Department of Computer Engineering, SVNIT, Surat

M Tech I - 1st Semester, Mid-Semester Examinations

CO 603: Algorithms and Computational Complexity

10:30 hrs to 12:00 hrs, 18th October 2021

Instructions:

1. *Write your Admission number clearly in the answer books along with the other details. Write page numbers on all pages that you use.*
2. *Assume any necessary data but giving proper justifications.*
3. *Be brief, precise, clear and to the point in answering the questions. Unnecessary elaboration WILL NOT fetch more marks.*
4. *Maximum marks = 30, Duration = 1.5 hrs. Each MAIN question carries 10 marks, maximum.*
5. *All sub-questions carry equal marks unless stated otherwise.*

1. (a) If T(n)=2^{10n}, then T(n)= $O(2^n)$? If T(n)=2^{10+n}, then T(n)= $O(2^n)$? Your answer must be supported with a formal proof. [2]

Answer: As per the definition of big-O notation, for T(n) = $O(2^n)$, we must have a suitable pair of positive constants c and n_0 (each independent of n), such that T(n) $\leq c * 2^n$ for all n $\geq n_0$.

(a) In this case, we are given that T(n)=2^{10n}, and we have to prove whether T(n)= $O(2^n)$ is true or not. Let us assume that

$$T(n) = O(2^n)$$

By the definition of big-O notation, this means there are positive constants c and n_0 such that

$$2^{10n} \leq c * 2^n$$

for all n $\geq n_0$. Now, because 2^n is a positive number, we can cancel it from both sides of this inequality to derive

$$2^{9n} \leq c$$

for all $n \geq n_0$. But this inequality is patently false: the right-hand side is a fixed constant (independent of n), while the left-hand side goes to infinity as n grows large. This shows that our assumption that T(n) = $O(2^n)$ cannot be correct, and we can conclude that T(n)= 2^{10n} is not $O(2^n)$

(b) Now, we are given that T(n)=2^{n+10} $= 2^{10} \;\; 2^n = 1024 * 2^n$. Therefore, we can assume c=1024 and the value of n_0=1, that gives us

$$T(n) = 2^{10+n} \leq c * (2^n)$$

for all n $\geq n_0$=1 and hence T(n)=O(2^n)

(b) For non-negative real-valued functions $f(n)$ and $g(n)$ defined on the positive integers, with f(n) and g(n) at least 2 for all n, and also given that f(n)=O(g(n)), and c be a positive constant, then is f(n) * lg ($f(n)^c$) = O(g(n) $* lg(g(n))$)? [4]

Answer: It is clear from the observation viz. that if T(n)=2^{10n}, then T(n) is not $O(2^n)$. That is, multiplying the exponent of an exponential function by a constant changes its asymptotic rate of growth. This has been proved in the answer to the question 1(a). Hence, the asymptotic notation that applies before the exponent of an exponential function is multiplied by a constant may not apply to that obtained after multiplication. Therefore, if f(n)=O(g(n)), and c be a positive constant, then f(n) * lg ($f(n)^c$) $\neq$ O(g(n) $* lg(g(n))$)

(c) Consider an integer vector X of size n, denoted as X $= x_1, x_2, x_3, ...x_n$, given as input to the Quicksort algorithm. Consider also a sorted instance of this vector denoted as L$= \ell_1, \ell_2, \ell_3, ...\ell_{i-1}, \ell_i, \ell_{i+1}, ...\ell_{j-1}, \ell_j, \ell_{j+1}, ...\ell_n$. Now with respect to the given data, answer the following questions: [4]

(i) If the number of comparisons made by the algorithm, each comparing the two elements x_i and x_j is denoted by $\sum_{j=i+1}^{n} \sum_{i=1}^{n-1} X_{ij}$, and the expected value of a random variable A is denoted by E(A), then what is the expression for the expected value of the average number of comparisons made ?
Answer: Given the data as above, the expression that denotes the expected value of the average number of comparisons made is

$$\sum_{j=i+1}^{n} \sum_{i=1}^{n-1} E(X_{ij})$$

(ii) If P_{ij} is the probability that the two elements ℓ_i and ℓ_j are compared, then in terms of P_{ij}, what is the value of the average number of comparisons made ?
Answer: If P_{ij} is the probability that the two elements ℓ_i and ℓ_j are compared, then in terms of P_{ij}, the expected value of the average number of comparisons made is as follows:

$$E(X_{ij}) = P_{ij} * 1 + (1 - P_{ij}) * 0 = P_{ij}$$

.

