Neilesh Sambhu U90484573
10/24/2018
COT 6405 Assignment 3

1.

```
Quicksort (A, p, r)
{
    if (p < r)
    {
        q = Partition (A, p, r);
        Quicksort (A, p, q − 1);
        Quicksort (A, q + 1, r);
    }
}

Partition (A, p, r)
{
    x ← A[r]
    i ← p − 1
    for j ← p to r − 1
        if A[j] ≤ x
            i ← i + 1
            exchange A[i] ↔ A[j]
    exchange A[i + 1] ↔ A[r]
    return i + 1
}
```

Setup: two strong induction proofs (one for ascending arrays and one for descending arrays). Assume arrays contain unique values

Ascending

Base case

Consider an input array of size $n = 2$. For the example array $\{3,4\}$, the partition function chooses 4 as the pivot. The if condition will always be true because all elements before the pivot are less than or equal to the pivot, by the nature of an ascending array. The for loop exchanges 3 with itself. The exchange after the for loop exchanges 4 with itself. The Partition function returns 2 as the index. Therefore, $q = 2$. The first recursive call to $Quicksort\ (A, p, q - 1)$ will be called on $\{3\}$ (size $r - p = 2 - 1 = 1$). The second recursive call to $Quicksort\ (A, q + 1, r)$ will be an empty array because the second argument $(q + 1) = 2 + 1 = 3$ exceeds the bounds of the array, which will be caught by the $if\ (p < r)$ statement in the recursive call, preventing any subsequent function calls.

Inductive step

Suppose that for a sorted array of size k the $Partition\ (A, p, r)$ function generates an empty array and an array of size $r - p$, for all $k \leq n$. Consider an input array of size $k + 1$. The partition function chooses the $k + 1th$ element as the pivot. Because all elements from index 1 to k are less than $k + 1th$ element, the if condition will always be true. The value of i will be incremented by 1, and each element from 1 to k will be swapped with itself. The exchange after the for loop exchanges the $k + 1th$ element with itself. The Partition function returns $k + 1$ as the index. Therefore, $q = k + 1$. The first recursive call to $Quicksort\ (A, p, q - 1)$ will be called on the following array indexes: $\{1, \dots, k\}$ (size $r - p = k + 1 - 1 = k$). The second recursive call to $Quicksort\ (A, q + 1, r)$ will be an empty array because the second argument $(q + 1) = k + 1 + 1 = k + 2$ exceeds the bounds of the array, which will be caught by the $if\ (p < r)$ statement in the recursive call, preventing any subsequent function calls.

Neilesh Sambhu U90484573
10/24/2018
COT 6405 Assignment 3

Descending

Base case

Consider an input array of size $n = 2$. For the example array $\{4,3\}$, the partition function chooses 3 as the pivot. The if condition will always be false because all elements before the pivot are greater than the pivot, by the nature of an descending array. The exchange after the for loop exchanges 4 with 3. The Partition function returns 1 as the index. Therefore, $q = 1$. The first recursive call to $Quicksort\ (A, p, q-1)$ will be an empty array because the second argument $(q-1) = 1-1 = 0$ exceeds the bounds of the array, which will be caught by the $if\ (p < r)$ statement in the recursive call, preventing any subsequent function calls. The second recursive call to $Quicksort\ (A, q+1, r)$ will be called on $\{4\}$ (size $r - p = 2 - 1 = 1$).

Inductive step

Suppose that for a sorted array of size k the $Partition\ (A, p, r)$ function generates an empty array and an array of size $r - p$, for all $k \leq n$. Consider an input array of size $k+1$. The partition function chooses the $k+1th$ element as the pivot. Because all elements from index 1 to k are greater than $k+1th$ element, the if condition will always be false. The value of i will remain as one less than the index of the first element, and no swapping will occur. The exchange after the for loop exchanges the $k+1th$ element with the first element. The Partition function returns 1 as the index. Therefore, $q = 1$. The first recursive call to $Quicksort\ (A, p, q-1)$ will be an empty array because the second argument $(q-1) = 1-1 = 0$ exceeds the bounds of the array, which will be caught by the $if\ (p < r)$ statement in the recursive call, preventing any subsequent function calls. The second recursive call to $Quicksort\ (A, q+1, r)$ will be called on the following array indexes: $\{2, \ldots, k+1\}$ (size $r - p = k + 1 - 1 = k$).

