CS 170 Homework 2

Due **Monday 2/5/2024, at 10:00 pm (grace period until 11:59pm)**

1 Study Group

List the names and SIDs of the members in your study group. If you have no collaborators, you must explicitly write "none".

2 Median

Recall the quickselect algorithm used for quickly finding the median of an array. Suppose you ran quickselect on the following array to find it's median.

12	78	13	1	97	45	48	26	85	100	78

1. Suppose you always pick the first element as the pivot. List the all the subarrays that quickselect recurses on, as well as the index k – the k smallest element of the current subarray which the algorithm returns.

   For example, quickselect begins on the entire array [12, 78, 13, 1, 97, 45, 48, 26, 85, 100, 78] and $k = 6$.

   Solution:

subarray	k
78, 13, 97, 45, 48, 26, 85, 100, 78	4
13, 45, 48, 26	4
45, 48, 26	3
48	1

2. Specify which elements should be picked as pivots at each step in order to **maximize** the runtime of this algorithm. Write the numerical value of the elements, not their indices.

   Solution: In order to maximize runtime, we must continuously pick either largest element or smallest element in the current subarray. Once no more elements are smaller than the median, continuously pick the largest element and vice versa. This ensures that we decrease the subproblem as slowly as possible; thus, maximizing the runtime. The following is one such solution. Note that 78 will only be picked as the pivot once.

   1, 12, 13, 26, 45, 100, 97, 85, 78, 48

3. Specify which elements should be picked as pivots at each step in order to **minimize** the runtime of this algorithm. Write the numerical value of the elements, not their indices.

   Solution: To minimize the runtime, the first pivot picked should be the median.

   48

3 Median of Medians

The QuickSelect(A, k) algorithm for finding the kth smallest element in an unsorted array A picks an arbitrary pivot, then partitions the array into three pieces: the elements less than the pivot, the elements equal to the pivot, and the elements that are greater than the pivot. It is then recursively called on the piece of the array that still contains the kth smallest element.

(a) Consider the array $A = [1, 2, \ldots, n]$ shuffled into some arbitrary order. What is the worst-case runtime of QuickSelect(A, $n/2$) in terms of n? Construct a sequence of 'bad' pivot choices that achieves this worst-case runtime.

Hint: refer to Q2.

Solution: A single partition takes $\mathcal{O}(n)$ time on an array of size n. The worst case would be if the partition times were $n + (n-1) + \cdots + 2 + 1 = \mathcal{O}(n^2)$.

This would happen if the pivot choices were e.g. $1, 2, 3, \ldots, n/2-1, n, n-1, \ldots, n/2+1$. In each of these cases, the partition happens so that one piece only has one element, and the other piece has all the other elements. To get a better runtime, we would like the pieces to be move balanced.

The above 'worst case' has a chance of occurring even with randomly-chosen pivots, so the worst-case time for QuickSelect is $\mathcal{O}(n^2)$, even though it achieves $\Theta(n)$ on average.

Based on QuickSelect, let's define a new algorithm DeterministicSelect that deterministically picks a consistently good pivot every time. This pivot-selection strategy is called 'Median of Medians', so that the worst-case runtime of DeterministicSelect(A, k) is $\mathcal{O}(n)$.

Median of Medians

1. Group the array into $\lfloor n/5 \rfloor$ groups of 5 elements each (ignore any leftover elements)
2. Find the median of each group of 5 elements (as each group has a constant 5 elements, finding each individual median is $\mathcal{O}(1)$)
3. Create a new array with only the $\lfloor n/5 \rfloor$ medians, and find the true median of this array using DeterministicSelect.
4. Return this median as the chosen pivot

(b) Let p be the pivot chosen by DeterministicSelect on A. Show that at least $3n/10$ elements in A are less than or equal to p, and that at least $3n/10$ elements are greater than or equal p.

Solution: Let the choice of pivot be p. At least half of the groups ($n/10$) have a median m such that $m \leq p$. In each of these groups, 3 of the elements are at most the median m (including the median itself). Therefore, at least $3n/10$ elements are at most the size of the median.

The same logic follows for showing that $3n/10$ elements are at least the size of the median.

(a) Show that the worst-case runtime of DeterministicSelect(A, k) using the 'Median of Medians' strategy is $\mathcal{O}(n)$.

Hint: Using the Master theorem will likely not work here. Find a recurrence relation for $T(n)$, and try to use induction to show that $T(n) \leq c \cdot n$ for some $c > 0$.