Then, the the expected value of the average number of comparisons made is given by

$$\sum_{j=i+1}^{n} \sum_{i=1}^{n-1} P_{ij}$$

(iii) In the given sorted instance L, when would the two keys ℓ_i and ℓ_j, be compared ?
Answer: The two keys ℓ_i and ℓ_j, will be compared, if either ℓ_i or ℓ_j is picked up first as the partitioning element, before any of the elements$\ell_{i+1}, ...\ell_{j-1}$ are picked up as the partitioning elements.

(iv) In terms of 1(c)(iii) above, what is the average number of comparisons made by the quicksort algorithm? How ?

2. (a) Consider the following problem: You are incharge of the MIS department at an institute. The MIS department has a processor that can operate 24 hours a day, every day. People submit requests to run daily jobs on the processor. Each such job comes with a start time and an end time within the 24 hours of one day only. And, if the job is accepted to run on the processor, it must run continuously, every day, for the period between its start and end times. Note that those jobs that are specified with the times that begin before midnight and end after midnight - are not accepted - that is, a job must start and complete within a specific date only - which means that the date can be ignored.
Given a list of n such jobs, your goal is to accept as many jobs as possible (regardless of their length), subject to the constraint that the processor can run at most one job at any given point in time. Design an algorithm with a running time that is polynomial in n, and write its pseudocode. You may assume for simplicity that no two jobs have the same start or end times. An Example: Consider the following four jobs, specified by (start-time, endtime) pairs. (6 P.M., 11 P.M.), (9 A.M., 4 P.M.), (3 A.M., 1 P.M.), (2 P.M., 6 P.M.). Then, the optimal solution would be to pick the three jobs, which can be scheduled without overlapping. [2]
Answer: Given the data and the context of the problem as in the question, let the formal representation of the problem be as follows:
We assume that we are given n jobs each with a defined start time s_i and finish time f_i, with the assumption that the start time and finish time is within 24 hours a single day - it does not spill over the midnight.

The goal is is to select a maximal set of mutually compatible jobs, subject to the constraint that the processor can run at most one job at any given point in time.
Then, the pseudocode of the algorithm is as follows:

```
Algorithm Simple_JobSelection(Job, $s_i$, $f_i$)
/*A=Set of selected mutually compatible jobs*/
1.Sort jobs by finish times so that $f_1 \leq f_2 \leq \ldots . \leq f_n$.
2.A ← $\phi$
3.for j = 1 to n
4.      if $f_j \leq s_{(j+1)}$ /*(job (j+1) is compatible with A)*/
5.      A = A ∪ {j}
6.return Selected_Jobs
```

The running time of this algorithm is $O(n\ lg\ n) + O(n) i.e. O(n\ lg\ n)$ as it does not require any major operation apart from sorting.

(b) Give a formal proof on the correctness of the algorithm that you designed in 2(a). [3]
Answer: The formal proof of the algorithm above is given below using the principle of mathematical induction:

