CS 170 Homework 2

Due **2/3/2020, at 10:00pm**

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 Asymptotic Properties

Provide a proof of each of the following properties of Θ:

(a) For all fixed $c > 0$, $\sum_{i=1}^{n} i^c = \Theta(n^{c+1})$.

Solution: $\sum_{i=1}^{n} i^c \leq \sum_{i=1}^{n} n^c = n^{c+1}$, so $\sum_{i=1}^{n} i^c = \mathcal{O}(n^{c+1})$.
$\sum_{i=1}^{n} i^c \geq \sum_{i=n/2}^{n} (n/2)^c = \frac{n^{c+1}}{2^{c+1}}$, so $\sum_{i=1}^{n} i^c = \Omega(n^{c+1})$.

(b) Consider $\sum_{i=0}^{n} \alpha^i$, where α is a constant. Three different things may happen, depending on the value of α:

(i) Show that if $\alpha > 1$, $\sum_{i=0}^{n} \alpha^i = \Theta(\alpha^n)$.

Solution: $\sum_{i=0}^{n} \alpha^i = \frac{\alpha^{n+1}-1}{\alpha-1} = \frac{\alpha^{n+1}}{\alpha-1} - \frac{1}{\alpha-1} = \Theta(\alpha^n)$
In this case, we can keep only the leading term, α^n.

(ii) Show that if $\alpha = 1$, $\sum_{i=0}^{n} \alpha^i = \Theta(n)$.

Solution: $\sum_{i=0}^{n} \alpha^i = \sum_{i=0}^{n} 1^i = n + 1 = \Theta(n)$

(iii) Show that if $0 < \alpha < 1$, $\sum_{i=0}^{n} \alpha^i = \Theta(1)$.

Solution: In this case, $\frac{\alpha^{n+1}}{\alpha-1} - \frac{1}{\alpha-1} = \Theta(1)$. As $\alpha < 1$, the second term (which is constant) is the leading term over the first (which is exponentially decaying).

3 In Between Functions

Find a function $f(n) \geq 0$ such that:

- For all $c > 0$, $f = \Omega(n^c)$
- For all $\alpha > 1$, $f = \mathcal{O}(\alpha^n)$

Give a proof for why it satisfies both these properties.

Solution: Let $f(n) = 2^{(\log n)^2}$.

- For any $n^c = 2^{c \log n}$, this is eventually dominated by $2^{(\log n) \cdot (\log n)}$. So $f(n) = \Omega(n^c)$ for any $c > 0$.

- For any $\alpha^n = \left(2^{\log \alpha}\right)^n = 2^{n \cdot \log \alpha}$, this will dominate $2^{(\log n)^2}$ (so long as $\log \alpha$ is positive). So $f(n) = \mathcal{O}(\alpha^n)$ for any $\alpha > 1$.

This shows that there can be algorithms whose running time grows faster than any polynomial but slower than any exponential.
In other words, there exists a region between polynomial-time and exponential-time.

4 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.

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 $1, 2, 3, \ldots, n$ or $n, (n-1), \ldots, 1$. 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.

(b) The above 'worst case' has a chance of occurring even with randomly-chosen pivots, so the worst-case time for a random-pivot QUICKSELECT is $\mathcal{O}(n^2)$.

Let's define a new algorithm BETTER-QUICKSELECT that deterministically picks a better pivot. This pivot-selection strategy is called 'Median of Medians', so that the worst-case runtime of BETTER-QUICKSELECT(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 BETTER-QUICKSELECT.
4. Return this median as the chosen pivot

Let p be the chosen pivot. Show that for least $3n/10$ elements x we have that $p \geq x$, and that for at least $3n/10$ elements we have that $p \leq x$.

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.

(c) Show that the worst-case runtime of Better-Quickselect(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 Better-Quickselect to find the median of the array of medians

(B) The recursive call to Better-Quickselect 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 Better-Quickselect 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, $T(n) = \mathcal{O}(n)$.

5 Two sorted arrays

You are given two sorted arrays, each of size n. Give as efficient an algorithm as possible to find the k-th smallest element in the union of the two arrays. What is the running time of your algorithm as a function of k and n?

Hint: You can use techniques similar to those developed in the previous problem.

Give a 3-part solution.

Solution:

Main idea

We will assume $k \leq n$. If this is not the case, then for $k' = 2n + 1 - k$, we are trying to find the k'-th largest element, and $k' \leq n$. But finding the k-th largest element and finding the k-th smallest element are symmetric problems, so with some minor tweaks we can use the same algorithm. So for brevity, we will only give the answer for when $k \leq n$.

