Homework 2

CS41, Spring 2023 Nina Zhuo

1. We imagine some function `indexMatch` that takes array A of distinct integers, an integer m where m is the starting index, and an integer n where n is end index of the array.

   We set $i = (n - m)/2$ such that i is the middle index of array A. Since A is sorted, we know that value stored at $A[i]$ the, or one of the, middle values of the array such that

   $$A[m] \text{ to } A[i-1] < A[i] < A[i+1] \text{ to } A[n]$$

   In the case that $A[i] = i$, we return `true` because there exists some index i such that $A[i] = i$. In the case that $m > n$, we return `false` because when the start index is greater than the end index, we know that we have looked at every item in the array and there is no index i such that $A[i] = i$.

   In the case that $A[i] < i$, we know that all numbers to the left of index i in array A will be $< i$. In other words, if the value stored at $A[i]$ is greater than its index i, it stands to reason that all values to the right of $A[i]$ will also be greater than their respective indices because as i increments by 1, the sorted values in array A will increment by at least one. So, we disregard that side of the array and recursively call `indexMatch` on array A, the start index m, and the end index $n = i - 1$ so we only search through the right side of index i.

   In the case that $A[i] > i$, we know that all numbers to the right of index i in array A will be $> i$. In other words, if the value stored at $A[i]$ is less than its index i, it stands to reason that all values to the left of $A[i]$ will also be less than their respective indices because as i decrements by 1, the sorted values in array A will decrement by at least one. So, we disregard that side of the array and recursively call `indexMatch` on array A, the start index $m = i + 1$, and the end index n.

   Proof. We prove by induction that this function works.

   Base Case: When `A` is of length $k = 0$, the start index $m = -1$ and end index $n = 0$. Since $0 > -1$, `indexMatch` will return `false`.

   Inductive Hypothesis: Assume this function works for any array whose size is $\geq k$. Then, this function will also work for any array whose size is $k + 1$.

   Inductive Step: From here, we have three cases:

   Case 1: $A[i] = i$

   Since the value stored at index i matches the index, `indexMatch` will return `true`.

   Case 2: $A[i] < i$

   Since the value stored at index i is less than the index and we know that array A is a set of distinct integers, we recursively call `indexMatch` to check the values stored in the second unchecked half of the sorted array from $A[i + 1]$ to the largest unchecked index $A[n]$.

Case 3: $A[i] > i$

Since the value stored at index i is greater than the index and we know that array A is a set of distinct integers, we recursively call `indexMatch` to check the values stored in the first unchecked half of the sorted array from smallest unchecked index $A[m]$ to $A[i-1]$.

$\square$

We can find the runtime of `indexMatch` using Master Theorem:

$$T(n) \leq a \cdot T(\frac{n}{b}) + O(n^c)$$

In this recursion, the array A remains the same size, but we only operate on subsets of size $n/2$, so $a = 1$ and $b = 2$. The only non-recursive work to account for involves returning `true` or `false` which takes constant time $O(1)$ or $O(n^0)$. Thus, $c = 0$ such that

$$T(n) \leq T(\frac{n}{2}) + O(1)$$

From here, we use $\frac{a}{b^c}$ to see which level of the recursion tree bears the most weight asymptotically:

$$\frac{a}{b^c} = \frac{1}{2^0} = 1$$

Since the above calculation gives us 1, the run time for this algorithm must be the same at every level, so we have $O(logn)$ run time.

2. We know that there are at least 2 values in the dataset that are less than or equal to our sub-median and at least 4 values that are greater than or equal to our sub-median. To find α such that the $\alpha\cdot$ nth smallest of the $\frac{n}{5}$ sub-medians will still yield a linear time selection algorithm, we find a value of α that will minimize the discrepancy in the number of values on either side of the sub-median:

$$2\alpha n = 4 \cdot (\frac{n}{5} - \alpha n)$$
$$\alpha n = 2 \cdot (\frac{n}{5} - \alpha n)$$
$$\alpha n = \frac{2n}{5} - 2\alpha n$$
$$\alpha n + 2\alpha n = \frac{2n}{5}$$
$$3\alpha n = \frac{2n}{5}$$
$$\alpha n = \frac{2n}{15}$$
$$\alpha = \frac{2}{15}$$

We use Master Theorem to find the resulting recurrence relation:

$$T(n) \leq T(\frac{n}{5}) + T(n - 2\alpha n) + O(n)$$

Since $\alpha = \frac{2}{15}$, we plug in α to get:

$$T(n) \leq T(\frac{n}{5}) + T(n - 2(\frac{2}{15})n) + O(n)$$
$$T(n) \leq T(\frac{n}{5}) + T(n - \frac{4}{15}) + O(n)$$
$$T(n) \leq T(\frac{n}{5}) + T(\frac{11}{15}n) + O(n)$$

In this case, $a = 1$, $b = \frac{15}{11}$, and $c = 1$ such that the recurrence is called once at every level on a data set that is at most size $\frac{11n}{15}$. The overall non-recursive work for partitioning will take $O(n^1)$ time. Therefore,

$$\frac{a}{b^c} = \frac{1}{(\frac{15}{11})^1} = \frac{11}{15}$$

Since $\frac{a}{b^c} < \frac{11}{15} < 1$, the run time is $O(n)$

3. Assuming A and B are sorted arrays of length m and n respectively where $m > n$, we can determine if there is any element that appears in both A and B such that $A[i] = B[j]$ using the follow algorithms. To compare the lengths asymptotically, we can treat m as a function of n.

   (a) We initialize $i = 0$ and $j = 0$ to keep track of the index of A and B respectively. Then, we set up a while loop to iterate over the length of the shorter array B, breaking when $j > n$ to return `false`. From here, we get three cases:

   Case 1: $A[i] = B[j]$

   Since the value stored at $A[i]$ matches the value stored at $B[j]$, we return `true` because there exists some element that appears in both A and B.

   Case 2: $A[i] < B[j]$

   Since the value stored at $A[i]$ is less than the value stored at $B[j]$ and we know that both A and B are sorted, then all values to the left of $A[i]$ will be less than $B[j]$, so we increment i by one so we can compare the next biggest value in A to $B[j]$.

   Case 3: $A[i] > B[j]$

   Since the value stored at $A[i]$ is greater than the value stored at $B[j]$ and we know that both A and B are sorted, then all values to the left of $B[j]$ will be less than $A[i]$, so we increment j by one so we can compare the next biggest value in B to $A[i]$.

   (b) We traverse over array B and perform a binary search for $B[j]$ in A since binary search takes $O(logn)$ time. From here, we get three cases:

   Case 1: $A[i] = B[j]$

   Since there exists some element in A that is the same as $B[j]$, we return `true`.

   Case 2: $A[i] \neq B[j]$

   Since there does not exist any element in A that is the same as $B[j]$, we use a recursive call to check if there is an element in A that is the same as $B[j + 1]$.

   Case 3: $j > n$

   Since we have incremented j to the point where we are attempting to access an array index beyond the end index, we know that we have iterated over all of B. We are still in the function, so for all values in B and all values in A, there is no value such that $A[i] = B[j]$, so we return `false`.