Solution: We end up with the following recurrence:

$$T(n) \leq \underbrace{T(n/5)}_{\text{(A)}} + \underbrace{T(7n/10)}_{\text{(B)}} + \underbrace{d \cdot n}_{\text{(C)}}$$

(A) Calling DeterministicSelect to find the median of the array of medians

(B) The recursive call to DeterministicSelect after performing the partition. The size of the partition piece is always at most $7n/10$ due to the property proved in the previous part.

(C) The time to construct the array of medians, and to partition the array after finding the pivot. This is $\mathcal{O}(n)$, but we explicitly write that it is $d \cdot n$ for convenience in the next part.

We cannot simply use the Master theorem to unwind this recurrence. Instead, we show by induction that $T(n) \leq c \cdot n$ for some $c > 0$. The base case happens when DeterministicSelect occurs on one element, which is constant time.

For the inductive case,

$$\begin{aligned} T(n) &\leq T(n/5) + T(7n/10) + d \cdot n \\ &\leq c(n/5) + c(7n/10) + d \cdot n \\ &\leq \left(\frac{9}{10}c + d\right) \cdot n \end{aligned}$$

We pick c large enough so that $\left(\frac{9}{10}c + d\right) \leq c$, i.e. $c \geq 10d$, and we are finished. Because $T(n) \leq c \cdot n$ for constant c, $T(n) = \mathcal{O}(n)$.

4 The Resistance

We are playing a variant of The Resistance, a board game where there are n players, s of which are spies. In this variant, in every round, we choose a subset of players to go on a mission. A mission succeeds if the subset of the players does not contain a spy, but fails if at least one spy goes on the mission. After a mission completes, we only know its outcome and not which of the players on the mission were spies.

Come up with a strategy that identifies all the spies in $O(s \log(n/s))$ missions. **Describe your strategy and analyze the number of missions needed.**

Hint 1: consider evenly splitting the n players into x disjoint groups (containing $\approx n/x$ players each), and send each group on a mission. At most how many of these x missions can fail? What should you set x to be to ensure that you can reduce your problem by a factor of at least $1/2$?

Hint 2: it may help to try a small example like $n = 8$ and $s = 2$ by hand.

Solution: Observe that if we partition the players into x disjoint groups and send each group on a mission, at least $x - s$ groups will succeed and we can rule out the players in those groups.

To discard at least half of the players in each iteration, partition them into $2s$ groups. For the missions that succeed, we remove those groups, and split the remaining groups in half to form a new set of groups. We repeat this procedure until each group has one person left, at which point we know they are a spy.

Runtime analysis: In the first iteration of this procedure we have $2s$ groups going on missions. After each group goes on a mission, since there are s spies, at most s of the missions fail, which means after splitting the groups that failed, we have at most $2s$ groups again.

In each iteration, at least half of the players are removed from groups. So we identify the spies after $O(\log(n/s))$ iterations. So the total number of missions needed is $O(s \log(n/s))$.

Alternate solution: Divide the players into two groups and send each group on a mission. If a group succeeds then discard those players and if a group fails then recursively use the same strategy on that group until a group of size 1 is reached.

Runtime Analysis: After drawing out the recursion tree you'll see that at level i, at most $\min(2^i, s)$ missions can fail and those are the nodes that will branch out to the next level. There are at most $\log n$ levels. Hence, the total number of missions is bounded by,

$$\begin{aligned}\sum_{i=i}^{\log n} \min(2^i, s) &\leq \sum_{i=1}^{\log s} 2^i + \sum_{i=\log s}^{\log n} s \\ &= O(s) + s(\log n - \log s) \\ &= O(s \log(n/s))\end{aligned}$$

5 Among Us

You are playing a party game with n other friends, who play either as imposters or crewmates. You do not know who is a crewmate and who is a imposter, but all your friends do. There are always more crewmates than there are imposters.

Your goal is to identify one player who is certain to be a crewmate.

Your allowed 'query' operation is as follows: you pick two players as partners. You ask each player if their partner is a crewmate or a imposter. When you do this, a crewmate must tell the truth about the identity of their partner, but a imposter doesn't have to (they may lie or tell the truth about their partner).

Your algorithm should work regardless of the behavior of the imposters.

(a) For a given player x, devise an algorithm that returns whether or not x is a crewmate using $O(n)$ queries. Just an informal description of your test and a brief explanation of why it works is needed.

