Name:	Qiuyang Fu
ID:	108667420

CSCI 3104, Algorithms **Escobedo & Jahagirdar**
Mid Term exam Summer 2020 (30 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.

Honor code: On my honor as a University of Colorado at Boulder student, I have neither given nor sought unauthorized assistance in this work

Initials	QF
Date	06/27/2020

If you violate the CU **Honor Code**, you will receive a 0.

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**
Mid Term exam Summer 2020 (30 points) **Summer 2020, CU-Boulder**

Master Method: Consider a recurrence relation of the form

$$T(n) = aT(n/b) + f(n)$$

where $a \geq 1$ and $b > 1$ are constants, and $T(n) = \textit{constant}$ for $n \leq 1$.
The asymptotic growth of T(n) is bounded as follows:

- **Case 1** $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0 \implies T(n) = \Theta(n^{\log_b a})$
- **Case 2** $f(n) = \Theta(n^{\log_b a}) \implies T(n) = \Theta(n^{\log_b a} \log(n))$
- **Case 3** $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$ and if $af(\frac{n}{b}) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n. $\implies T(n) = \Theta(f(n))$

Formulae

- $1 + 2 + 3 + ... + n = \frac{n(n+1)}{2}$
- $1^2 + 2^2 + ... + n^2 = \frac{n(n+1)(2n+1)}{6}$
- **Sequences** The formulae for the sum of an Arithmetic Progression (AP) and Geometric Progression (GP) are available here

Name:	Qiuyang Fu
ID:	108667420

CSCI 3104, Algorithms **Escobedo & Jahagirdar**
Mid Term exam Summer 2020 (30 points) **Summer 2020, CU-Boulder**

1. (3 pts) Which of $\mathcal{O}$, Ω or Θ is the correct asymptotic relationship for $f(n) = n^{\frac{1}{3}} \log_3(n)$ compared to $g(n) = \sqrt{n}$? Write your answer as $f(n) = \tau(g)$ where τ should be replaced by the appropriate symbol $(\mathcal{O}, \Omega, \Theta)$, show the necessary work to justify your answer.

Base on the concept of $\mathcal{O}$, Ω or Θ, $f(n) = O(g(n))$. Since $f(n) = n^{\frac{1}{3}} \log_3(n) = n^{\frac{1}{3}} \frac{\lg(n)}{\lg(3)}$, $g(n) = n^{\frac{1}{2}}$, it is obvious that $\lg(n)$ is smaller than $n^{\frac{3}{2}}$. So $f(n) \leq cg(n)$ for all $n \geq n_0$, n_0 is a relative large number.

Name: Qiuyang Fu

ID: 108667420

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

Mid Term exam Summer 2020 (30 points) **Summer 2020, CU-Boulder**

2. (2 pts) Which of $\mathcal{O}$, Ω or Θ is the correct asymptotic relationship for $f(n) = \frac{n^2-8n+15}{n-5}$ compared to $g(n) = \frac{1^2+2^2+....+n^2}{1+2+...+n}$? Write your answer as $f(n) = \tau(g)$ where τ should be replaced by the appropriate symbol $(\mathcal{O}, \Omega, \Theta)$, show the necessary work to justify your answer

Here $f(n) = \frac{n^2-8n+15}{n-5} = n-3$, $g(n) = \frac{1^2+2^2+....+n^2}{1+2+...+n} = \frac{2n+1}{3}$. Base on the concept of $\mathcal{O}$, Ω or Θ, it is obvious that $f(n) = \Theta(g(n))$. For both $f(n)$ and $g(n)$ are monotonically increasing and it is easy to find constant such that $c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0$.

3. (1 pts) State True or false without justification
If the running time of an algorithm satisfies the recurrence relation
$T(n) = T(n/18) + T(17n/18) + cn$, then $T(n) = \mathcal{O}(n\log(n))$.

