

CS3230 Semester 1 2025/2026
Design and Analysis of Algorithms

Tutorial 11
Order Statistics
For Week 12

Document is last modified on: November 4, 2025

1 Timing Remarks

For most semesters (especially Semester 2, due to Chinese New Year), we will not be able to reach Tutorial 11. For some other semesters, we could also only end up with Tutorial 09 (lost two slots) if the Public Holidays of that semester coincide with our tutorial slot and NUS well-being day. Recently, CS3230 has started tutorials from Week 02 to compensate for these, and since there is no public holiday on Wed-Thu-Fri this semester, we are fortunate to have all $11 + 1 = 12$ sessions.

The last tutorial 12 next week will be an optional revision tutorial where we will go through 5 LeetCode tasks that cover as many CS3230 topics as possible.

2 Lecture Review: Linear-Time Sorting and Selection

Back in lec04a, using decision tree model, we can show that any comparison-based sorting algorithm cannot sort n comparable elements (e.g., into non-decreasing order) faster than $\Omega(n \log n)$ time.

However, if we design sorting algorithms that do not compare elements, we may be able to break this lower bound. In class, we learn two such sorting algorithms:

1. Counting sort (simple version and the stable sort version)
Terms and conditions: Only applicable to sort integers that fall in a ‘small range’ of $[0..k - 1]$.
2. Radix sort (a.k.a. iterated Counting sort from Least to Most Significant Digit)
Terms and conditions: As it requires stable Counting sort as its sub-routine, then the same terms and conditions governing Counting sort also applies. If we break b -bit integers into $\frac{b}{r}$ groups of r -bits integers, then $k = 2^r$ of Counting sort needs to be of ‘small’.

We also learn the Selection (Order Statistics) problem: Given an unsorted array, find the i -th smallest element in that array. We assume that we can only find the order of elements by comparing, i.e., we cannot use Counting and/or Radix sort. In class, we learn several ways to do this:

1. Very slow $O(i \cdot n)$ ‘complete search’ version, worst when $i = \frac{n}{2}$, i.e., the median.
2. Reducing the Selection problem into Sorting problem, thus we can run $\Theta(n \log n)$ comparison-based sorting on the unsorted array, and then report index i of the sorted array. However, since Selection is ‘easier’ than Sorting, we are not bounded by the same $\Omega(n \log n)$ lower bound of comparison-based sorting, see the next two algorithms below.
3. Run the ‘median of medians’ selection algorithm that spend some computation time just to find a ‘good pivot’ by dividing n elements into $\lfloor \frac{n}{5} \rfloor$ groups of 5 elements, find the median of those small group of 5 using ‘brute force’, then find the median x of these medians. By partitioning the array around x , we can design a Divide and Conquer algorithm that runs in worst-case linear-time. But, this algorithm is not practical in real-life.
4. We can partition the array around a random pivot. In this tutorial, we will analyze that such randomized algorithm, called **Quickselect**, invented by the same Tony Hoare who invented **Quicksort**, runs in expected linear-time and is much practical.

3 Tutorial 11 Questions

Q1). Given a set of $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ of points on Euclidean plane, what is the best running time for finding all the \sqrt{n} points closest (using the normal Euclidean distance) to the origin, i.e. $(0, 0)$? Give a short explanation.

1. $\Theta(\sqrt{n})$
2. $\Theta(n)$
3. $\Theta(n \log n)$
4. $\Theta(n\sqrt{n})$

Q2). It is possible to modify non-randomized Quicksort to run in worst-case $\Theta(n \log n)$ time by changing the pivot selection method.

1. No way...
2. It is possible

Q3). Please refer to the pseudocode of the randomized **Quickselect** algorithm shown in lec11. This time, we will analyze its expected time complexity.

Part 1: Suppose in every recursive call of **Quickselect**, the size of subarray is reduced from n to a size of at most $\frac{9n}{10}$.

