Practice Quiz 1

- The following practice quiz is a compilation of relevant problems from previous semesters.
- This practice quiz may not be taken as a strict gauge for the difficulty level of the actual quiz.
- The quiz contains multiple problems, several with multiple parts, for a total of 120 points, which should take 120 minutes.
- **This practice quiz is not meant to exactly reflect the difficulty level of the actual quiz, but rather provide a collection of representative problems.**
- Write your solutions in the space provided. If you run out of space, continue on a scratch page and make a notation.
- Do not waste time deriving facts that we have studied. Just cite results from class.
- When we ask you to *give an algorithm* in this quiz, describe your algorithm in English or pseudocode, and provide a short argument for correctness and running time. You do not need to provide a diagram or example unless it helps make your explanation clearer.
- Do not spend too much time on any one problem.
- **Good Luck!**

Problem-1: [10 points] **True or False**

Please circle **T** or **F** for the following. *No justification is needed (nor will be considered).*

(a) [2 points] **T** **F**
In amortized analysis, we require that the real cost of executing any given operation is always upper bounded by the amortized cost of that operation.

Solution: False. The goal of amortized analysis is exactly to be able to argue that the amortized cost of a given operation is small, even if the real cost of executing that operation can be sometimes large.

(b) [2 points] **T** **F**
Given any two polynomials P and Q, and *any* two real numbers a and b, the discrete Fourier transform of the polynomial $a \cdot P + b \cdot Q$ is equal to $a \cdot P^* + b \cdot Q^*$, where P^* and Q^* are the discrete Fourier transforms of the polynomials P and Q, respectively.

Solution: True. The operation of taking a discrete Fourier transform is linear.

(c) [2 points] **T** **F**
Given an *unsorted* array of n distinct integers, we can output the $n/4$ smallest elements in $O(n)$ time.

Solution: True. Suffices to use the $O(n)$ rank/median finding algorithm from the class to find the rank-$n/4$ element. Then, we can scan the array in $O(n)$ time and delete from it all the entries that are smaller or equal to that element.

(d) [2 points] **T** **F**
Consider a family of hash functions $\mathcal{H} = \{h_1, h_2\}$ that maps a universe $\{a, b, c\}$ to $\{0, 1\}$, where $h_1(a) = h_1(b) = 0$, $h_1(c) = 1$ and $h_2(a) = h_2(b) = 1$, $h_2(c) = 0$. This family $\mathcal{H}$ gives rise to a universal hash function family.

Solution: False. For example, elements a and b collide in both h_1 and h_2.

(e) [2 points] **T** **F**
A UNION-FIND data structure of n elements is initially arranged as a binary tree. If the data structure is implemented with path compression, it can be reduced to a height-one tree with $O(\log n)$ calls to FIND-SET. (A height-one tree has all elements directly connected to the root.)

Solution: False. The tree has $\lg n$ levels and $n/2$ leaves. Transforming to a height-one tree requires that every element is directly connected to the root. This requires $O(n)$ FIND-SET calls.

Problem-2: [20 points] **Crossword Contest**

Every day you and your roommate stage a contest to see who can complete the New York Times crossword puzzle faster. The loser pays the winner one dollar. You begin the contest on day 1, and you and your roommate are equally matched, so the probability p that you win is 0.5 for each day.

(a) [10 points] You are happy with this arrangement so long as you don't risk too much money. Show that, for any $t \geq 1$, the probability that the net amount of dollars you lose after t days will be larger than $2\sqrt{t}$ is at most $\frac{1}{8}$.

Solution: Let X_i be the outcome of the ith game, defined as:

$$X_i = \begin{cases} 1 & \text{if win on day } i, \text{ w.p. } 1/2 \\ -1 & \text{if lose on day } i, \text{ w.p. } 1/2 \end{cases}$$

and let $X = \sum_i X_i$ be the total payout. We are interested in $\Pr[X \leq -2\sqrt{t}]$. Note that $\mu(X_i) = 0$ so, by linearity of expectation, $\mu(X) = 0$. Also, note that $\sigma^2(X_i) = E[X_i^2] - \mu(X_i)^2 = 1$ so, by linearity of variance under the independence assumption, $\sigma^2(X) = t$. Using Chebyshev's inequality,

$$\Pr[|X| \geq 2\sqrt{t}] \leq \frac{\sigma^2(X)}{4t} = \frac{1}{4}$$

The result follows as the random variable X is symmetric around 0 so

$$\Pr[X \leq -2\sqrt{t}] = \frac{1}{2}\Pr[|X| \geq 2\sqrt{t}] \leq \frac{1}{8}$$

(b) [10 points] You take the summer off to study for your GREs, and when you restart in the Fall, you are much better than your roommate, with a probability p that you win of 0.75 each day.

