Due Date: 17 September 2022
20% penalty for 1 day late
40% penalty for 2 days late
Submission not allowed afterwards

CS2009: Design and Analysis of Algorithms (Fall 2022)
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):

$$6, 16, 12, 27, 9, 1, 18, 5, 31$$

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.

[6, 16, 12, 27, 9, 1, 18, 5, 31] // k = 16 remains at its position
[16, 6, 12, 27, 9, 1, 18, 5, 31] // k = 12
[16, 12, 6, 27, 9, 1, 18, 5, 31] // k = 27
[27, 16, 12, 6, 9, 1, 18, 5, 31] // k = 9
[27, 16, 12, 9, 6, 1,18, 5, 31] // k = 1, 1 remains at its position
[27, 16, 12, 9, 6, 1, 18, 5, 31] // k = 18
[27, 18, 16, 12, 9, 6, 1, 5, 31] // k = 5
[27, 18, 16, 12, 9, 6, 5, 1, 31]
[31 , 27, 18, 16, 12, 9, 6, 5, 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]

$$6, 16, 12, 27, 9, 1, 18, 5, 31$$

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. [6, 16, 12, 27, 9, 1, 18, 5, 31]

$\boxed{6, 16, 12, 27, 9}$ $\boxed{1, 18, 5, 31}$

$\boxed{6, 16, 12}$ $\boxed{27, 9}$ $\boxed{1; 18}$ $\boxed{5; 31}$

$\boxed{6, 16}$ $\boxed{12}$ $\boxed{27, 9}$ $\boxed{1; 18}$ $\boxed{5; 31}$

$\boxed{6}$ $\boxed{16}$ $\boxed{12}$ $\boxed{27}$ $\boxed{9}$ $\boxed{1}$ $\boxed{18}$ $\boxed{5}$ $\boxed{31}$

$\boxed{16, 6}$ $\boxed{12}$ $\boxed{27, 9}$ $\boxed{18; 1}$ $\boxed{31; 5}$

$\boxed{16, 12, 6}$ $\boxed{27, 9}$ $\boxed{1; 18}$ $\boxed{5; 31}$

$\boxed{27, 16, 12, 9, 6}$ $\boxed{31, 18, 5, 1}$

$\boxed{31, 27, 18, 16, 12, 9, 6, 5, 1}$

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]

$$6, 16, 12, 27, 9, 1, 18, 5, 31$$

Solution. [6, 16, 12, 27, 9, 1, 18, 5, $\boxed{31}$] // pivot = 31
[31, 16, 12, 27, 9, 1, 18, 5, $\boxed{6}$] // pivot = 6
[31, 16, 12, 27, 9, $\boxed{18}$, 6, 5, 1] // pivot = 18
[31, 27, 18, 16, 9, $\boxed{12}$, 6, 5, 1] // pivot = 12
[31, 27, 18, 16, 12, 9, 6, 5, $\boxed{1}$] // pivot = 1
[31, 27, 18, 16, 12, 9, 6, 5, 1]
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. Suppose you're consulting for a bank that's concerned about fraud detection, and they come to you with the following problem. They have a collection of n bank cards that they've confiscated, suspecting them of being used in fraud. Each bank card is a small plastic object, containing a magnetic stripe with some encrypted data, and it corresponds to a unique account in the bank. Each account can have many bank cards corresponding to it, and we'll say that two bank cards are equivalent if they correspond to the same account.

   It's very difficult to read the account number off a bank card directly, but the bank has a high-tech "equivalence tester" that takes two bank cards and, after performing some computations, determines whether they are equivalent.

   Their question is the following: among the collection of n cards, is there a set of more than n/2 of them that are all equivalent to one another? Assume that the only feasible operations you can do with the cards are to pick two of them and plug them in to the equivalence tester.

   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]

Solution. Classical divide and conquer: split input array A into two subsets, $A1$ and $A2, \ldots,$ 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. Take a sequence of 2n real numbers as input. Design an $O(n \log_2 n)$ algorithm that partitions the numbers into n pairs, with the property that the partition minimizes the maximum sum of a pair. For example, say we are given the numbers (1,3,5,9). The possible partitions are ((1,3),(5,9)), ((1,5),(3,9)), and ((1,9),(3,5)). The pair sums for these partitions are (4,14), (6,12), and (10,8). Thus the third partition has 10 as its maximum sum, which is the minimum over the three partitions. [10 points]

Solution. Step 1: Use Sorting Algorithm like Merge Sort to sort. Step 2: Create pairs from endpoints
x = sort(x) for int in range 1 to n: index1 = i index2 = n - (i+1) new Pair(x[index1], x[index2])

■

6. Prove $n^3 - 2n + 1 = O(n^3)$. Determine the values of constant c and n_0. [5 Points] Prove $5n^2 \log_2 n + 2n^2 = O(n^2 \log_2 n)$. Determine the values of constant c and n_0. [5 Points]

Solution. Prove $n^3 - 2n + 1 = O(n^3)$.
$c \geq 2, n_0 = 1$
or $c = 1, n_0 \geq 2$
Prove $5n^2 \log_2 n + 2n^2 = O(n^2 \log_2 n)$
$c \geq 7, n_0 = 2$

■

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]

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

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

$$T(n) = 8T(\frac{n}{2}) + c.n^3. \tag{3}$$

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

2. $T(n) = 4T(\frac{n}{3}) + c.n$
a = 4, b = 3, d = 1 Case 3 : $a > b^d$
Complexity $O(n^{log_3 4}) = O(n^{1.26})$

3. $T(n) = 8T(\frac{n}{2}) + c.n^3$
a = 8, b = 2, d = 3 Case 2 : $a = b^d$
Complexity $O(n^3 logn)$

■

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

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

$$T(n) = 4T(\frac{n}{3}) + n, (T(1) = 1). \quad (5)$$

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 $g(n) + h(n) = \Omega(f(n))$.