Since $k \leq n$, the kth smallest element can't exist past index k of either array. So we cut off all but the first k elements of both arrays. Now, we just want to find the median of the union of the two arrays. To do this, we check the middle elements of both arrays, and compare them. If, the median of the first list is smaller than the median of the second list, then the median can't be in the left half of the first list or the right half of the second list. If the median of the second list is smaller, the opposite is true. So we can cut these elements off, and then recurse on the resulting arrays.

Pseudocode

procedure TwoArraySelection($a[1..n]$, $b[1..n]$, element rank k)
  Set $a := a[1, \cdots, k]$, $b := b[1, \cdots, k]$,
  Let $\ell_1 = \lfloor k/2 \rfloor$ and $\ell_2 = \lceil k/2 \rceil$
  while $a[\ell_1] \neq b[\ell_2]$ and $k > 1$ **do**
    if $a[\ell_1] > b[\ell_2]$ **then**
      Set $a := a[1, \cdots, \ell_1]$; $b := b[\ell_2 + 1, \cdots, k]$; k := ℓ_1
      Let $\ell_1 = \lfloor k/2 \rfloor$ and $\ell_2 = \lceil k/2 \rceil$
    else
      Set $a := a[\ell_1 + 1, \cdots, k]$; $b := b[1, \cdots, \ell_2]$; $k := \ell_2$
      Let $\ell_1 = \lfloor k/2 \rfloor$ and $\ell_2 = \lceil k/2 \rceil$
  if $k = 1$ **then**
    return min($a[1]$, $b[1]$)
  else
    return $a[\ell_1]$

Proof of correctness

Our algorithm starts off by comparing elements $a[\ell_1]$ and $b[\ell_2]$. Suppose $a[\ell_1] > b[\ell_2]$.

Then, in the union of a and b there can be at most $k - 2$ elements smaller than $b[\ell_2]$, i.e. $a[1, \cdots, \ell_1 - 1]$ and $b[1, \cdots, \ell_2 - 1]$, and we must necessarily have $s_k > b[\ell_2]$. Similarly, all elements $a[1, \cdots, \ell_1]$ and $b[1, \cdots, \ell_2]$ will be smaller than $a[\ell_1 + 1]$; but these are k elements, so we must have $s_k < a[\ell_1 + 1]$.

This shows that s_k must be contained in the union of the subarrays $a[1, \cdots, \ell_1]$ and $b[\ell_2 + 1, \cdots, k]$. In particular, because we discarded ℓ_2 elements smaller than s_k, s_k will be the ℓ_1th smallest element in this union.

We can then find s_k by recursing on this smaller problem. The case for $a[\ell_1] < b[\ell_2]$ is symmetric.

If we reach $k = 1$ before $a[\ell_1] = b[\ell_2]$, we can cut the recursion a little short and just return the minimum element between a and b. You can make the algorithm work without this check, but it might get clunkier to think about the base cases.

Alternatively, if we reach a point where $a[\ell_1] = b[\ell_2]$, then there are exactly k greater elements, so we have $s_k = a[\ell_1] = b[\ell_2]$.

Running time analysis

At every step we halve the number of elements we consider, so the algorithm will terminate in $\log(k)$ recursive calls. Assuming the comparison takes constant time, the algorithm runs in time $\Theta(\log k)$. Note that this does not depend on n.

6 Werewolves

You are playing a party game with n other friends, who play either as werewolves or citizens. You do not know who is a citizen and who is a werewolf, but all your friends do. There are always at least two more citizens than there are werewolves.

Your goal is to identify all the citizens exactly.

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

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

(a) Show how to solve this problem in $O(n \log n)$ queries (where one query is asking a pair of people to identify the other person).

You cannot use a linear-time algorithm here, as we would like you to get practice with divide and conquer.

Give a 3-part solution.

(b) (**Extra Credit**) Can you give a linear-time algorithm?

Give a 3-part solution.

Solution:

(a) We only need to identify *one* citizen, and then we can simply ask that one citizen to identify all the other citizens/werewolves in $\mathcal{O}(n)$ more pairwise operations. So the main part is to devise a divide-and-conquer algorithm to discover one citizen.

Main idea For the divide and conquer approach, we will try to devise an algorithm with the guarantee that if it is given a set of friends A with more than half being citizens, it returns a citizen. To this end, we start by first partitioning the group of friends G into two sub-groups A and B of roughly equal size (If n is odd, one of the sets will have 1 more element than the other) and running the algorithm recursively on each set A and B. Suppose, the candidates returned by the algorithm on A and B are f_1 and f_2. We then ask f_1 and f_2 to interact with each of the people in G and return whoever agrees with more people. In case of a tie, we may return either one.