You are now looking forward to winning some money. Let W_t be a lower bound on the net amount of money that you will win with probability of at least $1 - e^{-1}$. What is the best (lower bound) estimate on W_t that you can come up with?

Solution: As in part (a), let X_i be the outcome of the ith game defined as:

$$X_i = \begin{cases} 1 & \text{if win on day } i, \text{ w.p. } 3/4 \\ -1 & \text{if lose on day } i, \text{ w.p. } 1/4 \end{cases}$$

and let $X = \sum_i X_i$ be the total payout. Under the new probability distribution, we have $\mu(X_i) = \frac{1}{2}$ and $\mu(X) = \frac{t}{2}$. We would like to use Chernoff bounds to get the tightest bounds possible, but the X_i are not Bernoulli as required. One solution is to

transform the X_i into $Y_i = (1 + X_i)/2$ and $Y = \sum_i Y_i = (t + X)/2$, such that the Y_i are Bernoulli and so Chernoff can be applied. Note that $\mu(Y_i) = 3/4$ and $\mu(Y) = 3t/4$. We can now use the third version of Chernoff's bound

$$\Pr[Y < (1 - \beta)\mu(Y)] < e^{-\beta^2\mu(Y)/2}$$

We seek a setting for β for which

$$e^{-\beta^2\mu(Y)/2} = e^{-1}$$

Manipulating, we get $\beta^2 3t/8 = 1$, or

$$\beta = \sqrt{\frac{8}{3t}}$$

This gives us

$$\begin{aligned}
\Pr[Y < (1 - \beta)\mu(Y)] &= \Pr\left[Y < \frac{3t}{4} - \sqrt{\frac{8}{3t}}\frac{3t}{4}\right] \\
&= \Pr\left[Y < \frac{3t}{4} - \sqrt{\frac{3}{2}t}\right] \\
&\leq e^{-1}
\end{aligned}$$

Restating the result in terms of the original variable X, using $Y = (t + X)/2$, we get

$$\begin{aligned}
\Pr\left[Y < \frac{3t}{4} - \sqrt{\frac{3}{2}t}\right] &= \Pr\left[\frac{(t + X)}{2} < \frac{3t}{4} - \sqrt{\frac{3}{2}t}\right] \\
&= \Pr\left[X < \frac{t}{2} - \sqrt{6t}\right] \\
&< e^{-1}
\end{aligned}$$

Problem-3: [15 points] **Line World**

Ben has a list of the locations $x_1, x_2, \ldots, x_n$ of the n cities of Line World, which all lie along the positive x-axis, alphabetized by city name. Ben wants to place $p-1$ post offices in cities to provide equal service to the cities of the world. Thus, Ben decides to place post offices so that they partition the cities into p groups, each of size n/p. For the sake of this problem and Ben's sanity, we assume that this is possible without rounding error.

(a) [5 points] Suppose Ben only has the resources to build 3 post offices (ie. $p = 4$). Which cities should Ben choose? Design an $O(n)$ algorithm to select cities.

Solution: One solution is to apply the median finding algorithm to find the median and then apply the algorithm recursively to the cities below and then again to the cities above the median. This will run in $O(n)$ time.

The median finding algorithm can be modified to find an arbitrary rank. An alternative solution is to use the rank finding algorithm to find the cities of rank $\frac{n}{4}$, $\frac{n}{2}$ and $\frac{3n}{4}$.

(b) [10 points] Now suppose Ben can instead build $p-1$ post offices. Design an algorithm to figure out the location of $p-1$ post offices. For full credit, the algorithm should run in $O(n \log p)$ time.

Solution: An $O(np)$ algorithm would simply use the rank finding algorithm to find the cities of rank $\frac{ni}{p}$ for $i = 1, 2, \cdots, p-1$. We can also sort in $O(n \log n)$ time and find the $p-1$ locations. However, we can do better.