Part 2: Suppose in every recursive call of **Quickselect**, the size of subarray is reduced from n to a size of at most $n - 1$.

What is the running time of **Quickselect**?

1. $O(n)$ for Part 1 and $O(n)$ for Part 2.
2. $O(n^2)$ for Part 1 and $O(n)$ for Part 2.
3. $O(n)$ for Part 1 and $O(n^2)$ for Part 2.
4. $\Omega(n^2)$ for Part 1 and $\Omega(n^2)$ for Part 2.

Q4). Complete the analysis of expected time complexity of **Quickselect** below.

Let $T(n)$ be the random variable for the running time of **Quickselect** on an input of size n .

For $k \in [0, 1, \dots, n-1]$, we define the indicator random variable X_k , where:

$$X_k = \begin{cases} 1, & \text{if the Partition subroutine generates a } k : (n - k - 1) \text{ split} \\ 0, & \text{otherwise} \end{cases}$$

Q4a). What is the value of $E[X_k]$?

To obtain an upper bound, assume that the i -th element always falls in the larger side of the partition.

When $X_k = 1$, i.e., **Partition** subroutine generates a $k : (n - k - 1)$ split,

$T(n) = T(\max(k, n - k - 1)) + \Theta(n)$. And as $k \in [0, 1, \dots, n - 1]$, then:

$$T(n) = \begin{cases} T(\max(0, n - 1)) + \Theta(n), & \text{for } X_0 = 1 \\ T(\max(1, n - 2)) + \Theta(n), & \text{for } X_1 = 1 \\ \dots \\ T(\max(n - 1, 0)) + \Theta(n), & \text{for } X_{n-1} = 1 \end{cases}$$

Q4b). Complete the following computations:

1. Simplify the $T(n)$ formula above using the summation \sum symbol:
2. Then, the expectation $E[T(n)]$ is:
3. Use linearity of expectation to move the \sum outside of E :
4. Use the fact that each X_k is independent from other random choices to separate $E[X_k]$:
5. Use the value of $E[X_k]$ from Q4a). and simplify:
6. Use linearity of expectation to further simplify:
7. Realize that the terms $k \in [0, \frac{n}{2}]$ and $k \in [\frac{n}{2}, n - 1]$ are similar to simplify $T(\max(k, n - k - 1))$:

Q4c). If you do the steps of Q4b). correctly, you will end up with $E[T(n)] \leq \frac{2}{n} \sum_{k=\lfloor \frac{n}{2} \rfloor}^{n-1} E[(T(k)) + \Theta(n)]$.

Using substitution method, prove that $E[T(n)] \leq c \cdot n$ for some constant $c > 0$.

The constant c can be chosen to be large enough so that $E[T(n)] \leq c \cdot n$ for the base cases.

Hint: use this fact that $\sum_{k=\lfloor \frac{n}{2} \rfloor}^{n-1} k \leq \frac{3n^2}{8}$ (you do not need to re-prove this).

In summary, **Quickselect** works fast, i.e., in expected linear-time. It is also an excellent algorithm in practice, despite having a very bad worst-case of $\Theta(n^2)$ which can happen as shown on LeetCode demo during Tue, 04 Nov 25 lecture.

~~Optional Q5) - time permitting~~ (most likely there will be time, as there are only 4 questions this time. Discuss the following LeetCode task during tutorial (curated variations of order statistics problems): TA can choose to discuss in high-level only or show the partial/full code (in C++/Python/Java)

- Wednesday class: query-kth-smallest-trimmed-number
- Thursday class: kth-smallest-element-in-a-bst
- Friday class: kth-largest-element-in-a-stream

4 Postlude

As this is the (second) last tutorial of CS3230 S1 AY 25/26 and not everyone will attend the optional last Tutorial 12 (although students are strongly recommended to do so for final assessment preparation), some TAs may want to take class photo today as momento (otherwise, TAs can also do this on the last Tutorial 12 next week).

All the best for your final assessment on Tue, 25 Nov 25, 9.00-11.30am.