True

4. (1 pts) Let H be a hash table with 2020 slots with a hash function $h(x)$ that satisfies **uniform hashing** property. Given two items x_1, x_2, what is the probability that they do not hash to the same location?

Since the hash function $h(x)$ satisfies uniform hashing property, so the probability that the given two items do not hash to the same location is $1 - \frac{1}{m} = \frac{2019}{2020}$.

Name: Qiuyang Fu

ID: 108667420

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

Mid Term exam Summer 2020 (30 points) **Summer 2020, CU-Boulder**

5. (3 pts) Suppose $T(n) = T(n-5) + n$, where $T(n) = \Theta(1)$ if $n \leq 5$. Find a function $g(n)$ such that $T(n) = \Theta(g(n))$. Clearly justify your answer.

It is obvious that the recurrence relations can be showed as below:

$$\begin{aligned} T(n) &= T(n-5) + n \\ &= T(n-10) + 2n - 5 \\ &= T(n-15) + 3n - 15 \\ &= ... \\ &= T(n - \lfloor n/5 \rfloor \times 5) + \sum_{i=0}^{\lfloor n/5 \rfloor + 1} n - 5i \end{aligned} \tag{1}$$

Since $T(n) = \Theta(1)$ if $n \leq 5$, so $T(n) = \Theta(n^2)$.

Name: Qiuyang Fu

ID: 108667420

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

Mid Term exam Summer 2020 (30 points) **Summer 2020, CU-Boulder**

6. (1 pt) For the set $\Sigma = \{a, b, c\}$, give the set of inequalities on the frequencies f_a, f_b, f_c that would yield the corresponding codewords $00, 01, 1$ respectively under Huffman's algorithm. Assume that while constructing the tree, we merge two nodes such that the node with least frequency is the left child. In your tree branching left corresponds to 0 and branching right corresponds to 1. List the frequencies f_a, f_b, f_c below that produce the specified codewords. Your choice of frequencies should **always** produce the same tree/codes provided.

Base on the corresponding codewords of each words, the frequencies of f_a, f_b, f_c are 2, 3, 10 respectively. For $f_a < f_b < f_c$ and $f_a + f_b < f_c$.

7. (3 pts) For the given algorithm, solve the following.

You may assume the existence of a `max` function taking $\mathcal{O}(1)$ time, which accepts at most four arguments and returns the highest of the four.

```
FindMax(A[0, ..., n-1]):
     if A.length == 1:
         return A[0]
    count = 0
    for i = 1 to n/2:
        count++

     return max( FindMax(A[0, ..., floor(n/4)],
                 FindMax(A[floor(n/4) + 1, ..., floor(n/2)],
                 FindMax(A[floor(n/2) + 1, ..., floor(3n/4)],
                 FindMax(A[floor(3n/4) + 1, ..., n-1]))
```

Find a recurrence for the worst-case runtime complexity of this algorithm. You can

Name: Qiuyang Fu

ID: 108667420

assume that n is a multiple of four. Solve the recurrence found to obtain worst-case runtime complexity.

Suppose n is a multiple of four, the recurrence of this algorithm is $T(n) = 4T(n/4) + \Theta(n/2)$. Base on the Master Method, since $a = 4$, $b = 4$, $f(n) = n$, the worst-case runtime complexity of this algorithm is $n\log(n)$.

Name: Qiuyang Fu

ID: 108667420

CSCI 3104, Algorithms **Escobedo & Jahagirdar**
Mid Term exam Summer 2020 (30 points) **Summer 2020, CU-Boulder**

8. (4 pts) Assume there are n items $\{item_1, item_2, ...\ item_n\}$, each item has a weight w_i and value v_i associated with it. You have a bag that can carry a maximum load of weight W. Each of the n items can be divided into **fractions** such that the value and weight associated with the item decreases proportionally.
The inputs to your function will be values v, weights w, number of items n and capacity W.
Provide well commented pseudo or actual code, that returns the maximum total value of all items that can be carried.
Also briefly discuss the space and runtime complexity of your pseudo-code.

