Name: Jaryd Meek
ID:
Collaborators: Noah Nguyen, Emily Parker

CSCI 3104, Algorithms **Due Feb 19, 2021**
Problem Set 05 (50 points) **Spring 2021, CU-Boulder**

Advice 1: For every problem in this class, you must justify your answer: show how you arrived at it and why it is correct. If there are assumptions you need to make along the way, state those clearly.
Advice 2: Verbal reasoning is typically insufficient for full credit. Instead, write a logical argument, in the style of a mathematical proof.

Instructions for submitting your solution:

- The solutions **should be typed** and we cannot accept hand-written solutions. Here's a short intro to Latex.
- You should submit your work through **Gradescope** only.
- The easiest way to access Gradescope is through our Canvas page. There is a Gradescope button in the left menu.
- Gradescope will only accept **.pdf** files.
- It is vital that you match each problem part with your work. Skip to 1:40 to just see the matching info.

Name: Jaryd Meek
ID: ☐
Collaborators: Noah Nguyen, Emily Parker

CSCI 3104, Algorithms **Due Feb 19, 2021**
Problem Set 05 (50 points) **Spring 2021, CU-Boulder**

1. *(16 pts) Suppose in quicksort, we have access to an algorithm which chooses a pivot such that, the ratio of the size of the two subarrays divided by the pivot is a* ***constant*** *k. i.e an array of size n is divided into two arrays, the first array is of size $n_1 = \frac{nk}{k+1}$ and the second array is of size $n_2 = \frac{n}{k+1}$ so that the ratio $\frac{n_1}{n_2} = k$ a constant.*

   (a) *(3 pts) Given an array, what value of k will result in the best partitioning?*

   (b) *(10 pts) Write down a recurrence relation for this version of QuickSort, and solve it asymptotically using* ***recursion tree*** *method to come up with a big-O notation. For this part of the question assume $k = 3$. Show your work, write down the first few levels of the tree, identify the pattern and solve. Assume that the time it takes to find the pivot is $\Theta(n)$ for lists of length n. Note: Remember that a big-O bound is just an upper bound. So come up with an expression and make arguments based on the big-O notation definition.*

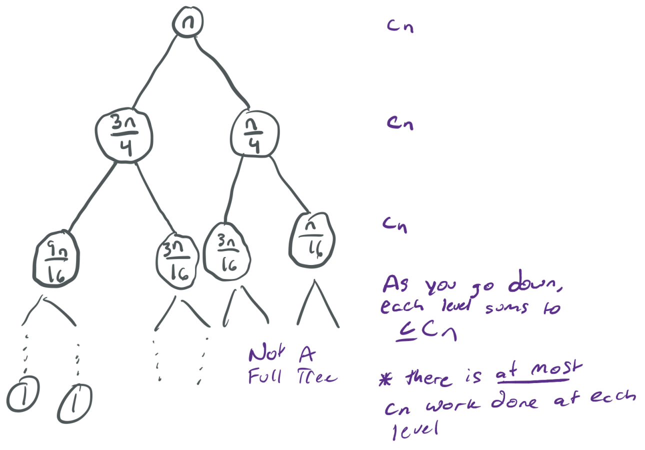
   (c) *(3 pts) Does the value of k affect the running time?*

Solution:

(a) Best partitioning will occur when k is 1. This will result with the in the array being split into two equal subarrays.

$$\text{when } k = 1,\ n_1 = \frac{n}{2} \text{ and } n_2 = \frac{n}{2}$$

(b)


n
3n/4
n/4
9n/16
3n/16
3n/16
n/16
1
1
cn
cn
cn
As you go down, each level sums to ≤ cn
Not A Full Tree
* there is at most cn work done at each level

$$\text{Recurrence Relation - } T(n) = T(\frac{3n}{4}) + T(\frac{n}{4}) + O(n)$$

How many levels? - Longest path is when $\left\lceil \frac{n}{(\frac{4}{3})^k} \right\rceil \leq 1$ (k = number of levels)

$$n \leq (4/3)^k$$

$$\log_{4/3} n \leq k$$

$$\log_{4/3} n + \epsilon = k \text{ where } 0 \leq \epsilon < 1$$

$$\text{number of levels - } \approx \log_{4/3} n$$

$$\text{Total work done} \leq cn \log_{4/3} n$$

$$\text{Therefore } O(n \lg n)$$

(c) K won't affect the big O time, but it will increase the average running time.

2. *(10 pts) Consider a chaining hash table A with b slots that holds data from a fixed, finite universe U.*

   (a) *(3 pts) State the simple uniform hashing assumption.*

   (b) *(7 pts) Consider the worst case analysis of hash tables. Suppose we start with an empty hash table, A. A* ***collision*** *occurs when an element is hashed into a slot where there is another element already. Assume that $|U|$ represents the size of the universe and b represents the number of slots in the hash table. Let us assume that $|U| \leq b$. Suppose we intend to insert n elements into A* ***Do not assume the simple uniform hashing assumption for this subproblem.***

      i. *What is the worst case for the number of collisions? Express your answer in terms of n.*

      ii. *What is the load factor for A in the previous question?*

      iii. *How long will a successful search take, on average? Give a big-Theta bound.*

Solution:

(a) The simple uniform hashing assumption assumes that each of the slots in the hash table have an equal probability of being selected.

