Due Date: 4th Oct 2021
20% penalty for 1 day late
40% penalty for 2 days late
Submission not allowed afterwards

CS2009: Design and Analysis of Algorithms (Fall 2021)
Assignment 1
Total Marks: 100

1. Show the steps insertion sort uses to sort the following list of integers in the descending order (from the highest to the lowest / biggest to the smallest):

$$15, 6, 12, 8, 7, 1, 9, 3, 5, 23$$

Show the value of the key variable, k, at each step. Explain briefly why time complexity of insertion sort is $O(n^2)$. Use Loop invariant to show its correctness. [5 Points]

Solution.

[15, 6, 12, 8, 7, 1, 9, 23] // k = 6, 6 remains at its position
[15, 6, 12, 8, 7, 1, 9, 23] // k = 12
[15, 12, 6, 8, 7, 1, 9, 23] // k = 8
[15, 12, 8, 6, 7, 1, 9, 23] // k = 7
[15, 12, 8, 7, 6, 1, 9, 23] // k = 1, 1 remains at its position
[15, 12, 8, 7, 6, 1, 9, 23] // k = 9
[15, 12, 9, 8, 7, 6, 1, 23] // k = 23
[23, 15, 12, 9, 8, 7, 6, 1,] **done.**

Complexity of Insertion Sort:
$T(n) = c_1n + c_2(n-1) + c_4(n-1) + c5\sum_{j=2}^{n}(t_j) + c6\sum_{j=2}^{n}(t_j - 1) + c6\sum_{j=2}^{n}(t_j - 1) + c8(n-1)$
$T(n) = c_1n + c_2(n-1) + c_4(n-1) + c5(1/2n(n+1) - 1) + c6(1/2n(n-1)) + c6(1/2n(n-1)) + c8(n-1)$
$T(n) = O(n^2)$
To prove insertion sort is correct, we will use "loop invariants." The loop invariant we'll use is:
Lemma: At the start of each iteration of the for loop, the subarray $A[1 \cdots j-1]$ consists of the elements originally in $A = [1 \cdots j-1]$, but in sorted order.
Initialization: Before the first iteration (which is when $j = 2$, the subarray $A[1 \cdots j-1]$ is just the first element of the array, $A[1]$. This subarray is sorted, and consists of the elements that were originally in $A = [1 \cdots 1]$.
Maintenance: Suppose $A[1 \cdots j-1]$ is sorted. Informally, the body of the for loop works by moving $A[j-1], A[j-2], A[j-3]$ and so on by one position to the right until it finds the proper position for $A[j]$. The subarray $A[1 \cdots j-1]$ then consists of the elements originally in $A[1 \cdots j-1]$ but in sorted order. Incrementing j for the next iteration of the for loop then preserves the loop invariant.
Termination: The condition causing the for loop to terminate is that $j > n$. Because each loop iteration increases j by 11, we must have $j = n+1$ at that time. By the initialization and maintenance steps, we have shown that the subarray $A[1 \cdots j-1+1-1] = A[1 \cdots n]$ consists of the elements originally in $A[1 \cdots n]$, but in sorted order.

■

2. Show the steps merge sort uses to sort the following list of integers in the descending order (from the highest to the lowest / biggest to the smallest): [5 points]

$$15, 6, 12, 8, 7, 1, 9, 3, 5, 23$$

Consider the following variation on Merge Sort, that instead of dividing input in half at each step of Merge Sort, you divide into three part, sort each part, and finally combine all of them using a three-way merge subroutine. What is the overall asymptotic running time of this algorithm? (Hint: Use Master Theorem) [5 points]

Solution. [15, 6, 12, 8, 7, 1, 9, 23]
15; 6; 12; 8 | 7; 1; 9; 23

15; 6 | 12; 8 | 7; 1 | 9; 23

15 | 6 | 12 | 8 | 7 | 1 | 9 | 23

15; 6 | 12; 8 | 7; 1 | 23; 9

15; 12; 8; 6 | 23; 9; 7; 1

23; 15; 12; 9; 8; 7; 6; 1;

Time Complexity using Master Method
$T(n) = aT(n/b) + f(n)$
a = 3, b = 3, f(n) = O(n)
$T(n) = 3T(n/3) + O(n)$
$T(n) = O(n \log_3 n)$