(b) Show how to find a crewmate in $O(n \log n)$ queries (where one query is taking two players x and y and asking x to identify y and y to identify x).

Hint: Split the players into two groups, recurse on each group, and use part (a). What invariant must hold for at least one of the two groups?

Give a 3-part solution.

(c) (**Optional, not for credit**) Can you give a $O(n)$ query algorithm?

Hint: Don't be afraid to sometimes 'throw away' a pair of players once you've asked them to identify their partners.

Give a 3-part solution.

Solution:

(a) To test if a player x is a crewmate, we ask the other $n-1$ players what $x's$ identity is. Claim: x is a crewmate if and only if at least half of the other players say x is a crewmate. To see this, notice that if x is a crewmate, at least half of the remaining players are also crewmates, and so regardless of what the imposters do at least half of the players will say x is a crewmate. On the other hand, if x is a imposter, then strictly more than half of the remaining players are crewmates, and so strictly less than half the players can falsely claim that $x$is a crewmate.

(b) **Main idea** The divide and conquer algorithm to find a crewmate proceeds by splitting the group of friends into two (roughly) equal sets A and B, and recursively calling the algorithm on A and B: $x = crewmate(A)$ and $y = crewmate(B)$, and checking x or y using the procedure in part (a) and returning one who is a crewmate. If there is only one player left, they are guaranteed to be a crewmate, so return that player.

Proof of correctness We will prove that the algorithm returns a crewmate if given a group of n players of which a majority are crewmates. By strong induction on n:

Base Case If $n = 1$, there is only one player in the group who is a crewmate and the algorithm is trivially correct.

Induction Hypothesis The claim holds for $k < n$.

Induction Step After partitioning the group into two groups A and B, at least one of the two groups has more crewmates than imposters. By the induction hypothesis the algorithm correctly returns a crewmate from that group, and so when the procedure from part (a) is invoked on x and y at least one of the two is identified as a crewmate.

Runtime analysis Two calls to problems of size $n/2$, and then linear time to compare the two players returned to each of the friends in the input group: $T(n) = 2T(\frac{n}{2}) + O(n) = O(n \log n)$ by Master Theorem.

Note to graders: $O(n)$ query solutions, as in part (c), should also get full credit.

(c) **Main idea** Split up the friends into pairs and for each pair, if either says the other is a imposter, discard both friends; otherwise, discard any one and keep the other friend. If n was odd, use part (a) to test whether the odd man out is a crewmate. If yes, you are done, else recurse on the remaining at most $n/2$ friends.

Proof of correctness After each pass through the algorithm, crewmates remain in the majority if they were in the majority before the pass. To see this, let n_1, n_2 and n_3 be the number of pairs of friends with both crewmates, both imposters and one of each respectively. Then the fact that crewmates are in the majority means that $n_1 > n_2$. Note that all the n_3 pairs of the third kind get discarded, and one friend is retained from each of the n_1 pairs of the first kind. So we are left with at most $n_1 + n_2$ friends of whom a majority n_1 are crewmates. It is straightforward to now turn this into a formal proof of correctness by strong induction on n.

Runtime analysis In a single run of the algorithm on an input set of size n, we do $O(n)$ work to check whether f_1 is a crewmate in the case that n is odd and $O(n)$ to pair up the remaining friends and prune the candidate set to at most $n/2$ players. Therefore, the runtime is given by the following recursion:

$$T(n) = T\left(\frac{n}{2}\right) + O(n) = O(n) \text{ by Master Theorem.}$$

Alternative solution: Pick a player at random and use part (a) to check whether they are crewmate, if not then repeat the process again. Since at least half the players are crewmates, the **expected** number of players we will have to check until we find a crewmate is $O(1)$. Since the check requires $O(n)$ queries, the overall solution also uses $O(n)$ queries **in expectation**.

6 Modular Fourier Transform

Fourier transforms (FT) have to deal with computations involving irrational numbers which can be tricky to implement in practice. Motivated by this, in this problem you will demonstrate how to do a Fourier transform in modular arithmetic, using modulo 5 as an example.

