Name: YOUR NAME HERE

ID: YOUR STUDENT ID HERE

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: YOUR NAME HERE
ID: YOUR STUDENT ID HERE

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.*

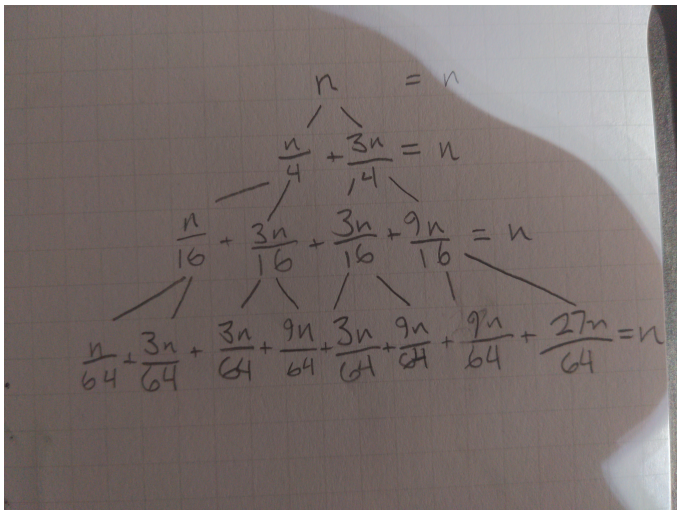
   (c) *(3 pts) Does the value of k affect the running time?*

Solution:
graphicx

(a) $k = 1$ will give the best partitioning, since it splits the problem into two equal subproblems, which is the best possible case because it creates a balanced subtree with the fewest possible calls.

(b)

$$T(n) = \begin{cases} 1 & n = 1 \\ T(\frac{n}{4}) + T(\frac{3n}{4}) + n & n > 1 \end{cases}$$



$Q(n) \exists O(nlog(n))$

(c) k does not affect the running time, since no matter what k is chosen, each level of recursion will still sum to a problem of size n, and will still be bound by $nlog(n)$.

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) When applying the SUHA, we assume that all items have an equal chance of occupying any given space, and that items will be evenly distributed in the hash table.

(b) i. For the first item, there are no items in the table yet for it to collide with. Every subsequent item may collide with the first, resulting in $n - 1$ total collisions worst case.

ii. The load factor is $\frac{n}{b}$

iii. On average, a search will take $\frac{n}{2} \in \Theta(n)$ steps, given a worst case load for our hash table.

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) *(3 pts) Suppose you have* n *keys in total chosen for each application. What is the resulting load factor* α *for each application?* For all applications the load factor will be $\frac{n}{100}$

(b) *(3 pts) Which application will yield the worst performance?* Application 1 has the worst performance, since only half of all slots may be filled, where application 2 may fill 60% of slots, and application 3 may fill 100% of slots.

(c) *(3 pts) Which application will yield the best performance?* Application 3 has the best performance, since keys are spread evenly across the entire table, rather than a limited section of the table.

(d) *(3 pts) Which application will allow the uniform hashing property to apply?* With application 3, all slots are guaranteed to have equal chance of being filled, since our keys are also distributed evenly across the entire input.

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

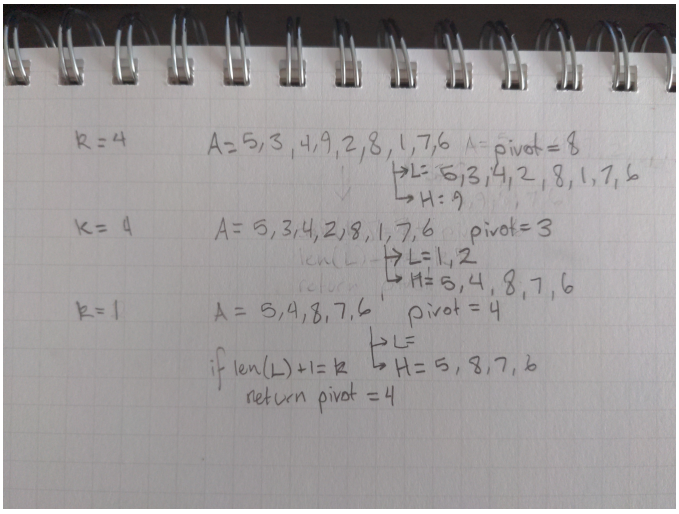
   (a) *(4 pts) Illustrate how to apply the QuickSelect algorithm to find the $k = 4$th smallest element in the given array:* `A = [5, 3, 4, 9, 2, 8, 1, 7, 6]` *by showing the recursion call tree. Refer to Sam's Lecture 10 for notes on QuickSelect algorithm works*

   (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) Pivots were randomly chosen using Python's *random.choice*() function.



(b) The median of medians algorithm is used to approximate the median of an array, since it is far more expensive to find the median for an unsorted dataset, we instead break the array into several parts (generally 5 item chunks), and find the median of the set of medians of each of those groups. That allows us to do a much smaller problem to get an approximately optimal median for many datasets.

(c) The algorithm continues on the items lower than the pivot value. For sets of even length the smaller of the two middle values is chosen.