Name: Qiuyang Fu
ID: 108667420

CSCI 3104, Algorithms **Escobedo & Jahagirdar**
Homework 2B (55 points) **Summer 2020, 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**, we cannot accept hand-written solutions. Here's a short intro to **Latex**.
- In this homework we denote the asymptomatic *Big-O* notation by $\mathcal{O}$ and *Small-O* notation is represented as o.
- We recommend using online Latex editor **Overleaf**. Download the **.tex** file from Canvas and upload it on overleaf to edit.
- You should submit your work through **Gradescope** only.
- If you don't have an account on it, sign up for one using your CU email. You should have gotten an email to sign up. If your name based CU email doesn't work, try the identikey@colorado.edu version.
- Gradescope will only accept **.pdf** files (except for code files that should be submitted separately on Canvas if a problem set has them) and **try to fit your work in the box provided**.
- You cannot submit a pdf which has less pages than what we provided you as Gradescope won't allow it.

Name:	Qiuyang Fu
ID:	108667420

CSCI 3104, Algorithms **Escobedo & Jahagirdar**
Homework 2B (55 points) **Summer 2020, CU-Boulder**

Piazza threads for hints and further discussion

Piazza Threads
Question 1
Question 2
Question 3
Question 4

Recommended reading
Divide & Conquer; Recurrence Relations: Ch. 2 2.3; Ch. 4 4.1, 4.2, 4.3, 4.4, 4.5
Quicksort: Chapter 7 complete
Efficient Data Structures: Hash Tables: Ch. 11 Complete

Name: Qiuyang Fu

ID: 108667420

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

Homework 2B (55 points) **Summer 2020, CU-Boulder**

1. *(20 pts) Let $A = \langle a_1, a_2, \ldots, a_n \rangle$ be an array of numbers. Let's define a 'flip' as a pair of distinct indices $i, j \in \{1, 2, \ldots, n\}$ such that $i < j$ but $a_i > a_j$. That is, a_i and a_j are out of order.*
   For example - In the array A = [1, 3, 5, 2, 4, 6], (3, 2), (5, 2) and (5, 4) are the only flips i.e. the total number of flips is 3. (Note that in this example the indices are the same as the actual values)

   Design a divide-and-conquer algorithm with a runtime of $\mathcal{O}(n \log(n))$ for computing the number of flips. Your algorithm has to be a divide and conquer algorithm that is modified from the Merge Sort algorithm. Explain how your algorithm works, including pseudocode. You can add a picture of your pseudo-code or properly commented code.