- For the job scheduling problem, let the set $A = i_1, i_2, i_3, ...i_k$ with $|A| = k$ be the set of jobs returned by the algorithm as the answer and
- Let the set $O = j_1, j_2, j_3, ...j_m$ be the optimal set of jobs that would be returned by an oracle.
- Then, to prove that our job scheduling algorithm works and is optimal, our goal is to prove that k = m.
- Since our algorithm picks up the jobs in the non-decreasing order of their finish times, this is equivalent to proving, that $f_{i_r} <= f_{j_r}$ i.e. to prove that the r^{th} accepted request in the algorithm's finishes no later than the r^{th} request in the optimal schedule. This means we are actually proving that our greedy algorithm stays ahead. We prove so using the principle of mathematical induction as in the following:
- **Basis**: This step requires us to prove that $f_{i_1} \leq f_{j_1}$. Since, our algorithm picks up the jobs in the non-decreasing order of their finish times, it is clear that the finish time of the first job picked up by our algorithm i.e. f_{i_1} is less than or equal to that of any other job in the job mix and so, $f_{i_1} \leq f_{j_1}$. Proved.
- **Inductive Hypothesis**: In this step we assume that the goal to be proved is true for all the jobs except for the rth job i.e. $f_{i_{(r-1)}} \leq f_{j_{(r-1)}}$
- **To prove:** $f_{i_r} <= f_{j_r}$.
- **Proof**:
  - We know that, $f_{j_{(r-1)}} <= f_{j_r}$. This is so, because the set O is the set of the optimal requests and so the last job selected therein must be compatible with the one selected prior to that. .....(1).
  - Also, from the inductive hypothesis, we know that $f_{i_{(r-1)}} \leq f_{j_{(r-1)}}$. ......(2)
  - From (1) and (2), and the principle of transitivity, we can deduce that $f_{i_{(r-1)}} \leq f_{j_r}$
  - This implies that the job $f_{i_{(r-1)}}$ is mutually compatible with the job f_{j_r}
  - What this further implies is that in the job mix that is used prior to the selection of the i_r^{th} job, the job j_r is also in consideration for being picked up.
  - That is, note that the job j_r - along with job i_r - is available for selection to the algorithm. Remember that i represents the set of jobs chosen by the algorithm.
  - How does our algorithm select a job ?
  - Well, the algorithm selects only that job whose finish time is the least i.e. amongst $f(i_r)$ and $f(j_r)$. And we know our algorithm has selected the job $f(i_r)$. Since the algorithm has selected i_r, its finish time i.e. $f(i_r)$ must be lesser. Therefore, $f_{i_r} <= f_{j_r}$. Therefore proved.

(c) Given the Mergesort recurrence by expressions viz.
$\rightarrow T(n) \leq 0$ if n=1 and
$\rightarrow T(n) \leq T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n$, otherwise,
that does not assume n to be a power of 2, prove that the solution to the recurrence, still is T(n) = O(n lg n). [5]
Answer: This proof was done explicitly in the class and so refer to your classnotes.

3. (a) A small business say, a photocopying service with a single large machine faces the following scheduling problem: each morning, they get a set of jobs from customers. They want to do the jobs on their single machine in an order that keeps their customers happiest. The customer i's job will take t_i time to complete. Given a schedule (i.e. an ordering of the jobs), [6]

- let C_i denote the finishing time of job i. For example, if job j is the first to be done, we would have $C_j = t_j$ and if job j is done right after job i, we would have $C_j = C_i + t_j$.
- Each customer i also has a given weight w_i that represents his or her importance to the business. The happiness of customer i is expected to be dependent on the finishing time of the customer i's job.
- Hence, the company decides that they want to order the jobs to minimize the weighted sum of the completion times, i.e. to minimize $\sum_{i=1}^{n} w_i * t_i$

Design an efficient algorithm to solve this problem and analyse its complexity, giving logical arguments. Consider that, you are given a set of n jobs with processing time t_i, and weight w_i, for each job. You want to order the jobs so as to minimize the weighted sum of the completion times, $\sum_{i=1}^{n} w_i * t_i$.
Answer: Note that in this problem we have to select a job ordering such that it schedules the jobs to minimize the weighted sum of the completion times, i.e. to minimize $\sum_{i=1}^{n} w_i * t_i$.