Input
v = [20, 27, 18]
w = [2, 3, 3]
$W = 3$
Output
29
The Algorithm should pick $item_1$ and $\frac{1}{3}^{rd}$ of $item_2$ leading to a total value of $20+27\frac{1}{3} =$ 29.

The space complexity of my algorithm is $O(n)$, and the runtime complexity of my algorithm is $O(nW)$.

```
int maximum_weight(int v[], int w[], int W, int len) {
  int price[len];
  int max_value = 0;
  // Calculate the value per weight
  for (int i = 0;i < len;i++) {
    price[i] = v[i] / w[i];
  }
  int i = W;
  // Loop to find the maximum until run out of the weight of bag
  while(i != 0){
    int max_temp = 0;
    int max_index = 0;
    for (int j = 0;j < len;j++) {
      if (w[j] != 0 && price[j] > max_temp) {
        max_temp = price[j];
        max_index = j;
      }
    }
    int num = min(w[max_index], i);
    w[max_index] -= num;
    max_value += max_temp * num;
    i -= num;
  }
  return max_value;
}
```

Name: Qiuyang Fu

ID: 108667420

CSCI 3104, Algorithms **Escobedo & Jahagirdar**
Mid Term exam Summer 2020 (30 points) **Summer 2020, CU-Boulder**

9. (6 pts) Given an array A and a value k, design a divide and conquer algorithm to find the k^{th} smallest element in the array. The algorithm proposed should not use extra space i.e the space complexity of the algorithm should be constant $\Theta(1)$. Your algorithm must have an average case runtime of $\mathcal{O}(n\log(n))$. You can assume the access to a function which returns a random number within a range in constant time. You are allowed to modify the array passed as input.
   The inputs to your function will be an array A and value k.
   Provide well commented pseudo or actual code for the algorithm.

   Assume that you have access to a function *rand(min, max)* which will return a random integer between *min* and *max* in constant time inclusive of both min and max.

   Input
   A = [10, 13, 20, 8, 7, 6, 100]
   $k = 4$
   Output
   10

   10 is the 4th smallest value in the given array.

```
int Partition(int nums[], int start, int end) {
  // Or can be replace using rand() function
  int mid = (start + end) / 2;
  swap(nums[mid], nums[end]);
   int k = start - 1;
   for (int i = start; i < end; i++) {
     if (nums[i] <= nums[end]) {
       k++;
       if (k != i) {
         swap(nums[k], nums[i]);
         }
       }
    }
    k++;
    swap(nums[k], nums[end]);
    return k;
   }

int findKthSmallest(int nums[], int k, int len) {
  int start = 0;
  int end = len - 1;
  // Get the pivot same as Quicksort
  int i = Partition(nums, start, end);
  // Loop until the kth smallest value is found
  while (i != k - 1) {
    if (i < k - 1) {
      start = i + 1;
      i = Partition(nums, start, end);
    } else {
      end = i - 1;
      i = Partition(nums, start, end);
    }
  }
  return nums[i];
}
```

Name: Qiuyang Fu

ID: 108667420

CSCI 3104, Algorithms **Escobedo & Jahagirdar**
Mid Term exam Summer 2020 (30 points) **Summer 2020, CU-Boulder**

10. (6 pts) Assume there are n carrots and n rabbits along a straight line. Each rabbit needs to eat a carrot. Rabbits can move in either direction, simultaneously along the line and travelling 1 unit of distance takes 1 minute. Design a greedy algorithm that takes $\mathcal{O}(n \log(n))$ to assign carrots to rabbits such that the time taken to eat the last carrot is minimized. The algorithm should return the value of the time taken to eat last carrot.
You will be given the position of rabbits and carrots along the straight line.

Expectations

- You should clearly describe the greedy choice that the algorithm makes in assigning carrots to rabbits.
- Provide well commented pseudo or actual code for the algorithm.
- Discuss the space and runtime complexity of the pseudo or actual code.