■

3. Repeat for Quick Sort. Use Loop invariant to show its correctness. [5 points]

$$15, 6, 12, 8, 7, 1, 9, 3, 5, 23$$

Solution. [15, 6, 12, 8, 7, 1, 9, 3, 5, 23] // pivot = 23
[23, 6, 12, 8,7, 1, 9, 1, 3, 5, 15]
[23, 6, 12, 8, 7, 1, 3, 5, 15 // pivot = 15
[23, 15, 12, 8,7, 1, 9, 3, 5, 6] // pivot = 6
[23, 15, 12, 8, 7, 9, [6, [3], [5], [1] // pivot = 9
[23, [15, 12, [9, 7, 8 , [6, 3 , 5, 1 // next pivot is 8
[23, [15, 12, [9, 7, [8 , [6, 3 , 5, 1 // next pivot is 1
[23, [15, 12, [9, 7, [8 , [6, 3 , 5, [1 // next pivot is 5
[23, 15, 12, 9, 8,7, 6, 5, 3, 1,] **done.**
Initialization: Before the first iteration, $i = p - 1$ and $j = p$ Because no values lie between p and i and no values lie between $i + 1$ and $j - 1$ the first two conditions of the loop invariant are trivially satisfied.
See page no. 173 for further details.

■

4. Let suppose you are building an Emoji classification model. You find out that your model is able to classify one emoji type easily and face difficulty in classifying other emojis types. The one possible reason behind your model biasness toward one particular emoji could be due to class imbalance. Your training input may contain a lot of examples of one particular emoji and very few examples for other emojis.

   Your tasks is to determine whether there is a emoji type in input array $A = [a_1, a_2, a_3, \cdots, a_n]$ that appears more than n/2 times. The elements of the array are GIF files. However you can answer questions of the form: is $A[i] = A[j]$? in constant time.

   Design an $O(n \log_2 n)$ algorithm to solve this problem. [5 points]

   Design linear-time $O(n)$ algorithm for solving above problem [5 points]

code	emoji	label
:heart:	❤️	0
:baseball:	⚾	1
:smile:	😀	2
:disappointed:	😔	3
:fork_and_knife:	🍴	4

Figure 1: EMOJISET - a classification problem.

Solution. Classical divide and conquer: split input array A into two subsets, $A1$ and $A2$, ..., and show $T(n)$ is $O(nlogn)$.
If input array A has a lot of examples of one particular emoji type (majority/dominating emoji type) E1, E1 must also be a majority/dominating emoji type of A1 or A2 or both. The equivalent contra-positive restatement is immediate: (If $E1$ is ¡= half (n/2) in each, it is ¡= half in the total.) If both parts have the same majority element, it is automatically the majority element for A. If one of the parts has a majority/dominating emoji type, count the number of repetitions of that emoji type in both parts (in O(n) time) to see if it is a majority/dominating emoji type. If both parts have a majority/dominating emoji type, you may need to do this count for each of the two candidates, still O(n). This splitting can be done recursively. The base case is when n = 1. A recurrence relation is T(n) = 2T(n/2) + O(n), so T(n) is O(n log n) by the Master Theorem.
Linear Time Algorithm using Boyer-Moore Majority Vote Algorithm.

- Initialize a max_index = 0 and count = 0. The element at index max_index is considered to be our current candidate for majority/dominating emoji type.
- Traverse the array updating max_index and count according to the following conditions:-
  - If A[max_index] == A[i], increase count by 1.
  - Else, decrease count by 1.
  - If count == 0, max_index = i and count = 1.
- Check the frequency of the emoji type at max_index via a linear traversal of the array

Time Complexity: Linear traversal of array + Finding frequency of A[max$_i ndex$] $= O(n) + O(n) = O(n)$

■

5. Let suppose you are given the task of creating two teams each of n players from the pool of 2n players for the competition. Each player has a numerical rating assigned to him/her according to their talent. Show plausible strategy for dividing players as fairly as possible to avoid talent imbalance between team 1 and team 2. Design an $O(n \log_2 n)$ algorithm to solve this problem. [10 points]

Solution. Use Sorting Algorithm like Merge Sort to sort players. Then select team 1 from odd index (1, 3, 5, 7, ..., n-1) and select other team 2 from even index (2, 4, 6, 8, ..., n) of sorted array.

