

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

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:**

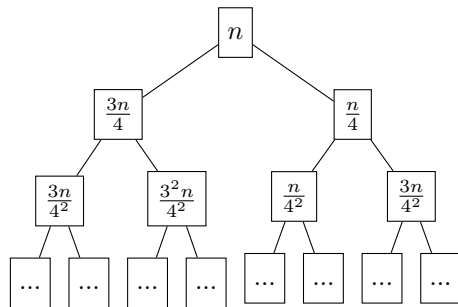
- (a) The best possible partitioning would be if the array were divided into two subarrays of the same length. Meaning  $n_1 = n_2$  and therefore  $\frac{n_1}{n_2} = k = 1$ .

**The value of  $k$  that would result in the best partitioning is  $k = 1$ .**

- (b) Recurrence relation for this version of QuickSort:  $T(n) = T(\frac{nk}{k+1}) + T(\frac{n}{k+1}) + Cn$

If  $k = 3$

Recursion Tree:



$$\text{For } k_1 : \frac{n}{\frac{4}{3}k_1} = 1 \rightarrow \frac{4}{3}k_1 = n \rightarrow k_1 = \log_{\frac{4}{3}}(n)$$

$$\text{For } k_2 : \frac{n}{4^{k_2}} = 1 \rightarrow 4^{k_2} = n \rightarrow k_2 = \log_4(n)$$

$$k_1 = \log_{\frac{4}{3}}(n)$$

$$k_2 = \log_4(n)$$

Since we have that  $k_1 > k_2$

$$\therefore T(n) = O(n \log_{\frac{4}{3}})$$

- (c) The value of  $k$  does not affect the running time because for any value of  $k$ , it will be a constant. Therefore, the function will be always asymptotically equal for any value of  $k$ .

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) Simple uniform hashing assumption states that each of the hash slots has the same probability of being chosen. That is, there is a hash function that will uniformly distribute all values into the hash table slots.
- (b)
  - i. The worst case is if each of the  $n$  items are assigned to the same bucket, meaning there are  $n$  collisions and search takes  $\Theta(n)$  time.
  - ii. The load factor of a hash table is  $(\text{number of elements})/(\text{number of slots}) \therefore$  the load factor for  $A$  is  $\frac{n}{b}$ .
  - iii. A successful search takes time  $\Theta(1 + \alpha)$ , on average.

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  $\alpha$  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:**

The keys can only be within the 30 to 80 interval, meaning there are 50 possible keys on this application

$$\therefore \alpha = \frac{n}{50}$$

**Application 2:**

The keys can only be within the 10 to 40 and 60 to 90 interval, meaning there are 60 possible keys on this application

$$\therefore \alpha = \frac{n}{62}$$

**Application 3:**

The keys can be within the 0 to 100 interval, meaning there are 100 possible keys on this application

$$\therefore \alpha = \frac{n}{100}$$

- (b) Application 1 will yield to the worst performance because it has the smallest range for key generation and the largest load factor.
- (c) Application 1 will yield to the best performance because it has the largest range for key generation and the smallest load factor.
- (d) Application 3 will allow the uniform hashing property to be applied because the key generation interval allows for every slots in the hashtable to have an equal chance to be picked.

## 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) Let  $k$  be the index to be searched on the in the sorted array. Also, let pivot be the last element of the given array and let  $p$  represent the Pivot Index.

Array:  $A = [5, 3, 4, 9, 2, 8, 1, 7, 6]$

Left Subarray

Right Subarray

$k = 3$  and  $p = 5$  Move to left sub tree  $k = 3$

$LeftA = [3, 4, 2, 1]$

Using a 5 as the pivot

$RightA = [9, 7, 8]$

$k = 3$  and  $p = 0$  Move to right sub tree  $k = k - 1 = 2$

$LeftA = []$

Using a 1 as the pivot

$RightA = [3, 4, 2, 5]$

$k = 2$  and  $p = 4$  Move to left sub tree  $k = 2$

$LeftA = [1, 3, 2, 4]$

Using a 5 as the pivot

$RightA = []$

$k = 2$  and  $p = 1$  Move to right sub tree  $k = k - 1 = 1$

$LeftA = [1]$

Using a 2 as the pivot

$RightA = [4, 3]$

$k = 1$  and  $p = 0$  Move to right sub tree  $k = k - 1 = 0$

$LeftA = []$

Pick a 3 as the pivot

$RightA = [4]$

$k = 0$  and  $p = 0$  Return the pivot value. (K+1)th min = 4

$LeftA = []$

Pick a 4 as the pivot

$RightA = []$

Thus 4 is the 4<sup>th</sup> smallest element in the given array.

- (b) The purpose of the median of medians algorithm is to find a pivot for the sorting algorithm that is not too far from the true median. By doing so, it reduces the worst-case complexity of a quickselect algorithm and builds an asymptotically optimal sorting algorithm.

(c) First step is to divide the array into groups of five elements:

$$A_1 = [6, 10, 80, 18, 20], A_2 = [82, 33, 35, 0, 31], A_3 = [99, 22, 56, 3, 32], A_4 = [73, 85, 29, 60, 68], A_5 = [99, 23, 57, 72, 25]$$

Then, sort each of the five groups:

$$A_1 = [6, 10, 18, 20, 80] A_2 = [0, 31, 33, 35, 82] A_3 = [3, 22, 32, 56, 99] A_4 = [29, 60, 68, 73, 85] A_5 = [23, 25, 57, 72, 99]$$

Take the median of each group and put these values into a list  $M$ .

$$M = [18, 33, 32, 68, 57]$$

Sort the list  $M$ :

$$M = [18, 32, 33, 57, 68]$$

Choose the median point of the  $M$  list as the pivot:

$$P = 33$$

Place all the elements that are smaller than  $P$  on the left and all elements that are greater than  $P$  on the right. We then get a list  $A$ :

$$A = [6, 10, 18, 20, 0, 31, 3, 22, 32, 29, 23, 25, 33, 80, 35, 82, 56, 99, 60, 68, 73, 85, 57, 72, 99]$$

Select all the elements greater than 33 and create a list out of them. Then sort this list. We then have:

$$A'' = [35, 56, 57, 60, 68, 72, 73, 80, 82, 85, 99, 99]$$

Then we have that the 4<sup>th</sup> largest value is 82.