CS6033 Design and Analysis of Algorithms I
Homework Assignment 1
09.17.2022

Group Members:
rk4305 (Sai Rajeev Koppuravuri)
sg7372 (Sriharsha Gaddipati)
vt2184 (Veeravenkata Raghavendra Naveenkumar Tata)
vt2182 (Venu Vardhan Reddy Tekula)

1. (5 points) What is the smallest value of n such that an algorithm whose running time is $100n^2$ runs faster than an algorithm whose running time is $2n$ on the same machine?

   Ans. n_0 when $100n^2 >= 2n$

   $100n^2 >= 2n$

   $50n >= n$

   $n >= 1/50$

   $n >= 0.02$

   Therefore, smallest value of n, $n_0 = 0.02$.

2. (15 points) You have 5 algorithms, A1 takes $O(n^3)$ steps, A2 takes $\Theta(n^2)$ steps, A3 takes $\Omega(n^2)$ steps, A4 takes $\Omega(n)$ steps, A5 takes $o(n)$ steps. For the following functions write down which all algorithms out of the 5 can be associated with it.

   (a) (3 points) $5n \log_2 n + 10 \log_2(\log_2 n)$
   Ans. Let's say $f(n) = 5n \log_2 n + 10 \log_2(\log_2 n)$. Since, we can ignore the lower order terms, the time complexity of $f(n)$ is $\Theta(n \log n)$.
   A1, A4 satisfy the condition.

   (b) (3 points) $12n!$'
   Ans. Let's say $f(n) = 12n!$.
   $f(n) = n!$
   $= n * (n - 1_*(n - 2) * ... * (n - (n - 1))$
   $= c_0 * n^0 + c_1 * n^1 + ... + c_n * n^n$ where $c_0, c_1...c_n$ are constants
   The time complexity of $f(n)$ is $\Theta(n^n)$ as we can ignore the lower order terms.
   A3, A4 satisfy the condition.

   (c) (3 points) $(\log_2 n)^2 + 32$
   Ans. Let's say $f(n) = (\log_2 n)^2 + 32$. The time complexity of $f(n)$ is $\Theta((\log n)^2)$.
   A1, A5 satisfy the condition.

   (d) (3 points) $2^{2 \log_2 n}$
   Ans. Let's say $f(n) = 2^{2 \log_2 n}$.

$f(n) = 2^{2\log_2 n}$
$= 2^{\log_2 n^2}$
$= n^2$
The time complexity of $f(n)$ is $\Theta(n^2)$.
A1, A2, A3, A4 satisfy the condition.

(e) (3 points) $2\log_8 n$

Ans. Let's say $f(n) = 2\log_8 n$.
$f(n) = 2\log_8 n$
$= 2\log_2^3 n$
$= 2/3\log_2 n$
The time complexity of $f(n)$ is $\Theta(\log_2 n)$.
A1, A5 satisfy the condition.

3. (10 points) You are a part of the organizing committee for a basketball tournament, where you are handling the distribution of jerseys for each team. You think it would be easy to distribute the jerseys if they have been sorted in batches for each team.

   You have two assistants under you, who provide you with an almost sorted batch each. Assistant A is new to the job and has been providing you with randomly ordered jerseys. Assistant B is a professional and provides you with almost sorted jerseys with utmost 1-2 jerseys out of place.

   You, being the supervisor, want to make sure that each batch is completely sorted and decide to perform a re-sort on each batch yourself.

   (a) (6 points) Which algorithm would you choose to sort the batch from assistant A and assistant B respectively, to make sure your sorting time is efficient?

   Ans.

   Assistant A $\rightarrow$ Merge Sort Algorithm as the elements will not be in sorted order because the assistant provides the jerseys in random order.

   Assistant B $\rightarrow$ Insertion Sort Algorithm as almost all the jerseys are sorted. So, it takes only less number of operations to keep all the jerseys in sorted order.

   (b) (4 points) Write down the average case time complexity in Theta(Θ) notation for each algorithm you choose for the given scenario.

   Ans.

   Assistant A $\rightarrow$ Merge Sort, Avg. Time Complexity = $\Theta(n \log n)$

   Assistant B $\rightarrow$ Insertion Sort, Avg. Time Complexity = $\Theta(n^2)$

4. (10 points) Consider a scenario where all the elements in an array of length n are the same. Eg: n = 5, A = [1, 1, 1, 1, 1]

   (a) (5 points) What will be the run time of insertion sort in this scenario?