Example 1:
rabbits = [7, 3, 2, 13, 2]
carrots = [1, 3, 5, 14, 21]
output : 8
In this example the assignment is as follows.

- The carrot at distance 1 (index 0) is eaten by rabbit at distance 2 (index 4).
- The carrot at distance 3 (index 1) is eaten by rabbit at distance 2 (index 2).
- The carrot at distance 5 (index 2) is eaten by rabbit at distance 3 (index 1).
- The carrot at distance 14 (index 3) is eaten by rabbit at distance 7 (index 0).
- The carrot at distance 21 (index 4) is eaten by rabbit at distance 13 (index 3), with 21-13=8 minutes being the longest time.

Example 2:
rabbits = [84, 15, 15, 161, 187, 9, 66, 1]
carrots = [92, 103, 163, 119, 63, 117, 144, 172]
output : 102

The greedy choice of the algorithm is each rabbit will always get the carrot which is the nearest in the remaining carrots. In example 1, since carrot at distance 5 is already been taken by rabbit at distance 3 so rabbit at distance 7 will get to carrot 14 somehow. Since the algorithm showed below modified the origin rabbits and carrots array so the space complexity is $O(1)$ and runtime complexity is $O(n \log n)$.

```
int time_last_carrot(int rabbits[], int carrots[], int len)  {
  sort(rabbits, rabbits + len);
  sort(carrots, carrots + len);
  int max_time = 0;
  for (int i = 0;i < len;i++) {
    if (abs(rabbits[i] - carrots[i]) > max_time) {
      max_time = abs(rabbits[i] - carrots[i]);
    }
  }
  return max_time;
}
```

Name: Qiuyang Fu

ID: 108667420

CSCI 3104, Algorithms **Escobedo & Jahagirdar**
Mid Term exam Summer 2020 (30 points) **Summer 2020, CU-Boulder**

11. **For extra credit pick one of the two questions provided. Extra credit will only be considered if your midterm score less than 100% (2 pts)**

    For this extra credit question, please refer the leetcode link provided below. 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.

    Please provide your solution with proper comments, solutions without proper comments will not be considered.

    https://leetcode.com/problems/gas-station/

    OR

    https://leetcode.com/problems/maximal-square/

```
class Solution {
 public:
  int maximalSquare(vector<vector<char>>& matrix) {
    long m = matrix.size();
    // Handle the special case while the matrix is empty
    if (m == 0) {
      return 0;
    }
    long n = matrix[0].size();
    int status[m][n];
    int maxArea = 0;
    // Initial the status matrix and maxArea
    for (int i = 0; i < m; i++) {
      for (int j = 0; j < n; j++) {
        status[i][j] = matrix[i][j] - '0';
        if (status[i][j] == 1) {
          maxArea = 1;
        }
      }
    }
    // Solving this using dp
    for (int i = 1; i < m; i++) {
      for (int j = 1; j < n; j++) {
        int lines = status[i - 1][j - 1];
        // Loop until the square is found or all cases are handled
        while (lines != 0 && matrix[i][j] != '0') {
          bool flag = true;
          // check the row whether it can form a square
          for (int k = i - 1; k >= i - lines && k >= 0; k--) {
            if (matrix[k][j] == '0') {
              flag = false;
              break;
            }
          }
          // check the column whether it can form a square
          for (int k = j - 1; k >= j - lines && flag && k >= 0; k--) {
            if (matrix[i][k] == '0') {
              flag = false;
              break;
            }
          }
          // Update the status and the maxArea when a new square found
          if (flag) {
            status[i][j] = lines + 1;
            int tmpArea = status[i][j] * status[i][j];
            if (tmpArea > maxArea) {
              maxArea = tmpArea;
            }
            break;
          }
          lines--;
        }
      }
    }
    return maxArea;
  }
};
```