Proof of correctness By strong induction on n:

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

Induction hypothesis The algorithm is correct for $k \leq n$.

Induction step: Suppose the algorithm is given as input G with $|G| = n+1$ and more than half of the friends in G are citizens. Then, after partitioning the group into two groups A and B, at least one of the two groups has citizens as more than half of its members. Let the algorithm return f_1 and f_2 on input A and B respectively. Therefore, at least one of f_1 and f_2 is a citizen. If both are citizens, either one is returned and the algorithm remains correct. Otherwise, more than half of the friends on G will disagree with the werewolf and therefore, we will return the citizen. Thus, the algorithm remains correct in this case.

Running time analysis Two calls to problems of size $n/2$, and then linear time to compare the two people 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.

(b) **Main idea** The main idea is that removing a pair of people one of whom is a werewolf does not change the fact that the citizens are in the majority. As before, we will try to devise an algorithm to output a single citizen. We will maintain a set of candidate citizens which will initially be the entire set of friends G and gradually reduce the size of the set until it contains only a single element. If n is odd, we choose one person, say f_1 from the set of friends and ask them to interact with each of the remaining $n-1$ people. If at least $(n-1)/2$ people agree that f_1 is a citizen, we say f_1 and everyone who agrees with them is a citizen. Otherwise, we remove f_1 as a candidate citizen and recurse on the remaining $n-1$ people. Therefore, we may assume that n is even. We now pair up people in the group arbitrarily to form $n/2$ groups. Suppose the groups are $(f_1, f_2), \ldots, (f_{n-1}, f_n)$. We ask the two members in each group whether the other member is a citizen. If both report that the other member is a citizen, we retain one of the members arbitrarily. If either member of a pair declares that the other is not a citizen, we remove both of them from our candidate set. Through this process, we obtain a new set G' that is almost half the size of the original set. We then recurse with the new set G' and return when we are reduced to a set with a single element.

Proof of correctness As before, we will prove the correctness of our algorithm by strong induction on the number of friends playing the game n:

Base Case If $n = 1$, the group only contains one friend who is guaranteed to be a citizen.

Induction hypothesis Given a set of people of size k where more than half the people are citizens, the algorithm correctly returns a citizen.

Induction step If n is odd, we first choose a friend, say f_1 and ask every other friend in the list whether f_1 is a citizen. Recall, we return f_1 as a citizen if at least $(n-1)/2$ people agree that f_1 is a citizen. Otherwise, we remove f_1 from our set of citizen candidates. Now, if f_1 is indeed a citizen, at least $(n-1)/2$ friends would agree that they are a citizen as more than half of the n friends are citizens. In this case, we would correctly return f_1. Otherwise, if f_1 is a werewolf, there are at least $(n+1)/2$

citizens in the set of n friends who would deny that they are a citizen, in which case, we would rightly discard them. Now, the correctness of the algorithm for input size $k = n$ follows from the induction hypothesis as we will recurse on a set of candidates of size $n - 1$ where the citizens are still in the majority as we have simply removed a werewolf.

Consider the case where n is even. In this case, we have pairs of the form $(f_1, f_2), (f_3, f_4), \ldots (f_{n-1}, f_n)$. Suppose, we first get rid of all the pairs which one of the friends claims the other is a werewolf. Let (f, g) be such a pair and suppose that f claims that g is a werewolf. Either f is a citizen, in which case g is a werewolf or f is werewolf. In either case, the pair (f, g) contains at least one werewolf. Suppose there are m such pairs. Removing all such pairs, we remove at least m werewolves and at most m citizens. Therefore, among the remaining $n-2m$ candidates, we still have the property that most of the candidates are citizens. Now, in the remaining pairs, we have that each pair consist of either two citizens or two werewolves as in a pair with a citizen and werewolf, the citizen would've reported the other member a werewolf. By arbitrarily picking one member of each pair to remain in our candidate set, we simply reduce the number of candidates by half exactly halving the number of citizens and werewolves forming a new set of size $(n-2m)/2$ where the majority of the people are still citizens. The correctness of the algorithm follows from the induction hypothesis.

Running time 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 citizen in the case that n is odd and to pair up the remaining friends and pruning the candidate set to at most $n/2$ people. Therefore, the runtime is given by the following recursion:

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

7 Hadamard matrices

The Hadamard matrices $H_0, H_1, H_2, \ldots$ are defined as follows:

- H_0 is the 1×1 matrix $[1]$
- For $k > 0, H_k$ is the $2^k \times 2^k$ matrix

$$H_k = \left[\begin{array}{c|c} H_{k-1} & H_{k-1} \\ \hline H_{k-1} & -H_{k-1} \end{array}\right]$$

(a) Write down the Hadamard matrices H_0, H_1, and H_2.