(b) i. $\boxed{n-1}$ the first item will not collide, but every one after will collide.

   ii. The load factor (α) is defined to be $\frac{\text{number of items}}{\text{number of slots in table}}$, so in this case since we are talking about the worst case scenerio, there is only one slot in the table. Therefore, the load factor will be $\frac{n}{1} = \alpha = n$

   iii. For hash tables with chaining, a successful search will take on average $\Theta(1+\alpha)$ where $\alpha = \frac{\text{number of items}}{\text{number of slots in table}}$ So, the average successful search will be $\Theta\left(1 + \frac{\text{number of items}}{\text{number of slots in table}}\right)$. This is proved in Prof. Cox's Lecture 9 - page 12, so I'm just working from that.

3. *(12 pts) Consider a hash table of size 100 with slots from 1 to 100. Consider the hash function* $h(k) = \lfloor 100k \rfloor$ *for all keys* k *for a table of size 100. You have three applications.*

   - ***Application 1****: Keys are generated uniformly at random from the interval* $[0.3, 0.8]$.
   - ***Application 2****: Keys are generated uniformly at random from the interval* $[0.1, 0.4] \cup [0.6, 0.9]$.
   - ***Application 3****: Keys are generated uniformly at random from the interval* $[0, 1]$.

   (a) *(3 pts) Suppose you have* n *keys in total chosen for each application. What is the resulting load factor* α *for each application?*

   (b) *(3 pts) Which application will yield the worst performance?*

   (c) *(3 pts) Which application will yield the best performance?*

   (d) *(3 pts) Which application will allow the uniform hashing property to apply?*

Solution:

(a) **Application 1 -** $\boxed{\alpha = n/50}$
Application 2 - $\boxed{\alpha = n/60}$
Application 2 - $\boxed{\alpha = n/100}$

(b) Application 1 will yield the worst performance since it has the smallest number of bins, leading to the largest load factor.

(c) Application 3 will yield the best performance since it has the largest number of bins, leading to the smallest load factor.

(d) Application 3 will allow the uniform hashing property to apply since it has 100 numbers, and 100 bins, all equally likely to be selected.

Name: $\boxed{\text{Jaryd Meek}}$
ID: $\boxed{}$
Collaborators: Noah Nguyen, Emily Parker

CSCI 3104, Algorithms **Due Feb 19, 2021**
Problem Set 05 (50 points) **Spring 2021, CU-Boulder**

4. *(12 pts) Median of Medians Algorithm*

   (a) *(4 pts) Illustrate how to apply the QuickSelect algorithm to find the $k = 4$th smallest element in the given array:* `A = [5, 3, 4, 9, 2, 8, 1, 7, 6]` *by showing the recursion call tree. Refer to Sam's Lecture 10 for notes on QuickSelect algorithm works*

   (b) *(4 pt) Explain in 2-3 sentences the purpose of the Median of Medians algorithm.*

   (c) *(4 pts)Consider applying Median of Medians algorithm (A Deterministic QuickSelect algorithm) to find the 4th largest element in the following array:* `A = [6, 10, 80, 18, 20, 82, 33, 35, 0, 31, 99, 22, 56, 3, 32, 73, 85, 29, 60, 68, 99, 23, 57, 72, 25]`*.Illustrate how the algorithm would work for the first two recursive calls and indicate which sub array would the algorithm continue searching following the second recursion. Refer to Rachel's Lecture 8 for notes on Median of Medians Algorithm*

Solution:

(a)

NOTE - I'm using Sam's pseudocode, which would mean that for a k giving the 4th smallest element, k = 3. I'm 0 indexing it.

Starting Array - `A = [5, 3, 4, 9, 2, 8, 1, 7, 6]`, $k = 3$
Randomly Select a Pivot (using Siri to pick a random index 0-8) index = 1,
pivot = A[1] = 3
Sort the array into two arrays, one of lower numbers, one of higher numbers -
`Lower = [2, 1]`, `Higher = [5, 4, 9, 8, 7, 6]`
Since $k = 3$, we enter the else of the if statement, therefore we recurse with the higher array, and $k = k - 1 - \text{length(lower)} = k = 0$

<u>Recurse</u>
Array - `A = [5, 4, 9, 8, 7, 6]`, $k = 0$
Randomly Select a Pivot (using Siri to pick a random index 0-5) index = 0,
pivot = A[0] = 5
Sort the array into two arrays, one of lower numbers, one of higher numbers -
`Lower = [4]`, `Higher = [9, 8, 7, 6]`
Since $k = 0$, we enter the first if statement, therefore we recurse with the lower array, and same $k = 0$

<u>Recurse</u>
Array - `A = [4]`, $k = 0$
Since we have 1 item in the array, we return the item, returning 4

$\boxed{\text{4th Smallest Number} = 4 \text{ according to QuickSelect}}$

Therefore the function returns 4 which does confirm to be the 4th smallest element in the array once sorted by hand.