   Ans. $\Theta(n)$

(b) (5 points) Lets say we call the PRINT(”DAA”) subroutine inside the inner while loop of insertion sort, which just prints the string ”DAA”. Mention the number of times the string ”DAA” will be printed.

Ans. 0 times as the elements are already in sorted order. So, the inner loop doesn’t execute.

5. (10 points) Using Figure 2.2 (Given in CLRS book) as a model, illustrate the operation of INSERTION SORT on the array A = (31, 41, 59, 26, 41, 15)

Ans.

Algorithm:

1. The first element in the array is assumed to be sorted. Take the second element and store it separately in key. Compare key with the first element. If the first element is greater than key, then key is placed in front of the first element.

2. Now, the first two elements are sorted.
Take the third element and compare it with the elements on the left of it. Placed it just behind the element smaller than it. If there is no element smaller than it, then place it at the beginning of the array.

3. Similarly, place every unsorted element at its correct position.

Solution:

Initial array
$\rightarrow$ (31, 41, 59, 26, 41, 15)

The next element is 41, key is 41. Since it is greater than 41, there won’t be any swapping.
$\rightarrow$ (31, 41, 59, 26, 41, 15)

The next key is 59 and it is greater than 31 and 49. No swapping again.
$\rightarrow$ (31, 41, 59, 26, 41, 15)

The next key is 26. Since 26 is less than 59, it gets swapped. It further keeps swapping till it is placed in the correct position in the array.
$\rightarrow$ (31, 41, 59, 26, 41, 15)
$\rightarrow$ (31, 41, 26, 59, 41, 15)
$\rightarrow$ (31, 26, 41, 59, 41, 15)
$\rightarrow$ (26, 31, 41, 59, 41, 15)

The next element is 41, it gets swapped with 59 to be correctly placed.
$\rightarrow$ (26, 31, 41, 59, 41, 15)
$\rightarrow$ (26, 31, 41, 41, 59, 15)

The last element is 15, it gets swapped iterative till it gets its correct place.
$\rightarrow$ (26, 31, 41, 41, 59, 15)
$\rightarrow$ (26, 31, 41, 41, 15, 59)
$\rightarrow$ (26, 31, 41, 15, 41, 59)
$\rightarrow$ (26, 31, 15, 41, 41, 59)
$\rightarrow$ (26, 15, 31, 41, 41, 59)
$\rightarrow$ (15, 26, 31, 41, 41, 59)

The sorted array is (15, 26, 31, 41, 41, 59).

6. (20 points) You are interested in starting a club at NYU. You have been seeking suggestions from your connections. You kept adding the suggestions to a list as they came. Now you have a lot of duplicate suggestions and would like to clean your list by removing the duplicates.

   (a) (10 points) Write an algorithm/pseudo-code to do so that runs in $O(n^2)$ time.

   Ans.

```
sg_list = ['sgstn1', 'sgstn2',...]
unq_sg_list = []
n = len(sg_list)

flag = [True for i in range(n)]

for i in range(n):
    if flag[i] is True:
            for j in range(i+1, n):
                    if (sg_list[i] == sg_list[j]):
                            flag[j] = False
for i in range(n):
    if flag[i] == True:
            unq_sg_list.append(sg_list[i])

print(unq_sg_list)
```

   Total number of operations performed in each iteration
   $= (n-1) + (n-2) + (n-3) + ... + 1$
   $= (n-1) * (n)/2$

   Time complexity of above code $= O(n^2)$

   (b) (8 points) Also prove the correctness of the algorithm you used above using loop invariant.

   Ans.

   In outer loop, It stores all the unique suggestions for first i suggestions.

   In inner loop, It marks flag[i] as false for all the repeated suggestions found in [i+1, n]. Making it as false, so that those elements does not get process as they are duplicates. This way, by the end of the outer loop, all the duplicates are marked as false and unique elements as true.

   (c) (2 points) Which algorithm can you use to make your task more efficient and what will be the reduced complexity?

   Ans.

```
sg_list = ['sgstn1', 'sgstn2',...]
ung_sg_list = {}
n = len(sg_list)
```

```
for i in range(n):
    if sg_list[i] not in ung_sg_list:
            ung_sg_list(arr[i]) = 1
    else:
            ung_sg_list(arr[i]) += 1

return ung_sg_list.keys()
```

Time complexity of the above code is $O(n)$.

7. (10 points) Solving Recurrences give asymptotic upper and lower bounds for $T(n)$ in each of the following recurrences. Assume that $T(n)$ is constant for $n <= 2$. Make your bounds as tight as possible and justify your answers.