Solution: $H_0 = 1$

$$H_1 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$H_2 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

(b) Compute the matrix-vector product $H_2 \cdot v$ where H_2 is the Hadamard matrix you found above, and

$$v = \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix}$$

Note that since H_2 is a 4×4 matrix, and the vector has length 4, the result will be a vector of length 4.

Solution:

$$H_2 v = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 4 \end{bmatrix}$$

(c) Now, we will compute another quantity. Take v_1 and v_2 to be the top and bottom halves of v respectively. Therefore, we have that

$$v_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, v_2 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

Compute $u_1 = H_1(v_1 + v_2)$ and $u_2 = H_1(v_1 - v_2)$ to get two vectors of length 2. Stack u_1 above u_2 to get a vector u of length 4. What do you notice about u?

Solution:

$$H_1(v_1 + v_2) = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$H_1(v_1 - v_2) = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 2 \\ -2 \end{bmatrix} = \begin{bmatrix} 0 \\ 4 \end{bmatrix}$$

We notice that $u = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 4 \end{bmatrix} = H_2 v$

(d) Suppose that

$$v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

is a column vector of length $n = 2^k$. v_1 and v_2 are the top and bottom half of the vector, respectively. Therefore, they are each vectors of length $\frac{n}{2} = 2^{k-1}$. Write the matrix-vector product $H_k v$ in terms of H_{k-1}, v_1, and v_2 (note that H_{k-1} is a matrix of dimension $\frac{n}{2} \times \frac{n}{2}$, or $2^{k-1} \times 2^{k-1}$). Since H_k is a $n \times n$ matrix, and v is a vector of length n, the result will be a vector of length n.

Solution: $H_k v = \begin{bmatrix} H_{k-1} v_1 + H_{k-1} v_2 \\ H_{k-1} v_1 - H_{k-1} v_2 \end{bmatrix} = \begin{bmatrix} H_{k-1}(v_1 + v_2) \\ H_{k-1}(v_1 - v_2) \end{bmatrix}$

(e) Use your results from (c) to come up with a divide-and-conquer algorithm to calculate the matrix-vector product $H_k v$, and show that it can be calculated using $O(n \log n)$ operations. Assume that all the numbers involved are small enough that basic arithmetic operations like addition and multiplication take unit time. You do not need to prove correctness.

Solution: Compute the 2 subproblems described in part (d), and combine them as described in part (c). Let $T(n)$ represent the number of operations taken to find $H_k v$. We need to find the vectors $v_1 + v_2$ and $v_1 - v_2$, which takes $O(n)$ operations. And we need to find the matrix-vector products $H_{k-1}(v_1 + v_2)$ and $H_{k-1}(v_1 - v_2)$, which take $T(\frac{n}{2})$ number of operations. So, the recurrence relation for the runtime is:

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

Using the master theorem, this give us $T(n) = O(n \log n)$.

8 Warmup: FFT

The *n-th roots of unity* are the n complex numbers ω such that $\omega^n = 1$. They are:

$$e^{2\pi ik/n}, \qquad k = 0, 1, 2, \ldots, n-1$$

(a) Find an n-th root of unity ω such that for all n-th roots of unity z, $z = \omega^k$. In other words, find an ω such that all n-th roots of unity are powers of ω.

Solution: This is $e^{2\pi i/n}$. Notice that $\left(e^{2\pi i/n}\right)^k = e^{2\pi ik/n}$.

(b) Show that the squares of the $2n$-th roots of unity are the n-th roots of unity.

Hint: This requires showing two things: (1) the square of every $2n$-th root of unity is an n-th root of unity, and (2) every n-th root of unity is the square of some $2n$-th root of unity.

Solution:

(1) For any k, $\left(e^{2\pi ik/2n}\right)^2 = e^{2\pi i2k/2n} = e^{2\pi ik/n}$

(2) For any k, $e^{2\pi ik/n} = \left(e^{2\pi ik/2n}\right)^2$

(c) For a given n-th root of unity $\omega = e^{2\pi ik/n}$, determine all of the $2n$-th roots of unity whose squares are ω.

Solution:

$$e^{2\pi ik/2n}, e^{2\pi i(k+n)/2n}$$

There can only be two: there are $2n$ $2n$-th roots of unity and n n-th roots of unity, and each n-th root of unity has exactly two $2n$-th roots of unity that 'square' to it.