Consider modifying the median finding algorithm to recurse on halves until we find the quantiles. Assume that $p = 2^\ell$. In the first level, we look for the median, which is the element of rank $n/2$. In the second level, we look for the median of the two smaller lists, which gives the quantiles $n/4$ and $3n/4$. The key observation is that we only have to recur $\log p = \ell$ times to find the p quantiles. Each level of recursion requires $O(n)$ time in total, providing an $O(n \log p)$ runtime. The recurrence will be $T(n, p) = 2T(n/2, p/2) + O(n)$ or $T(n, p) = O(n \log p)$.

When p is not a power of 2, raise it to p', which is the next power of 2 – this might involve one more recursion. After running the above algorithm, run the rank finding algorithm on each of the p' lists to find the appropriate element. This will take an additional $O(n)$ time total, since the lists are of size $O(n/p')$.

Problem-4: [30 points] **Antenna Repair**

You are one of the first colonizers of Mars! You arrived on Mars as part of Melon Usk's Mars Three expedition. (Sadly, the Mars One and Mars Two were not successful. The third time's the charm!) Your duty today is to maintain an array of antennas (arranged in a line) that enables the colony to communicate with Earth. Due to fairly strong solar radiation, the antennas tend to malfunction often. Each time it happens you must replace the faulty electronics with spare parts you have in your EdisonY Mars Rover. Unfortunately, the rover has only a limited battery life, so you need to use it sparingly and, in particular, sometimes it is better to simply take the spare part with you and walk to the faulty antenna by yourself instead of driving there.

This gives rise to the following online problem. Let $p_0 = 0$ be the initial position (on the line) of the rover. In each round t:

- You receive a faulty antenna request at location r_t. (The value of r_t is a location on the line.)
- Upon receiving the request r_t, you have to decide which position p_t to move your rover to. (Staying in place, i.e., setting $p_t = p_{t-1}$ is a valid choice). The cost to move is $C \cdot |p_t - p_{t-1}|$, i.e., C times the distance the rover moved, for some $C > 1$.
- Then, you need to service the request and this incurs an additional cost of $2|p_t - r_t|$ corresponding to your needing to walk from the rover to the faulty antenna (and then come back).

(a) [10 points] Consider a strategy in which the rover always stays put. That is, in each round t you choose $p_t = p_{t-1}$. Construct a family of inputs (i.e., sequence of antenna faults) that shows that the competitive ratio of this strategy can be arbitrarily large. (Assume there is an antenna at every integer coordinate of the line.)

Solution: Let $r_i = 1$ for $i = 1 \ldots n$. The strategy of always staying put will incur a cost of $2n$. The strategy of driving the rover to 1 and then staying put will incur a cost of C. The competitive ratio of staying put is thus at least $2n/C$ which is unbounded as n becomes large.

Many other families of inputs will produce an arbitrarily large competitive ratio. For example, $r_i = i$ gives an optimal cost of at most Cn (drive the rover to each antenna fault) while the stay put strategy will incur a cost that is $\Omega(n^2)$.

(b) [10 points] Consider next a strategy in which you always move the truck all the way to the faulty antenna. That is, $p_t = r_t$, in each round t. Argue that this strategy has a competitive ratio of *at least* $\frac{C}{2}$. Specifically, design a sequence of requests such that the total cost of the strategy is at least $\frac{C}{2}$ larger than the cost of the optimal solution for that sequence.

Solution: Consider the sequence of requests $r_i = \frac{1}{2}(-1)^i + \frac{1}{2}$ (i.e. $1, 0, 1, 0, \dots$). On a sequence of length n, driving the rover to each location will give a total cost of $C + C + C + C + \cdots = Cn$. Leaving the rover at 0 and walking to each antenna would give a cost of $2 + 0 + 2 + 0 + \cdots = n$, for a competitive ratio of $C \geq \frac{C}{2}$.

(c) [10 points] Argue that the competitive ratio of the strategy from point (b) is $O(C)$.
Hint: Use a potential function $\Phi(p_t, OPT_t) = C \cdot |p_t - OPT_t|$ and triangle inequality. Here, OPT_t is the position of the rover in the optimal strategy after it moves in round t.