- To design the algorithm let us take an example first. Let there be two jobs. The first takes time t_1=1 and has weight w_1=10; similarly for the second job t_2=3 and w_2=2. Then doing job 1 first would yield a weighted completion time of 10*1+2*4=18; whereas doing second job first would yield the larger weighted completion time of 10*4+2*3=48.
- Therefore this problem appears to be similar to the shortest job first scheduling. However, since the weight w_i, for each job may not strictly be in sequentially increasing order, the same approach as in SJF scheduling will not apply.
- As a greedy criterion since we want to maximize weight while the job completes as soon as possible, let us compute the value of w_i/t_i. Then applying it to the example, we have w_1/t_1 = 10/1 = 10 and w_2/t_2 = 2/3 = 0.66. We obtained the smaller weighted completion time of 18 when we scheduled the jobs in the non-increasing order of the value of w_i/t_i. We can check it with other examples and observe that this approach would apply.
- Therefore the algorithm would be written as follows:

```
1. for i=1 to n
2.  x[i] = w[i]/t[i]
3. index ← ALLOCATE_MEMORY(n)
3. IndirectSort(n, x[i], index[i] )
4. for i=1 to n
5.       output i.e schedule  job with
                    s[index[i]], f[index[i]], weight[index[i])
6. delete index
```

- In this algorithm, the sorting function assigns the indices to the variable index[i] such that index[1] contains the numeric index of the job with maximum w[i]/t[i], index[2] points to the job with next higher w[i]/t[i] and so on
- The complexity of this algorithm is O(n) + O(n lg n) + O(n) i.e. O(n log n).

(b) Consider the following problem: Prof M S Gaur, the Director, IIT Jammu, drives an automobile from Jammu to Delhi along NH1A and then onwards. His car's petrol tank, when full, holds

enough petrol to travel n kms and his map gives the distances between petrol stations on his route. The professor wishes to make as few petrol station stops as possible along the way. Assume that an efficient method, based on the Greedy design technique using which Professor Gaur can determine, at which minimal number of petrol stations should he stop is given and is as shown in the box, below.
Then, show using formal notations, how does this solution exhibits **the Optimal substructure and the Greedy choice properties** ? [4]
You can use the following notations:

- m possible gas stations,
- the first stop is the k^{th} gas station, i.e. there are k gas stations beyond the start within n kilometres of the start and the greedy solution chooses the k^{th} station as its first stop.
- an optimal solution has s gas stations, say...

Starting will a full tank of petrol, Professor Gaur should go to the farthest gas station he can get to within n kilometres of Jammu. Fill up there. Then go to the farthest gas station further he can get to, within n kilometres of where he last filled up, and fill up there, and so on. Looked at another way, at each petrol station, Professor Gaur should check whether he can make it to the next petrol station without stopping at this one. If he can, skip this one. If he cannot, then fill up. Professor Gaur doesn't need to know how much petrol he has or how far the next station is to implement this approach, since at each fillup, he can determine which is the next station at which hell need to stop.

Answer: We are already given the algorithm used by Prof Gaur to stop at the petrol stations. We only have to show that the proposed approach exhibits the **the Optimal substructure and the Greedy choice properties**. This can be argued as follows:

- The problem has optimal substructure.
  - As per the notations given in the problem, suppose there are m possible petrol pump stations. Consider an optimal solution with s stations and whose first stop is at the kth petrol pump station.
  - Then the rest of the optimal solution must be an optimal solution to the subproblem of the remaining $m - k$ stations.
  - Otherwise, if there were a better solution to the subproblem, i.e., one with fewer than $s - 1$ stops, we could use it to come up with a solution with fewer than s stops for the full problem, contradicting our supposition of optimality.
- This problem also has the greedy-choice property.
  - Suppose there are$k$ petrol pump stations beyond the start that are within n kilometres of the start. The greedy solution chooses the k^{th} station as its first stop.
  - No station beyond the k^{th} works as a first stop, since Professor Gaur runs out of petrol first. If a solution chooses a station $j < k$ as its first stop, then Professor Gaur could choose the k^{th} station instead, having at least as much petrol when he leaves the k^{th} station as if hed chosen the j^{th} station.
  - Therefore, he would get at least as far without lling up again if he had chosen the k^{th} station. If there are m petrol pump stations on the map, Prof Gaur needs to inspect each one just once.

***paper ends here**