2. No. Given an array in which all values are the same, each call to the partition function is only able to reduce the array by 1. Therefore, n calls are still required, and each call costs $\theta(n)$.
3.
    1.
    $q1 = i + 1$
    $for\ j = q1 + 1\ to\ r - 1$
    $\quad if\ A[j] \leq x$
    $\quad\quad exchange\ A[i] \leftrightarrow A[j]$
    $exchange\ A[i+1] \leftrightarrow A[r]$
    (Assume that the exchange method will check the bounds of the array and NOT swap with a memory location outside the bounds of the array.)
    $q2 = i + 1$
    2. Yes, it is possible for line 1 and line 9 of Triple-PARTITION to choose the same element as the pivot. In the case of a sorted ascending array, the exchange in line 7 will exchange the last element with itself (see question #1). Because r does not point to different element, line 9 will choose the same pivot as line 1.
    No, is it not possible for $q1 = q2$. On line 8, $q1$ is initialized to $i + 1$. On line 10, i is set to $q1$. The for loop in lines 11-14 will only ever increment the value of i. If the if condition in line 12 is always false, i will not be incremented. Regardless, $q2$ will be set to $i + 1 \geq q1$.
    3. In worst-case, Triple-PARTITION reduces the size of the array by only 1. This produces the maximum number of function calls. (Reducing the size of the array by any other factor will reduce the number of function calls and improve the functional complexity of the algorithm.) Therefore, one of the calls to Triple-QUICKSORT will be on an array of size $n - 1$. The second call to Triple-QUICKSORT will be on an array of size 0. The third call to Triple-QUICKSORT will be on an array of size 0. Note: if $q1$ is the last

element in the array, $q2$ will be outside the bounds of the array, which is why the first call to Triple-QUICKSORT gets an array of size $n-1$ instead of $n-2$. The worst-case complexity of the Triple-PARTITION takes $\theta(2n) = \theta(n)$. The for loop in lines 3-6 will iterate $\theta(n)$ times, with the body of loop taking $\theta(1)$ time. The for loops in lines 11-14 will iterate $\theta(n)$ times, with the body of loop taking $\theta(1)$ time. All other operations in Triple-PARTITION take $\theta(1)$ time. Therefore, Triple-PARTITION follows the following recurrence:

$T(n) = T(n-1) + T(1) + T(1) + \theta(2n)$
$T(n) = T(n-1) + \theta(1) + \theta(1) + \theta(2n)$
$T(n) = T(n-1) + \theta(1) + \theta(1) + \theta(n)$
$T(n) = T(n-1) + \theta(n)$

It takes n iterations for n to equal 0. The cost of each iteration is $\theta(n)$. Therefore, the complexity is $\theta(n^2)$.

4. Proof by strong induction

   Base case

   For an array composed only of elements where $d = 1$. StableSort on digit 1 will trivially sort the array with one iteration of the for loop because the digit lengths are 1.

   Inductive step

   Suppose radix sort correctly sorts arrays with d digits, for all $d \leq n$. For an array with values that are $n+1$ digits, the $n+1th$ iteration of the for loop will sort the array entirely. If two digits in the $n+1th$ position are different, the StableSort will trivially ensure that the values are sorted properly because the lower order digits are irrelevant. If the two digits in the $n+1th$ position are the same, the StableSort will ensure that the values are sorted properly because the sorting order of the lesser significant digits will be preserved.

5. $n = 80{,}000{,}000$

   $b = 64, each\ number\ is\ 64\ bits$

   $$T(n) = \theta\left(\frac{b}{r} * (n + 2^r)\right)$$

   1. $r = 32$, 32-bit digits

      $$T(n) = \frac{64}{32} * (80{,}000{,}000 + 2^{32}) = 2(80{,}000{,}000 + 2^{32}) = 8.749934592 \times 10^9$$

   2. $r = 16$, 16-bit digits

      $$T(n) = \frac{64}{16} * (80{,}000{,}000 + 2^{16}) = 4(80{,}000{,}000 + 2^{16}) = 3.20262144 \times 10^8$$

   3. $r = 8$, 8-bit digits

      $$T(n) = \frac{64}{8} * (80{,}000{,}000 + 2^{8}) = 8(80{,}000{,}000 + 2^{8}) = 6.40002048 \times 10^8$$

   4. $r = \lceil \lg(n) \rceil$, $\lceil \lg(n) \rceil$-bit digits

      $$T(n) = \frac{64}{\lceil \lg(80{,}000{,}000) \rceil} * \left(80{,}000{,}000 + 2^{\lceil \lg(80{,}000{,}000) \rceil}\right) = 5.07775355 \times 10^8$$

   5. $r = \lceil \lg(n) \rceil$, $\lceil \lg(n) \rceil$-bit digits

      $$T(n) = \frac{64}{\lceil \lg(80{,}000{,}000) \rceil} * \left(80{,}000{,}000 + 2^{\lceil \lg(80{,}000{,}000) \rceil}\right) = 3.62114126 \times 10^8$$

   The running time is at a minimum for $r = 16$ and at a maximum for $r = 32$. There must be a balance in the value of r to achieve a minimum running time. For $r\ much\ less\ than\ 16$, the $\frac{b}{r}$ term increases the runtime. For $r\ much\ greater\ than\ 16$, the 2^r term increases the runtime.