(a) There exists $\omega \in \{0,1,2,3,4\}$ such that $\{\omega^0, \omega^1, \omega^2, \omega^3\}$ the are 4^{th} roots of unity (modulo 5), i.e., solutions to $z^4 = 1 \pmod 5$. When doing the FT in modulo 5, this ω will serve a similar role to the primitive root of unity in our standard FT. Show that $\{1,2,3,4\}$ are the 4^{th} roots of unity (modulo 5), with $\omega = 2$ as the primitive root. Also show that $1 + \omega + \omega^2 + \omega^3 = 0 \pmod 5$ for $\omega = 2$.

(b) Using the FFT, produce the transform of the sequence $(0,2,3,0)$ modulo 5; that is, evaluate the polynomial $2x + 3x^2$ at $\{1,2,4,3\}$ using the recursive FFT algorithm defined in class, but with $\omega = 2$ and in modulo 5 instead of with $\omega = i$ in the complex numbers. Note that all calculations should be performed modulo 5.

Hint: You can verify your calculation by evaluating the polynomial at $\{1,2,4,3\}$ *using the slow method (i.e. DFT).*

(c) Now perform the inverse FFT on the sequence $(0,1,4,0)$, also using the recursive algorithm. Recall that the inverse FFT is the same as the forward FFT, but using ω^{-1} instead of ω, and with an extra multiplication by 4^{-1} for normalization.

(d) Now show how to multiply the polynomials $2x + 3x^2$ and $3 - x$ using the FFT modulo 5. You may use the fact that the FFT of $(3,4,0,0)$ modulo 5 is $(2,1,4,0)$ without doing your own calculation.

Solution:

(a) We can check that $1^4 = 1 \pmod 5$,
$2^4 = 16 = 1 \pmod 5$,
$3^4 = 81 = 1 \pmod 5$,
$4^4 = 256 = 1 \pmod 5$.

Observe that taking $\boxed{\omega = 2}$ produces the following powers: $(\omega, \omega^2, \omega^3) = (2,4,3)$. Verify that

$$1 + \omega + \omega^2 + \omega^3 = 1 + 2 + 4 + 3 = 10 = 0 \pmod 5.$$

(b) Let $p(x) = 2x + 3x^2$. Then, the FFT divide-and-conquer algorithm gives us

$$\text{FFT}(0,2,3,0) = \begin{pmatrix} p(\omega^0) \\ p(\omega^1) \\ p(\omega^2) \\ p(\omega^3) \end{pmatrix} = \begin{pmatrix} E((\omega^0)^2) + \omega^0 \cdot O((\omega^0)^2) \\ E((\omega^1)^2) + \omega^1 \cdot O((\omega^1)^2) \\ E((\omega^2)^2) + \omega^2 \cdot O((\omega^2)^2) \\ E((\omega^3)^2) + \omega^3 \cdot O((\omega^3)^2) \end{pmatrix} = \begin{pmatrix} E(\omega^0) + \omega^0 \cdot O(\omega^0) \\ E(\omega^2) + \omega^1 \cdot O(\omega^2) \\ E(\omega^0) - \omega^0 \cdot O(\omega^0) \\ E(\omega^2) - \omega^1 \cdot O(\omega^2) \end{pmatrix}$$

$$= \begin{pmatrix} \text{FFT}(0,3)[0] + 1 \cdot \text{FFT}(2,0)[0] \\ \text{FFT}(0,3)[1] + 2 \cdot \text{FFT}(2,0)[1] \\ \text{FFT}(0,3)[0] + 1 \cdot \text{FFT}(2,0)[0] \\ \text{FFT}(0,3)[1] + 2 \cdot \text{FFT}(2,0)[1] \end{pmatrix}$$

where $E(x), O(x)$ are the even and odd polynomials that form p (as described in lecture and DPV). Note that by definition, $\text{FFT}(0,3) = (E(\omega^0), E(\omega^2))$ and $\text{FFT}(2,0) = (O(\omega^0), O(\omega^2))$.

Then, we further recurse on $\text{FFT}(0,3)$ and $\text{FFT}(2,0)$ as follows:

$$\text{FFT}(0,3) = \begin{pmatrix} \text{FFT}(0) + 1 \cdot \text{FFT}(3) \\ \text{FFT}(0) - 1 \cdot \text{FFT}(3) \end{pmatrix}$$
$$\text{FFT}(2,0) = \begin{pmatrix} \text{FFT}(2) + 1 \cdot \text{FFT}(0) \\ \text{FFT}(2) - 1 \cdot \text{FFT}(0) \end{pmatrix}$$