B and C on next page

(b) The purpose of the Median of Medians algorithm is to select a better pivot than a random selection. By picking a pivot that is closer to the median of your data, you can divide the array into more even parts, causing there to be less recursions, making the algorithm faster. So by picking the median of medians, you may not be picking the actual median, but you will be picking something close, which can greatly decrease your running time.

(c)

<u>Median of Medians -</u>
Starting array -
`A = [6, 10, 80, 18, 20, 82, 33, 35, 0, 31, 99, 22, 56, 3, 32, 73, 85, 29, 60, 68, 99, 23, 57, 72, 25]`

Divide array into sub arrays of size 5 (plus one array with remainders if any exist)
`A = [6, 10, 80, 18, 20 | 82, 33, 35, 0, 31 | 99, 22, 56, 3, 32 | 73, 85, 29, 60, 68 | 99, 23, 57, 72, 25]`

Sort Each subarray (Sorting is $\Theta(n)$)
`A = [6, 10, 18, 20, 80 | 0, 31, 33, 35, 82 | 3, 22, 32, 56, 99 | 29, 60, 68, 73, 85 | 23, 25, 57, 72, 99]`

Take the median of each subarray
`A = [6, 10,` <u>`18`</u>`, 20, 80 | 0, 31,` <u>`33`</u>`, 35, 82 | 3, 22,` <u>`32`</u>`, 56, 99 | 29, 60,` <u>`68`</u>`, 73, 85 | 23, 25,` <u>`57`</u>`, 72, 99]`
Medians = `[18, 33, 32, 68, 57]`

Sort that subarray
Medians = `[18, 32, 33, 57, 68]`

Take the median of medians
Medians = `[18, 32,` <u>`33`</u>`, 57, 68]`

$\boxed{\text{Median of Medians} = 33}$

<u>Now use that value as a pivot in our quicksort algorithm</u>
Starting array -
`A = [6, 10, 80, 18, 20, 82, 33, 35, 0, 31, 99, 22, 56, 3, 32, 73, 85, 29, 60, 68, 99, 23, 57, 72, 25]`
k = 4th largest element = $k = 21$
Pivot determined by Median of Medians = 33
Sort the array into two arrays, one of lower numbers, one of higher numbers -
`Lower = [6, 10, 18, 20, 0, 31, 22, 3, 32, 29, 23, 25], Higher = [80, 82, 35, 99, 56, 73, 85, 60, 68, 99, 57, 72]`
Since $k = 21$, we enter the else of the if statement, therefore we recurse with the higher array, and $k = k - 1 - \text{length(lower)} = k = 21 - 1 - 12 = 8$

Continued on Next Page

Recurse

Run Median of Medians on the new array -
Array - `A = [80, 82, 35, 99, 56, 73, 85, 60, 68, 99, 57, 72]`

Divide array into sub arrays of size 5 (plus one array with remainders if any exist)
`A = [80, 82, 35, 99, 56 | 73, 85, 60, 68, 99 | 57, 72]`

Sort Each subarray (Sorting is $\Theta(n)$)
`A = [35, 56, 80, 82, 99 | 60, 68, 73, 85, 99 | 57, 72]`

Take the median of each subarray
A = [35, 56, $\underline{80}$, 82, 99 | 60, 68, $\underline{73}$, 85, 99 | 57, $\underline{72}$]
Medians = `[80, 73, 72]`

Sort that subarray
Medians = `[72, 73, 80]`

Take the median of medians
Medians = [72, $\underline{73}$, 80]

$\boxed{\text{Median of Medians} = 73}$

Now use that value as a pivot in our quicksort algorithm
Starting array -
Array - `A = [80, 82, 35, 99, 56, 73, 85, 60, 68, 99, 57, 72]`, $k = 8$
Pivot determined by Median of Medians = 73
Sort the array into two arrays, one of lower numbers, one of higher numbers -
`Lower = [35, 56, 60, 68, 57, 72]`, `Higher = [80, 82, 99, 85, 99]`
Since $k = 8$, we enter the else of the if statement, therefore we recurse with the higher array, and
$k = k - 1 - \text{length(lower)} = k = 8 - 1 - 6 = 1$

Recurse

Run Median of Medians on the new array -
Array - `A = [80, 82, 99, 85, 99]`
There's only 5 items, so sort array and find median

Sort Array -
`A = [80, 82, 85, 99, 99]`

Find Median -
A = [80, 82, $\underline{85}$, 99, 99]

$\boxed{\text{Since there is only one median, our median of medians is 85.}}$

Continued on Next Page

Now use that value as a pivot in our quicksort algorithm

Starting array -

Array - `A = [80, 82, 99, 85, 99]`

Pivot determined by Median of Medians = 85

Sort the array into two arrays, one of lower numbers, one of higher numbers -

`Lower = [80, 82], Higher = [99, 99]]`

Since $k = 1$ we enter the first if of the if statement, and would recurse with the lower array and same k value.

Example Call - `quickSelect([80, 82], 1)`