6. $array\ range\ of\ values = [0, k * n^{100} - 1]$

Neilesh Sambhu U90484573
10/24/2018
COT 6405 Assignment 3

$k < n$

$Let\ b = \lfloor \log(k * n^{100} - 1) + 1 \rfloor$

The logarithm (base 10) of a number is equal to the number of digits needed to represent the number in decimal. However, we must also consider numbers that are exactly powers of 10, where the output of the logarithm will be one less than the number of bits needed to represent the number. This is why we need to add one and floor the final result.

$Let\ r = \lg(n)$

$T(n) = \frac{b}{r} * (n + 2^r)$

$T(n) = \frac{\lfloor \log(k*n^{100}-1)+1 \rfloor}{\lg(n)} * \left(n + 2^{\lg(n)}\right)$ by substitution

$T(n) = \frac{\lfloor \log(k*n^{100}-1)+1 \rfloor}{\lg(n)} * (n + n)$ by log identities from the introduction slides

$T(n) = \frac{\lfloor \log(k * n^{100} - 1) + 1 \rfloor}{\lg(n)} * (2n)$

$T(n) \leq \frac{\log(k*n^{100}-1)+1}{\lg(n)} * (2n)$ by the fact that floor function will always make a number smaller or equal to itself but never larger

$T(n) \leq \frac{\log(k*n^{100})+1}{\lg(n)} * (2n)$ by the fact that removing a negative value from the input to the logarithm will only increase the output.

$$T(n) \leq \left(\frac{\log(k * n^{100})}{\lg(n)} + \frac{1}{\lg(n)}\right) * (2n)$$

$$\lim_{n\to\infty} \left(\frac{1}{\lg(n)}\right) = 0$$

$T(n) \leq \left(\frac{\log(k*n^{100})}{\lg(n)} + 0\right) * (2n)$ by substitution from the aforementioned limit

$$T(n) \leq \left(\frac{\log(k * n^{100})}{\lg(n)}\right) * (2n)$$

$T(n) \leq \left(\frac{\lg(k*n^{100})}{\lg(n)}\right) * (2n)$ by the fact that reducing the base of a logarithm will only ever increase the output.

Assume $k * n^{100} \geq 1$

$T(n) \leq \left(\frac{\lg(k)+\lg(n^{100})}{\lg(n)}\right) * (2n)$ by log identities from the introduction slides

$$T(n) \leq \left(\frac{\lg(k)}{\lg(n)} + \frac{\lg(n^{100})}{\lg(n)}\right) * (2n)$$

$T(n) \leq \left(\frac{\lg(k)}{\lg(n)} + \frac{100*\lg(n)}{\lg(n)}\right) * (2n)$ by log identities from the introduction slides

$$T(n) \leq \left(\frac{\lg(k)}{\lg(n)} + 100\right) * (2n)$$

$\lim_{n\to\infty} \left(\frac{\lg(k)}{\lg(n)}\right) = 0$ because k is a constant less than n

$T(n) \leq (0 + 100) * (2n)$ by substitution from the aforementioned limit

$T(n) \leq 200n = \theta(n)$

Because radix sort can sort this array in $\theta(n)$, we can sort these numbers in $O(n)$ time.

Neilesh Sambhu U90484573
10/24/2018
COT 6405 Assignment 3

7. $n\ integers$
   $range = [0, k]$

   $PreprocessingAlgorithm(A, k)$:
   $C = Array(k)$ // initialize array C to have size k
   $for\ i = 0\ to\ k$
   $C[i] = 0;$
   $for\ j = 1\ to\ A.length$
   $C[A[j]]\ += 1;$
   $for\ i = 1\ to\ k$
   $C[i]\ = C[i] + C[i-1];$
   $return\ C$

   $QueryAlgorithm(C, a, b)$:
   $return\ C[b] - C[a]$

   The PreprocessingAlgorithm originates from lines 1-7 of CountingSort. The last for loop in lines 8-10 of CountingSort is omitted from PreprocessingAlgorithm. CountingSort is $O(n+k)$, and PreprocessingAlgorithm performs less work than CountingSort. Therefore, PreprocessingAlgorithm is also $O(n+k)$. QueryAlgorithm is 2 simple memory lookups and a subtraction operation. Therefore, QueryAlgorithm is $O(1)$.