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

- Let f (n) and g(n) be asymptotically nonnegative functions, then max(f (n), g(n)) = Θ (f (n) + g(n)).
- if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$, then we have $(f(n))^2 = (g(n))^2$

Solution. If $f(n) = 2n$, $g(n)$ 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 ■

$$T(n) = 2T(n/3) + n^2$$

$$\therefore T(n/3) = 2T(n/9) + n^2/9$$

$$\therefore T(n/9) = 2T(n/9/3) + n^2/81$$

$$T(n/9) = 2T\left(\frac{n}{27}\right) + n^2/81$$

Start substitution

$$T(n) = 2T(n/3) + n^2$$

Replace

$$T(n/3) = 2T(n/9) + n^2/9$$

$$T(n) = 2\cdot(2\cdot T(n/9) + n^2/9 + n^2$$

Replace $T(n/9)$

$$T(n) = 2\cdot 2(2\,T(n/27)) + \frac{n^2}{81} + \frac{n^2}{9} + n^2$$

Scanned with CamScanner

$$T(n) = 2 \times 2 \times 2\, T\left(n/3^3\right) + \frac{n^2}{(3^2)^2} + \frac{n^2}{(3^1)^2} + \frac{n^2}{(3^0)^2}$$

$$T(n) = 2^K T\left(\frac{n}{3^K}\right) + \underbrace{\left(\frac{1}{(3^2)^2} + \frac{1}{(3^1)^2} + \frac{1}{(3^0)^2}\right)}_{\text{geometric series}} n^2$$

$$= 2^K T\left(n/3^K\right) + n^2$$

$$= \text{For } K = \log_3 n \Rightarrow n = 3^K$$

$$= 2^{\log_3 n} \cdot T(1) + n^2$$

$$= n^{\log_3 2} + n^2$$

This is larger number so complexity will be n^2

$$= O(n^2)$$

Scanned with CamScanner

$T(n) = 4T(n/3) + n, \; T(1) = 1$

$\therefore$

$T(n/3) = 4T(n/9) + n/3$

$T(n/9) = 4T(n/9/3) + n/9$

$T(n/9) = 4T(n/27) + n/9$

Start Substitution

$T(n) = 4T(n/3) + n \rightarrow ①$

Put value of $T(n/3)$ in $T(n)$ equat 1

$T(n) = 4 \cdot 4\, T(n/9) + n/3 + n \rightarrow$ eq ②

Put value of $T(n/9)$ in eq ②

$T(n) = 4 \cdot 4 \cdot 4\, T(n/27) + n/9 + n/3 + n$

$T(n) = 4^3\, T\left(\frac{n}{3^3}\right) + \underbrace{\left(\frac{1}{9} + \frac{1}{3} + 1\right)}_{\text{geometric series}} n$

geometric series equal to 1

$T(n) = 4^3 \, T(n/3^3) + n$

$T(n) = 4^k \, T\left(\frac{n}{3^k}\right) + n$

For $k = \log_3 n \Rightarrow n = 3^k$

$T(n) = 4^{\log_3 n} \, T\left(\frac{n}{3^{\log_3 n}}\right) + n$

$= 4^{\log_3 n} \, T\left(\frac{n}{n^{\log_3 3}}\right) + n$

$\log_3 3 = 1$

$= \cancel{4 \log} \; 4^{\log_3 n} \, T(n/n) + n$

$= n^{\log_3 4} \, T(1) + n$

$= \underbrace{n^{1.26}}_{} + n$

This is larger number
so complexity will be $n^{1.26}$

$= O(n^{\log_3 4})$ or $O(n^{1.26})$

Scanned with CamScanner

$h(n) = \max(f(n), g(n))$. Then

$$h(n) = \begin{cases} f(n) & \text{if } f(n) \geq g(n), \\ g(n) & \text{if } f(n) < g(n). \end{cases}$$

Since $f(n)$ and $g(n)$ are asymptotically nonnegative, there exists n_0 such that $f(n) \geq 0$ and $g(n) \geq 0$ for all $n \geq n_0$. Thus for $n \geq n_0$, $f(n) + g(n) \geq f(n) \geq 0$ and $f(n) + g(n) \geq g(n) \geq 0$. Since for any particular n, $h(n)$ is either $f(n)$ or $g(n)$, we have $f(n) + g(n) \geq h(n) \geq 0$, which shows that $h(n) = \max(f(n), g(n)) \leq c_2(f(n) + g(n))$ for all $n \geq n_0$ (with $c_2 = 1$ in the definition of Θ).

Similarly, since for any particular n, $h(n)$ is the larger of $f(n)$ and $g(n)$, we have for all $n \geq n_0$, $0 \leq f(n) \leq h(n)$ and $0 \leq g(n) \leq h(n)$. Adding these two inequalities yields $0 \leq f(n) + g(n) \leq 2h(n)$, or equivalently $0 \leq (f(n) + g(n))/2 \leq h(n)$, which shows that $h(n) = \max(f(n), g(n)) \geq c_1(f(n)+g(n))$ for all $n \geq n_0$ (with $c_1 = 1/2$ in the definition of Θ).