at least one of the  $k$  neighbors is by the **union bound** at most  $p \cdot \frac{k}{k+1} \leq p$ . Thus,  $v$  is not in conflict with any neighbor with probability at least  $(1 - p)$ .  
Since any node  $v$  will be colored in the current iteration if it is active and not in conflict with any neighbor, this happens with probability at least  $p \cdot (1 - p) = 1/4$ .
- (b) Let  $X_v$  be the random variable indicating the iteration  $1 \leq i$  when  $v$  successfully decides on a color. Let  $Z$  be the geometrical distribution with success probability  $1/4$ . By task (a) we know that the success probability for a node  $v$  to get colored is  $\geq 1/4$ , hence, we have  $E[X_v] \leq E[Z]$ . As the expected value of any geometrical distribution is the reciprocal of the success probability we have  $E[X_v] \leq E[Z] = 4$ .

(c) By (a), we have that  $v$  is not colored within the first  $i$  iterations with probability at most  $(3/4)^i$ . Hence, by the union bound there exist at least one uncolored node after  $i$  iterations with probability  $n \cdot (3/4)^i$ . Contrarily, the algorithm terminates in  $i$  iterations with probability at least  $1 - n \cdot (3/4)^i$ . Choosing  $i = (c + 1) \log_{4/3}(n)$ , we get the desired high probability bound through the following arithmetic:

$$P(\text{Termination after } (c + 1) \log_{4/3}(n) \text{ iterations}) \geq 1 - n \cdot (3/4)^{\log_{4/3}(n^{c+1})} = 1 - \frac{n}{n^{c+1}} = 1 - 1/n^c$$