8. $Let\ i = 1$
   Elements in red text indicate the choice of pivot.

Iteration	q	Partitions
1	9	{3,2,0,7,5,4,8,6,1,9}
2	8	{3,2,0,7,5,4,6,1,8}
3	7	{3,2,0,5,4,6,1,7}
4	6	{3,2,0,5,4,1,6}
5	5	{3,2,0,4,1,5}
6	4	{3,2,0,1,4}
7	3	{2,0,1,3}
8	2	{0,1,2}
9	1	{0,1}
10	0	{0}

   When choosing the largest element for the partition, we only reduce the size of the search space in the array by 1 for each call to the RandomizedSelect function. This will generate the maximum number of function calls for the algorithm to terminate, maximizing the runtime to $O(n^2)$. There are n calls to the RandomizedPartition function, and each call takes $O(n)$ time.

9. Assume there are no duplicate values in the array.
   $n \geq 140$
   $\left\lceil \frac{n}{4} \right\rceil \geq \frac{140}{4}$
   $\left\lceil \frac{n}{4} \right\rceil \geq 35$
   There are at least $\frac{140}{5} = 28$ groups.
   When SELECT partitions the $\frac{n}{5}$ groups around the median of median (called x), each group whose median is greater than x (half the groups) contains 2 elements whose values are greater than x. Moreover, the median of

those groups is also greater than x. Therefore, each of those groups of 5 elements contributes 3 elements whose values are greater than x.

$\frac{1}{2} * \frac{3}{5} * n = \frac{3}{10}n$

$\frac{3}{10}n \geq \left\lceil \frac{n}{4} \right\rceil \geq \frac{n}{4}$

$\frac{3}{10}n \geq \frac{n}{4}$

$\frac{3}{10}(140) \geq \frac{(140)}{4} + 1$

$42 \geq 36$

The coefficient $\frac{3}{10}$ dominates the coefficient $\frac{1}{4}$ and constant of 1.

When SELECT partitions the $\frac{n}{5}$ groups around the median of median (called x), each group whose median is less than x (half the groups) contains 2 elements whose values are less than x. Moreover, the median of those groups is also less than x. Therefore, each of those groups of 5 elements contributes 3 elements whose values are less than x.

$\frac{1}{2} * \frac{3}{5} * n = \frac{3}{10}n$

$\frac{3}{10}n \geq \left\lceil \frac{n}{4} \right\rceil \geq \frac{n}{4}$

$\frac{3}{10}n \geq \frac{n}{4}$

$\frac{3}{10}(140) \geq \frac{(140)}{4} + 1$

$42 \geq 36$

The coefficient $\frac{3}{10}$ dominates the coefficient $\frac{1}{4}$ and constant of 1.

10. Assume the array contains an odd number of elements. Assume that "closest" is defined as the count of elements between a given number and the median, if the elements are sorted. Assume that "closest" does not refer to a difference in the value of elements. Assume that the median of S is the first closest element to the median of S.

$kClosestToMedian(A, p, r, k)$:

$upper = Select\left(A, p, r, \left\lceil \frac{n}{2} \right\rceil + \left\lceil \frac{k}{2} \right\rceil\right)$

$lower = Select\left(A, p, r - \left(n - \left(\left\lceil \frac{n}{2} \right\rceil + \left\lceil \frac{k}{2} \right\rceil\right)\right), \left\lceil \frac{n}{2} \right\rceil - \left\lceil \frac{k}{2} \right\rceil\right)$ // we don't want to change the ordering of the elements in the upper part of the array beyond $\left\lceil \frac{n}{2} \right\rceil + \left\lceil \frac{k}{2} \right\rceil$

$upper\_index = 1$

$lower\_index = 1$

$for\ i = 1\ to\ n$

    $if\ A[i] = upper$

        $upper\_index = i$

    $if\ A[i] = lower$

        $lower\_index = i$

$return\ A[lower\_index \ldots upper\_index]$

The two calls to Select are $2 * \theta(n) = \theta(2n) = \theta(n)$. Assigning the values for the upper and lower indexes take $\theta(1)$. Iterating through the for loop takes $\theta(n)$ time because there are $\theta(n)$ iterations and the body of the for

Neilesh Sambhu U90484573
10/24/2018
COT 6405 Assignment 3

loop consists of operations that are all $\theta(1)$. Returning the segment of the array takes $\theta(1)$. The total complexity is $\theta(n) + \theta(1) + \theta(n) + \theta(1) = \theta(n)$.