■

6. Prove $5n^2 - 2n + 30 = O(n^2)$. Determine the values of constant c and n_0. [5 Points] Prove $2n^2 \log_2 n + 3n^2 = O(n^2 \log_2 n)$. Determine the values of constant c and n_0. [5 Points]

Solution. Prove $5n^2 - 2n + 30 = O(n^2)$.
$\frac{f(n)}{g(n)} = \frac{5n^2-2n+30}{n^2} < \frac{5n^2-2n^2+30n^2}{n^2} = 33$
$c = 33, n_0 = 1$
or $c = 35, n_0 = 1$
Prove $2n^2 \log_2 n + 3n^2 = O(n^2 \log_2 n)$
$\frac{f(n)}{g(n)} = \frac{2n^2 \log_2 n+3n^2}{n^2 \log_2 n} < \frac{2n^2 \log_2 n+3n^2 \log_2 n}{n^2 \log_2 n} = 5$
$c = 5, n_0 = 2$

■

7. Watch the video lecture on Big O, Big Ω and Big Θ notation from

   http://www.youtube.com/watch?v=6Ol2JbwoJp0. Write the summary of the lecture in your words. [10 Points]

Solution. Give full marks if they write something about Big O, Big Ω and Big Θ notation

■

8. Use Master Theorem, to calculate the time complexity of the following [15 points]

$$T(n) = 4T(\frac{n}{2}) + n^3. \tag{1}$$

$$T(n) = 3T(\frac{n}{2}) + n^2. \tag{2}$$

$$T(n) = 9T(\frac{n}{2}) + n. \tag{3}$$

Solution. 1. $T(n) = 4T(\frac{n}{2}) + n^3$
a = 4, b = 2, d = 3
Case 1 : $a < b^d$
Complexity $O(n^3)$

2. $T(n) = 3T(\frac{n}{2}) + n^2$
a = 3, b = 2, d = 2 Case 1 : $a < b^d$
Complexity $O(n^2)$

3. $T(n) = 9T(\frac{n}{2}) + n$
a = 9, b = 2, d = 1 Case 3 : $a > b^d$
Complexity $O(n^{log_2 9}) = O(n^{2.7})$ ■

9. Use Iteration Method, to calculate the time complexity of the following [10 points]

$$T(n) = 6T(\frac{n}{2}) + n, (T(1) = 1). \tag{4}$$

$$T(n) = 3 + T(\frac{n}{2}), (T(1) = 1). \tag{5}$$

Solution. 1. $T(n) = 6T(\frac{n}{2}) + n, (T(1) = 1).$
Divide both side with n
$\frac{T(n)}{n} = \frac{6T(\frac{n}{2})}{n} + 1$

Convert 6 into $2^{log_2 6}$

$\frac{T(n)}{n} = \frac{T(\frac{n}{2})}{n/2^{log_2 6}} + 1$

$\frac{T(n)}{n} = \frac{T(\frac{n}{4})}{n/4^{log_2 6}} + 1 + 1$

$\frac{T(n)}{n} = \frac{T(\frac{n}{8})}{n/8^{log_2 6}} + 1 + 1 + 1$

.....

$\frac{T(n)}{n} = \frac{T(\frac{n}{n})}{n/n^{log_2 6}} + 1 + 1 + ... + 1$

$\frac{T(n)}{n} = T(1).n^{log_2 6}/n + logn$

$T(n) = n^{log_2 6} + nlogn$
Complexity is $O(n^{log_2 6})$

■

Solution. 2. $T(n) = 3 + T(\frac{n}{2}), (T(1) = 1)$.
$T(n) = 3 + (3 + T(\frac{n}{4}))$

$T(n) = 3 + 3 + (3 + T(\frac{n}{8}))$

$T(n) = 3 + 3 + 3 + (3 + T(\frac{n}{16}))$

.....

$T(n) = 3 + 3 + 3 + 3 + ...(3 + T(\frac{n}{n}))$

$T(n) = logn + T(1)$

Complexity is $O(logn)$

■

10. For each of the following questions, indicate whether it is T (True) or F (False) and justify using some examples e.g. assuming a function? [15 Points]

- For all positive $f(n), g(n)$ and $h(n)$, if $f(n) = O(g(n))$ and $f(n) = \Omega(h(n))$, then $g(n) + h(n) = (f(n))$.

  Solution. $f(n) = n, O(g(n))$ can be n , and $\Omega(h(n))$ can be $logn$, Thus, $n + logn \neq n$, Thus Equation is False

  ■

- if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$, then we have $(f(n))^3 = \Theta((g(n))^3)$

  Solution. If $f(n) = 2n$, $g(n)$ can be n or $(2n - 1)$ or any equation with linear n in order to satisfy both $f(n) = O(g(n)$ and $f(n) = \Omega(g(n))$ simultaneously. Thus True

  ■

- if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$, then we have $(f(n))^2 = (g(n))^2$

  Solution. $if f(n) = n, O(g(n))$ can be n, $\Omega(g(n))$ can be $logn$,
  Thus, $(n)^2 \neq (logn)^2$, Thus Equation is False

  ■