CS3230 Tutorial 11

Deng Tianle (T15)

6 November 2025

Q1

Given a set $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ of n points on $\mathbb{R}^2$, what is the best running time for finding the $\sqrt{n}$ points closet to the origin?

Notice that this is $\Omega(n)$ since we need to look at all n points at least once.

Q1

Given a set $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ of n points on $\mathbb{R}^2$, what is the best running time for finding the $\sqrt{n}$ points closet to the origin?

Notice that this is $\Omega(n)$ since we need to look at all n points at least once.

Answer: $\Theta(n)$ using median of medians (but large constant factor):

1. Compute all n distances from origin in $\Theta(n)$
2. Select the $i = \sqrt{n}$-th closet point to origin in $\Theta(n)$ (this uses the median of medians algorithm introduced in lecture)
3. Partition around this $\sqrt{n}$-th element and report first $\sqrt{n}$ points in $\Theta(\sqrt{n})$.

If you do not know/want median of medians, then just sort the distances in $\Theta(n \log n)$.

Q2

Is it possible to modify non-randomised Quicksort to run in worst-case $\Theta(n \log n)$ time by changing the pivot selection method?

Yes. We can use median of medians to select the pivot to be the median itself in $\Theta(n)$ times. This just adds to the running time to partition, also in $\Theta(n)$. This gives the time complexity

$$T(n) = 2T(n/2) + kn,$$

hence $T(n) \in \Theta(n \log n)$.

However, this version is much slower in practice than randomised Quicksort due to large constant factor.

Q3

Similar considerations as in Quicksort.

Suppose in every recursive call of Quickselect, size of subarray reduced from n to at most $9n/10$. This is good because

$$T(n) = T(9n/10) + kn$$

gives $T(n) = \Theta(n)$ e.g. by master theorem (check regularity condition).

Suppose in every recursive call of Quickselect, size of subarray reduced from n to $n-1$. In this case,

$$T(n) = T(n-1) + kn$$

gives $T(n) \in \Theta(n^2)$ similar to sum of arithmetic progression.

Q3

Actually, this suggests a way of seeing the expected time. For example, we partition the list into the middle half (i.e. lower to upper quartile) and otherwise. Then on expectation, our random pivot fall in the middle half after two random choices (by geometric distribution, expectation is $1/0.5 = 2$). Then $T(n) \in \Theta(n)$.

Q4

Formal analysis of expected time complexity of Quickselect. Let $T(n)$ be the running time of Quickselect on an input of size n. For $k = 0, 1, \ldots, n-1$, we define the indicator random variable X_k where

$$X_k = \begin{cases} 1, & \text{if Partitaion routine generates a } k \text{ to } n-k-1 \text{ split} \\ 0, & \text{otherwise} \end{cases}$$

Note that $E[X_k] = \frac{1}{n}$.

Now refer to the recurrrence in the tutorial sheet. We can simplify it to:

$$T(n) = \sum_{k=0}^{n-1} X_k (\, T(\max(k, n-k-1)) + \Theta(n)).$$

Then taking expectation and using linearity,

$$E[T(n)] = \sum_{k=0}^{n-1} E[X_k (\, T(\max(k, n-k-1)) + \Theta(n))].$$

Q4

Fix k, X_k is independent from $T(\max(k, n-k-1))$. This is because X_k only depends on the first pivot choice while $T(\max(k, n-k-1))$ only depends on later pivot choices. Hence

$$E[T(n)] = \sum_{k=0}^{n-1} E[X_k] E[(T(\max(k, n-k-1)) + \Theta(n))].$$

By linearity and substitution:

$$E[T(n)] = \frac{1}{n} \sum_{k=0}^{n-1} E[T(\max(k, n-k-1))] + \frac{1}{n} \sum_{k=0}^{n-1} \Theta(n)$$

By symmetry,

$$E[T(n)] \leqslant \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[T(k)] + \Theta(n).$$

Q4

Use substitution method to show that $E[T(n)] \leqslant cn$. We use the fact that $\sum_{k=\lfloor n/2 \rfloor}^{n-1} k \leqslant 3n^2/8$, compute

$$E[T(n)] \leqslant \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} ck + dn$$

$$E[T(n)] \leqslant \frac{2c}{n} \left(\frac{3n^2}{8} \right) + dn$$

$$E[T(n)] \leqslant cn - \left(\frac{cn}{4} - dn \right)$$

so the proof goes through by choosing $c > 4d$.

LeetCode

Find k-th largest element in a stream.

Use a (min) priority queue. Cap size to be at most k. If adding the next element will make the size go above k, we compare the root (which is the k-th largest currently) with the next element. Keep the larger of the two. From now on, output the root after each addition.