   (a) (2 points) $T(n) = 4T(n/4) + 5n$

   Ans.

   $T(n) = 4T(n/4) + 5n$
   here, $a = 4$, $b = 4$ and $f(n) = 5n$
   let, $g(n) = n^{\log_b a}$
   $g(n) = n^{\log_4 4}$
   $g(n) = n$
   As $f(n) = \Theta(g(n))$, $T(n) = \Theta(n^{\log_b a} * \log n)$
   $T(n) = \Theta(n^{\log_4 4} * \log n)$
   $T(n) = \Theta(n \log n)$

   (b) (2 points) $T(n) = 4T(n/5) + 5n$

   Ans.

   $T(n) = 4T(n/5) + 5n$
   here, $a = 4$, $b = 5$ and $f(n) = 5n$
   let, $g(n) = n^{\log_b a}$
   $g(n) = n^{\log_5 4}$
   As $f(n) >= g(n)$, $T(n) = \Theta(n)$

   (c) (2 points) $T(n) = 5T(n/4) + 4n$

   Ans.

   $T(n) = 5T(n/4) + 4n$
   here, $a = 5$, $b = 4$ and $f(n) = 4n$
   let, $g(n) = n^{\log_b a}$
   $g(n) = n^{\log_4 5}$
   As $g(n) >= f(n)$, $T(n) = \Theta(n^{\log_b a})$
   $T(n) = \Theta(n^{\log_4 5})$

   (d) (2 points) $T(n) = 25T(n/5) + n^2$

   Ans.

$T(n) = 25T(n/5) + n^2$
here, $a = 25$, $b = 5$ and $f(n) = n^2$
let, $g(n) = n^{\log_b a}$
$g(n) = n^{\log_5 25}$
$g(n) = n^{\log_5 5^2}$
$g(n) = n^{2\log_5 5}$
$g(n) = n^2$
As $f(n) = \Theta(g(n))$, $T(n) = \Theta(n^{\log_b a} * \log n)$
$T(n) = \Theta(n^{\log_5 25} * \log n)$
$T(n) = \Theta(n^2 \log n)$

(e) (2 points) $T(n) = 4T(n/5) + \log_2 n$

Ans.
$T(n) = 4T(n/5) + \log_2 n$
here, $a = 4$, $b = 5$ and $f(n) = \log_2 n$
let, $g(n) = n^{\log_b a}$
$g(n) = n^{\log_5 4}$
As $n^{\log_5 4}$ grows faster than $log_2 n$
$T(n) = \Theta(n^{\log_5 4})$

8. (20 points) Merge sort runs in $\Theta(n \log_2 n)$ worst-case time and insertion sort runs in $\Theta(n^2)$ worst-case time. But the constant factors in insertion sort can make it faster in practice for small problem sizes on many machines. Thus, it makes sense to coarsen the leaves of the recursion by using insertion sort within merge sort when sub-problems become sufficiently small.

Consider a modification to merge sort in which (n/k) sub-lists of length k are sorted using insertion sort and then merged using the standard merging mechanism, where k is a value to be determined.

(a) (5 points) Show that insertion sort can sort the n/k sublists, each of length k, in $\Theta(nk)$ worst-case time.

Ans. Time complexity of insertion sort on each sub-list of length $k = \Theta(k^2)$
In total, we had n/k sub-lists and so insertion sort needs to be repeated for n/k times
$= (n/k) * k^2$
$= (n * k)$
Time complexity = $\Theta(nk)$

(b) (5 points) Show how to merge the sublists in $\Theta(n \log(n/k))$ worst-case time.

Ans. Here, we had (n/k) sublists with each of length k.
$2^i = n/k$
$i = \log_2(n/k)$
Add all the overheads
$= n + n + ... + (\log_2(n/k) times)$
$= n * \log_2(n/k)$

(c) (10 points) Now given that the modified algorithm runs in $\Theta(nk+n\log_2(n/k))$ worst-case time, what is the largest value of k as a function of n for which the modified algorithm has the same running time as standard merge sort, in terms of Θ - notation?

Ans.

In order to prove, $\Theta(nk + n\lg(n/k))$ = $\Theta(n\lg(n))$
Either nk = $\Theta(n\lg(n))$, or $n\lg(n/k))$ = $\Theta(n\lg(n))$
In first case k cannot grow faster than $\Theta(\lg(n))$
In second case, k cannot grow faster than $\Theta(constant)$

Since, $\Theta(constant) < \Theta(\lg(n)))$
Largest possible value of $k = \Theta(\lg(n))$