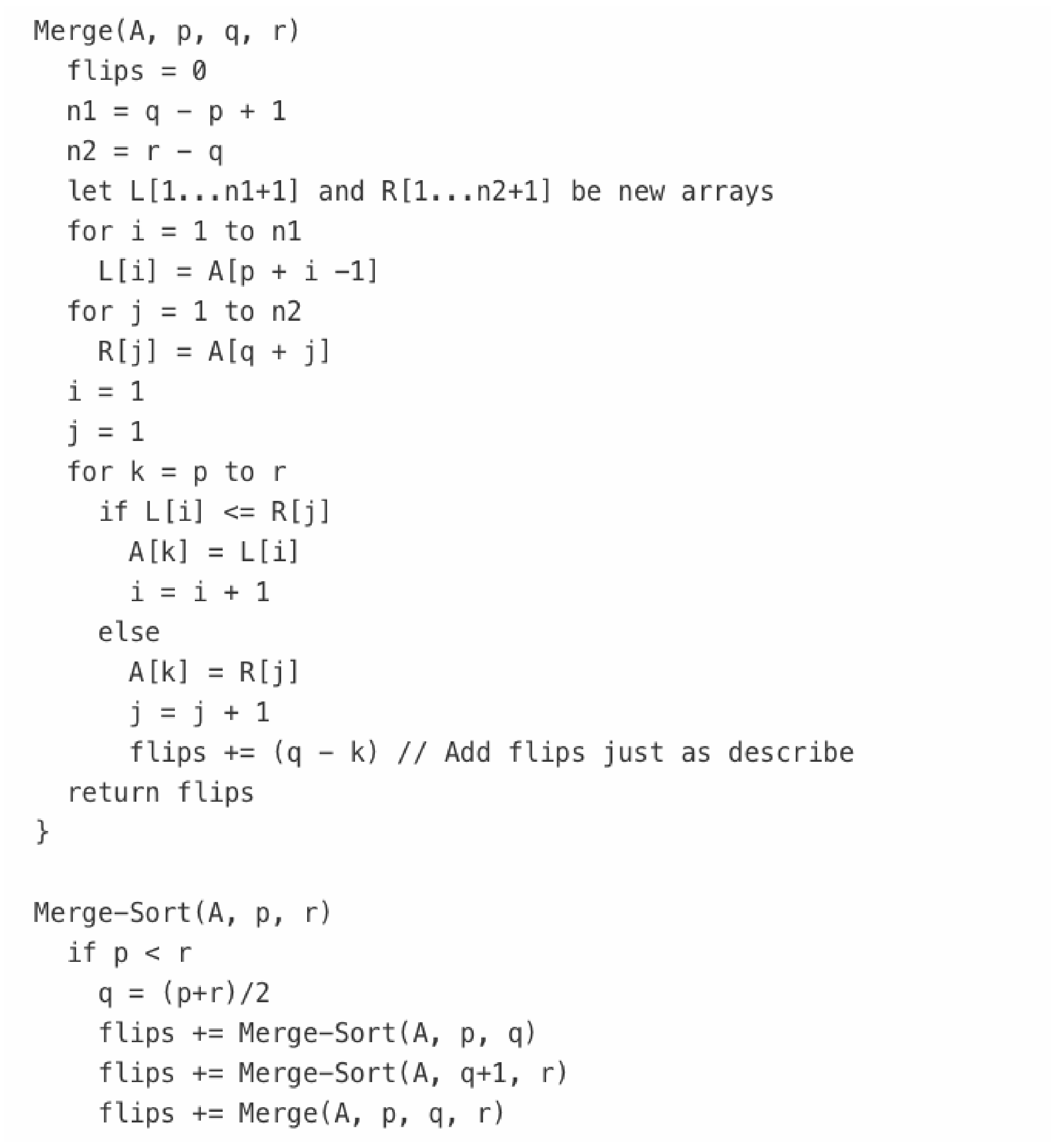
The algorithm to find the total number of flips using divide-and-conquer method is, we divide the main problem into two sub-problems iteratively until the number of elements in the sub-problem is equal to 1. Suppose the number of flips in the left and right sub-problem is already know, then the flips between the two subarrays need to accumulate in merge step. The detail description of this algorithm is showed as below:

(a) Divide the array into two equal halves in each step until there is only one element in the subarray.

(b) Using an almost same merge method in merge-sort that counts the number of inversions when two halves of the array are merged. The counting principle is suppose i is the index for left subarray and j is an index of the right subarray. if a[i] is greater than a[j], it means that there are (mid – i) inversions. because left and right subarrays are sorted, so all the remaining elements in left-subarray are greater than a[j].

(c) Recursively divide the array into halves and the number of flips is the sum of flips inside left and right subarray and the number of flips between them while merging.

Following the above description, here is the pseudo-code for the whole procedure:



```
Merge(A, p, q, r)
  flips = 0
  n1 = q - p + 1
  n2 = r - q
  let L[1...n1+1] and R[1...n2+1] be new arrays
  for i = 1 to n1
    L[i] = A[p + i -1]
  for j = 1 to n2
    R[j] = A[q + j]
  i = 1
  j = 1
  for k = p to r
    if L[i] <= R[j]
      A[k] = L[i]
      i = i + 1
    else
      A[k] = R[j]
      j = j + 1
      flips += (q - k) // Add flips just as describe
  return flips
}

Merge-Sort(A, p, r)
  if p < r
    q = (p+r)/2
    flips += Merge-Sort(A, p, q)
    flips += Merge-Sort(A, q+1, r)
    flips += Merge(A, p, q, r)
```