Solution: We will show that the competitive ratio is C. We will use the potential function

$$\Phi_i = |p_i - OPT_i|$$

where OPT_i is the position of the optimal offline algorithm at time i. Note that by definition, $\Phi_i \geq 0$ for all i, and $\Phi_0 = 0$ as $p_0 = OPT_0 = 0$. Hence Φ is a valid potential function. With this potential, the amortized cost of the algorithm at time i is

$$\hat{c}_A(i) = C\left(|p_i - p_{i-1}| + |p_i - OPT_i| - |p_{i-1} - OPT_{i-1}|\right)$$

and the cost for the optimal algorithm is

$$c_{OPT}(i) = C\,|OPT_i - OPT_{i-1}| + 2\,|OPT_i - p_i|\,.$$

Note that by the triangle inequality, we have

$$|p_i - p_{i-1}| \leq |p_i - OPT_i| + |OPT_i - OPT_{i-1}| + |OPT_{i-1} - p_{i-1}|$$

which gives

$$\begin{aligned}\hat{c}_A(i) &\leq C\left(2\,|p_i - OPT_i| + |OPT_i - OPT_{i-1}|\right)\\ &\leq C^2\,|OPT_i - OPT_{i-1}| + 2C\,|OPT_i - p_i|\\ &= Cc_{OPT}(i)\end{aligned}$$

and thus the competitive ratio is at most C.

Alternatively, one can argue that the optimal strategy must walk or drive at least as far as the strategy from part (b) does, immediately giving a competitive ratio of at most C.

Consider the strategy that moves the rover exactly as OPT does, but instead of walking to each request, drives to the request as well. That is, at the beginning of round t the rover is at OPT_{t-1}. Then, drive to the request r_t, back to OPT_{t-1}, and finally to OPT_t. This strategy does at least as much driving as the strategy from part (b), as the strategy from part (b) drives to a subsequence of the points that this strategy drives to. (We are effectively using the triangle inequality here.) And clearly our cost is at most C times the cost of the optimal solution, as the only difference is driving to service the request instead of walking.

Problem-5: [10 points] **Connecting Mars**

Melon Usk has set up a colony on Mars that is made up of n stations. The stations are connected by a fiber network that consists of a total of m links, each connecting two stations. Initially, every station can reach every other station through this network.

Melon checks the weather forecast and sees that there is a dust storm approaching! He can see that the dust storm will destroy k of the links in his network (where $k \leq m$). At each time t_i for $1 \leq i \leq k$, where $t_1 < t_2 < \ldots < t_k$, the fiber link between stations u_i and v_i will be destroyed by the dust storm. Once some links are cut, the stations might not all be connected to each other anymore. For each time t_i, Melon wants to calculate the size of the largest subset of stations that can still reach each other through the fiber network at time t_i. (A station is reachable from another station if there exists a path through the network connecting the two stations.)

Design an algorithm to solve this problem in $O(m\alpha(n))$ time. Justify the correctness and runtime of your algorithm.

Solution: Use the Union-Find data structure, augmented by storing the size of each set in the set representative. We work backwards, starting by computing the number of reachable stations at time t_k and working back to time t_1. Initially, make a set for each station, and call $Union(u, v)$ for each link (u, v) that is still intact at time t_k. As we do this, we also keep track of the size of the maximum size set (which is initially 1), and we update it each time a Union operation produces a new set that is larger than our current maximum. For each time t_i from $i = k-1, k-2, \ldots, 1$, we then update the size of the maximum set by calling $Union(u_i, v_i)$, and checking if it is larger than the previous maximum at time t_{i+1}. This algorithm correctly keeps track of the size of the maximum set because the only way the size of the largest set can increase is if two sets are unioned to produce a new largest set.

In this algorithm, we do $O(n + m)$ Union-Find operations. Since the graph was initially connected, $m \geq n - 1$ so $m = O(n)$. Each Union-Find operation takes $O(\alpha(n))$ amortized time, so the overall runtime of this algorithm is $O(m\alpha(n))$.

Problem-6: [20 points] **Nights of the Hash Table**

Melon Usk is organizing a dinner party where n guests are seated around a round table with $N = 2n$ seats, numbered 1 through N clockwise.

The guests arrive one by one, and each guest takes his or her seat before the next guest arrives. To optimize the amount of networking between the guests, Melon has come up with an unusual way of seating them. Namely, to find his or her seat, each guest i first receives a seat number r_i chosen uniformly and independently at random. Then, if seat r_i is not currently occupied, the guest sits there; otherwise, the guest walks clockwise around the table until he or she finds the first unoccupied seat, and sits there.