The FFT of one element does nothing, so $\text{FFT}(0) = 0$, $\text{FFT}(2) = 2$, and $\text{FFT}(3) = 3$. This gives

$$\text{FFT}(0,3) = (3,-3) = (3,2) \pmod 5$$
$$\text{FFT}(2,0) = (2,2) \pmod 5$$

and finally,

$$\text{FFT}(0,3,2,0) = \begin{pmatrix} 3+2 \\ 2+4 \\ 3-2 \\ 2-4 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 3 \end{pmatrix} \pmod 5.$$

(c) Note that $\omega^{-1} = 3$ and $4^{-1} = 4 \pmod 5$, so we are trying to compute

$$4^{-1} \cdot \text{iFFT}(0,1,4,0) = 4 \cdot \begin{pmatrix} \text{iFFT}(0,4)[0] + 1 \cdot \text{iFFT}(1,0)[0] \\ \text{iFFT}(0,4)[1] + 3 \cdot \text{iFFT}(1,0)[1] \\ \text{iFFT}(0,4)[0] - 1 \cdot \text{iFFT}(1,0)[0] \\ \text{iFFT}(0,4)[1] - 3 \cdot \text{iFFT}(1,0)[1] \end{pmatrix}$$

Following a similar recursive procedure as part (b), we then compute

$$\text{iFFT}(0,4) = \begin{pmatrix} 0+4 \\ 0-4 \end{pmatrix} = \begin{pmatrix} 4 \\ 1 \end{pmatrix} \pmod 5$$
$$\text{iFFT}(1,0) = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \pmod 5$$

Note that the unnormalized inverse FFT at $n = 2$ and below is identical to the forward FFT!

Hence, we have

$$\text{iFFT}(0,1,4,0) = \begin{pmatrix} 4+1 \\ 1+3 \\ 4-1 \\ 1-3 \end{pmatrix} = \begin{pmatrix} 0 \\ 4 \\ 3 \\ 3 \end{pmatrix},$$

which gives a final answer of

$$4 \cdot (0,4,3,3) = (0,1,2,2).$$

(d) We will run our FFT on the coefficient representations of the polynomials, point-wise multiply the resulting vectors, and then run the inverse FFT on the resulting vector by to get a coefficient representation for their product.

We already have that the FFT of $2x + 3x^2$ is $(0, 1, 1, 3)$ from part (b); we have that the FFT of $3 - x$ is $(2, 1, 4, 0)$ given in the question (since $-1 = 4$ modulo 5, and so $(3, 4, 0, 0)$ is the coefficient representation of $3 - x$).

Multiplying these elementwise gives us $(0, 1, 4, 0)$ as our answer modulo 5.

We already know from our answer to part (c) that the inverse FFT of this is $(0, 1, 2, 2)$, so our final answer is $x + 2x^2 + 2x^3 \pmod 5$. We can check that this is correct by multiplying the polynomials directly using FOIL.

7 Coding Question: FFT

For this week's coding questions, you'll implement the **Fast Fourier Transform (FFT)** algorithm and apply it to speed up polynomial multiplication. There are two ways that you can access the notebook and complete the problems:

1. **On Datahub**: click here and navigate to the `hw02` folder.
2. **On Local Machine**: `git clone` (or if you already cloned it, `git pull`) from the coding homework repo,

   https://github.com/Berkeley-CS170/cs170-sp24-coding

   and navigate to the `hw02` folder. Refer to the `README.md` for local setup instructions.

Notes:

- *Submission Instructions:* Please download your completed submission `.zip` file and submit it to the Gradescope assignment titled "Homework 2 Coding Portion".
- *Getting Help:* Conceptual questions are always welcome on edstem and office hours; *note that support for debugging help during OH will be limited.* If you need debugging help first try asking on the public edstem threads. To ensure others can help you, make sure to:

  1. Describe the steps you've taken to debug the issue prior to posting on Ed.
  2. Describe the specific error you're running into.
  3. Include a few small but nontrivial test cases, alongside both the output you expected to receive and your function's actual output.

  If staff tells you to make a private Ed post, make sure to include *all of the above items* plus your full function implementation. If you don't provide them, we will ask you to provide them.
- *Academic Honesty Guideline:* We realize that code for some of the algorithms we ask you to implement may be readily available online, but we strongly encourage you to not directly copy code from these sources. Instead, try to refer to the resources mentioned in the notebook and come up with code yourself. That being said, we **do acknowledge** that there may not be many different ways to code up particular algorithms and that your solution may be similar to other solutions available online.