Name: Qiuyang Fu

ID: 108667420

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

Homework 2B (55 points) **Summer 2020, CU-Boulder**

2. *(10 pts) For the following problems, you must use the pseudocode for QuickSort and Partition in Section 7.1 of the Introduction to Algorithms 3rd edition (CLRS) and the array A = [22, 40, 67, 55, 10, 92, 66].*

   (a) *(3 pts) What is the value of the pivot in the call Partition(A; 1; 7)?* (Array indexing starts at 1 as per the convention in the book)

   From the pseudocode for QuickSort and Partition, it is obvious that the value of the pivot in the call Partition(A, 1, 7) is $A[7] = 66$, which always selects the last element as a pivot.

Name: Qiuyang Fu

ID: 108667420

CSCI 3104, Algorithms **Escobedo & Jahagirdar**
Homework 2B (55 points) **Summer 2020, CU-Boulder**

(b) *(3 pts) What is the index of that pivot value at the end of that call to Partition?*

From the pseudocode for QuickSort and Partition, the index of that pivot value at the end of that call to Partition is 5.

(c) *(4 pts) On the next recursive call to Quicksort, what sub-array does Partition evaluate? (Give the indices specifying the subarray.)?*

Since the previous calling to Partition function return the index of pivot as 5, means that elements before index 5 is smaller than $A[5]$ and elements after index 5 is larger than $A[5]$. Now the whole array is A = [22, 40, 55, 10, 66, 92, 67]. Then on the next recursive call to Quicksort the sub-array which Partition will evaluate is Partition(A, 1, 4).

Name: Qiuyang Fu

ID: 108667420

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

Homework 2B (55 points) **Summer 2020, CU-Boulder**

3. *(15 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?*

   It is obvious that when the value of k is 1 will result in the best partitioning which means that the pivot will partition the array into two almost balanced subarray.

Name: Qiuyang Fu

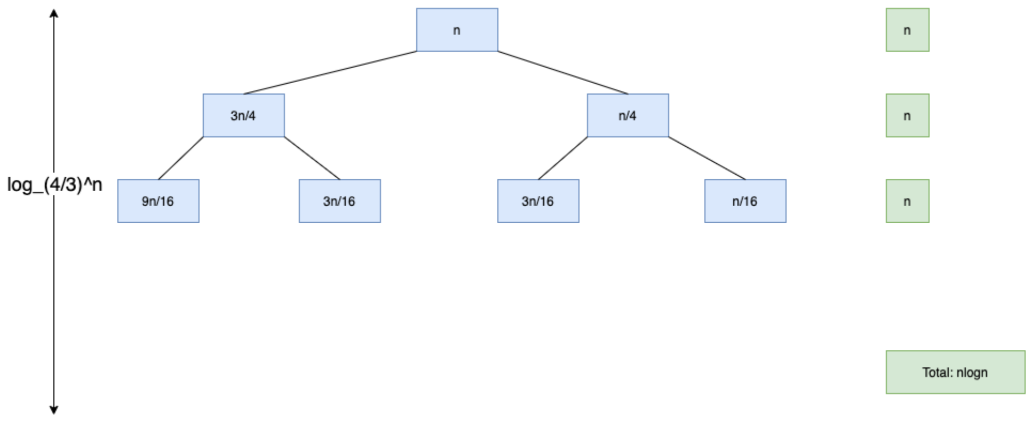
ID: 108667420

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

Homework 2B (55 points) **Summer 2020, CU-Boulder**

(b) *(9 pts) Write down a recurrence relation for this version of QuickSort, and solve it asymptotically using* ***recurstion tree*** *method. 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.*

The recurrence relation for this version of QuickSort is $T(n) = T[\frac{kn}{k+1}] + T[\frac{n}{k+1}] + \Theta(n)$. The recurrence relation of this question with the assume that $k = 3$ is $T(n) = T[\frac{3n}{4}] + T[\frac{n}{4}] + \Theta(n)$. The recurstion tree is showed as below, from the tree, it is obvious that the result is $n \log_{4/3} n$.