A *block* is a set of consecutive seats that are all occupied but with the seats before and after it being unoccupied. Let $p_{j,k}$ be the probability that after all the guests have taken their seats there is a block of length k starting at seat j , i.e. that $\{j, \ldots, j + k - 1\}$ is a block in the final configuration.

(a) [4 points] Let $E_{j,k}$ be the event that at least k of the random numbers $r_1, \ldots, r_n$ given to the guests lie in the set $\{j, \ldots, j + k - 1\}$. Argue that $p_{j,k} \leq Pr[E_{j,k}]$.

Solution: Denote $B = \{j, \ldots, j + k - 1\}$. Suppose that B is a block in the final configuration. Then nobody who got an $r_i \notin B$ is sitting in any of the seats B. Indeed, otherwise they would have passed seat $j - 1$ by, which would mean that there was someone sitting there, whereas we know that seat $j - 1$ is empty in the final configuration, and was thus empty during the entire seating process. Thus, all the people sitting in the seats B had their r_i's in B, which means that there were exactly k of the r_i's which were in the set B. So we showed that the even that B is a block implies that the event $E_{j,k}$ happened, hence $p_{j,k} \leq Pr[E_{j,k}]$.

(b) [9 points] Show that $Pr[E_{j,k}] \leq \frac{1}{c^k}$ for some constant $c > 1$ independent of the problem parameters.
Hint: Consider random variables X_i, for each guest i, where $X_i = 1$ if $r_i \in \{j, \ldots, j + k - 1\}$ and $X_i = 0$, otherwise; and recall that $e > 1$.

Solution: Defining random variables as in the hint, we see that $\mathbb{E}[X_i] = \frac{k}{N}$ and thus letting $X = X_1 + \ldots + X_n$ we have $\mathbb{E}[X] = \frac{nk}{N} = \frac{k}{2}$ by linearity of expectation. We thus want to bound the probability that

$$Pr[X \geq k] = Pr[X \geq (1 + 1)\mathbb{E}[X]]$$

We will use (the multiplicative form of) the Chernoff bound for that. Note that since $\beta = 1$ here, either the form for $\beta \leq 1$ or $\beta \geq 1$ works here. With this, we get

$$Pr[X \geq k] \leq e^{-k/6} = \frac{1}{(e^{1/6})^k}$$

so we can use $c = e^{1/6} > 1$.

(c) [7 points] After all the guests have taken their seats, Melon finally joins them and heads for his favorite seat (which of course is number 1). However, it is quite possible that by that time there is already somebody sitting there because nobody saved Melon's favorite seat!

Show that if Melon follows the same protocol for resolving this problem (i.e., walk clockwise until the first unoccupied seat), with probability at least $1 - \frac{1}{N^2}$, the number of the seat he ends up taking will be $O(\log n)$.

Hint: Prove that with probability at least $1 - \frac{1}{N^2}$*, there is no block of length* $C \cdot \log n$ *at the table, for some sufficiently large constant* C*.*

Solution: The Hint asks us to prove a stronger claim than necessary, but in doing so, we'll have solved the original problem with nicer bounds.

Let $q_{j,k}$ denote the probability that there exists a block of length *at least* k starting at seat j. Then, $q_{j,k} \leq \sum_{r=k}^{n} p_{j,r} \leq \sum_{r=k}^{n} \frac{1}{c^r} \leq \frac{d}{c^k}$ by Union Bound and part (b), where $d = \frac{1}{1-1/c}$ is a constant.

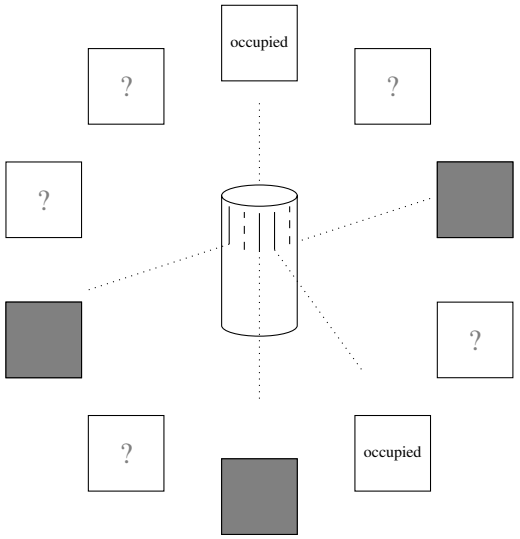
Let s_k be the probability that there exists a block of length at least k anywhere at the table. Taking another Union Bound over all possible start indices for the block, we have $s_k \leq \sum_{j=1}^{N} q_{j,k} = N \frac{d}{c^k}$.