(c) *(3 pts) Provide a verbal explanation of how this Partition algorithm affects the running time of QuickSort, in comparison with the case where the best possible pivot is always used. Does the value of k affect the running time?*

From the above recurrence tree and equation, it is obvious that the Partition algorithm affects the height of the recurstion tree. Using the ratio of the size of the two subarrays divided by the pivot is a constant k, the height of the tree can be represent as $\log_{\frac{k+1}{k}} n$. Think about when k is relative large, this will lead to the worst case running time of quicksort with the height of tree equal to n and the time complexity is $O(n^2)$.

Name: Qiuyang Fu

ID: 108667420

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

Homework 2B (55 points) **Summer 2020, CU-Boulder**

4. *(10 pts) 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) *(2 pts) For which application does the hash function $h(k)$ perform worse? Please explain/justify adequately your answer.*

   The hash function $h(k)$ in application 1 perform worse. Since the keys will gather from 30 to 80 the utilize of the hash table is 50%. In application 2 the utilize of the hash table is 60%, and the utilize of the hash table in application 3 is 100%.

(b) *(3 pts) In each of the three applications does the hash function satisfy the uniform hashing property? Please explain/justify adequately your answer.*

In application 1, the hash function doesn't satisfy the uniform hashing property since some slots in the hash table will not be mapped. In application 2, the hash function doesn't satisfy the uniform hashing property since some slots in the hash table will not be mapped. In application 3, the hash function satisfy the uniform hashing property for all the slot in the hash table will be mapped equally with $\frac{1}{100}$.

(c) *(5 pts) Suppose you have n keys in total for each application. What is the resulting load factor α for each application? Assume that in each application only slots of the hash table that have a chance of being filled are considered when calculating α.*

The load factor for application 1 is $n/50$, the load factor for application 2 is $n/60$, the load factor for application 3 is $n/100$.

Name: Qiuyang Fu
ID: 108667420

CSCI 3104, Algorithms **Escobedo & Jahagirdar**
Homework 2B (55 points) **Summer 2020, CU-Boulder**

5. ***Extra Credit (5% of total homework grade)*** *For this extra credit question, please refer the leetcode link provided below or click here. Multiple solutions exist to this question ranging from brute force to the most optimal one. Points will be provided based on Time and Space Complexities relative to that of the most optimal solution.*

   Note*: The brute force approach in this question would be to create an array of size* 10^6 *and directly access the key using the index of the array. For this question, we expect you to perform collision handling once you obtain a hash value for a particular key.*

   Please provide your solution with proper comments which carries points as well.

   https://leetcode.com/problems/design-hashmap/

```
class MyHashMap {
public:
    vector<list<pair<int,int>>> hash_table;
    size_t hash_size = 10000;
    /** Initialize your data structure here. */
    MyHashMap() {
     // resize the hash_table with 10000 slots equal to the number of operations
        hash_table.resize(hash_size);
    }

    /** value will always be non-negative. */
    void put(int key, int value) {
        list<pair<int,int>> &slot = hash_table[key % hash_size];
        // loop to check whether the slot is empty handle collision
        for (auto &item : slot) {
            if (item.first == key) {
                item.second = value;
                return;
            }
        }
        slot.push_back(pair(key, value));
    }

    /** Returns the value to which the specified key is mapped, or -1 if this map c
    int get(int key) {
        list<pair<int,int>> slot = hash_table[key % hash_size];
        if (!slot.empty()) {
            for (auto item : slot) {
                if (item.first == key) {
                    return item.second;
                }
            }
        }
        return -1;
    }

    /** Removes the mapping of the specified value key if this map contains a mappi
    void remove(int key) {
        list<pair<int,int>> &slot = hash_table[key % hash_size];
        list<pair<int,int>>::iterator it = slot.begin();
        // loop to erase the value with the given key
        while (it != slot.end()) {
            if (it->first == key) {
                slot.erase(it);
                break;
            }
            it++;
        }
    }
};
```