Plugging in $k = 4 \log_c N = O(\log n)$ (or more precisely, $k = 3 \log_c N + \log_c d$), we have $s_{C \log n} \leq \frac{1}{N^2}$ for sufficiently large C.

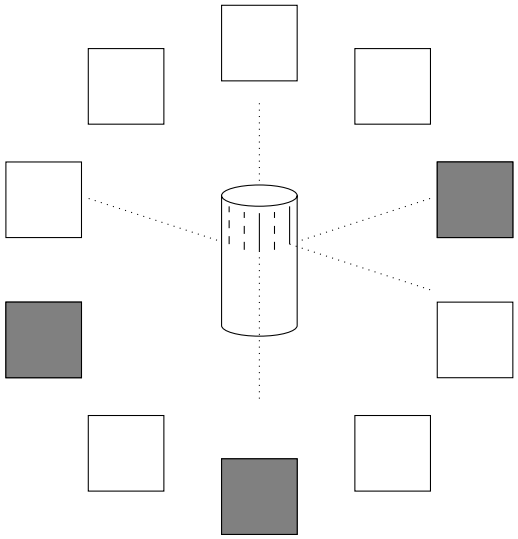
Problem-7: [15 points] **The Room Where It Happened**

As his right hand aide, Detective Baron Hurr takes you to a crime scene where you see a circle of n chairs, some of which have been splattered with paint. As you look to the side, you see a canister of paint with vertical slits along the circumference. Surprisingly, all the slits occur in multiples of $2\pi/n$. You realize that the can was in the middle of the circle when it erupted, causing paint to be shot through the slits. However, some of the chairs were occupied during this time and thus were shielded from paint, but you have no idea which chairs were occupied.

Given the positions of the slits and which chairs are painted, *how many orientations could the canister have been in when it exploded?*



(a) Valid orientation. Chairs with "occupied" must have been occupied. Chairs with "?" could have been occupied.



(b) Invalid orientation. In this orientation, not all painted chairs were shot at.

Figure 1: *Shaded squares are chairs with paint. In valid orientations, all painted chairs are aligned with a slit. (Slits on the back side of the canister are not shown.)*

Detective Hurr marks the canister clockwise with $a_1a_2a_3 \ldots a_n$ where a_i is 1 if the canister has a slit in position i and 0 otherwise. He then marks the chairs clockwise $b_1b_2b_3 \ldots b_n$ where b_i is 1 if the chair has paint on it and 0 otherwise.

In short, paint must have shot towards all painted chairs, but also could have shot towards unmarred chairs. The floor is a complete mess, and you can't tell anything about the trajectories of paint based on the rest of the room. Recall, you want to determine the number of valid orientations of the canister given the state of the room.

Everyone's got their eyes on you. What do you say? Design an algorithm to find the number of valid orientations of the canister in $O(n \log n)$ time.

Solution: Valid orientations of the canister have slits pointed towards all the chairs that have paint on them. (There can also be slits towards some of the chairs without paint because a person could have been sprayed instead.) We are trying to align the canister with the chairs. This can be obtained efficiently with convolution.

We use FFT to compute the convolution between $a_1 \dots a_n$ and $b_n b_{n-1} \dots b_1 b_n b_{n-1} \dots b_2$ (doubling and/or reversing string a_i instead of b_i yields an equivalent answer). We discard the first and last $n-1$ outputs as they correspond to not every location on the can being matched with a chair. The value of each of the rest of the outputs is the number of times the slits align with the chairs. For all the chairs to have received paint in that orientation, this must be equal to the number of chairs with paint. If β chairs have paint on them (that is, $\sum_{i=1}^{n} b_i = \beta$), then the number of remaining outputs with value β is the number of valid orientations.

This takes $O(n \log n)$ because we compute the convolution using FFT on arrays of size n and $2n-1$. The scan to check for values of k takes linear time, so the total time is $O(n \log n)$.