End Sem- Solutions & Rubrics

Q2:
Rubrics:
Favorite algorithm/idea - 1.5 mark
Reason - 2 marks
Most surprising algorithm/idea - 1.5 mark

Q3 solution:
1. **sqrt(n)**
2. **2^sqrt(logn)**
3. **(2^2^sqrt(log(log n)))**
4. **2^n**
5. **n**
6. **n.logn**
7. **n.logn.loglogn**

Increasing Order: 3,2,1, 5,6,7,4
Rubrics:
1. Correct answer - 4 marks
2. Incorrect answer. No partial marks - 0 marks

Q4 Solution:

Date ....

$$T(n) = T(n/2) + T(2n/5) + 7n$$

Using tree method, we know $T(n/2)$ branch will have more length. So for O notation.

$$T(n) \Rightarrow 7n$$

$$T(n/2) \quad T(2n/5) \Rightarrow \frac{7n}{2} + \frac{14n}{5} = \frac{63n}{10} = 7\left(\frac{9n}{10}\right)$$

$$T(n/4) \quad T(-) \Rightarrow 7\left(\frac{81n}{100}\right)$$

$$T(1)$$

$\log_2^n$ levels

Clearly the sum of all cost of each level for $\log_2^n$ is worst case T.C

$$T.C = \sum_{\log_2 n} 7n + 7\left(\frac{9n}{10}\right) + 7\left(\frac{9^2}{10^2}n\right) \dots 7\left(\frac{9^{\log n}}{10^{\log n}}n\right)$$

$$T.C = 7\,(\text{some decreasing geometric series})$$
$$= O(n)$$

If ass Sim

Similary for $\log_{5/2} n$ levels will give best case T.C which will also be $\Omega(n)$

$\therefore$ Θ (theta) T.C is also $O(n)$

~~Rubrics:~~

~~Correct solution: 5 marks~~

~~Partially correct: 3 marks~~

~~T.C correct : 1 marks~~

~~Incorrect : 0 marks~~

Rubrics:

Correct : 5 marks

Upper bound : 2.5 marks

Lower bound : 2.5 marks

Incorrect : 0 marks

<u>Q5 solution:</u>

We need to apply binary search and need to change the check conditions.

At position mid,
If a[mid-1]<a[mid]<a[mid+1] then we are in increasing part of array. So change the lower bound to mid.

Else if a[mid-1]>a[mid]>a[mid+1] then we are in decreasing part of array. So change the upper bound to mid.

Else we have the answer.

Rubrics:

1. O(n) solution and correctness proof - 3 marks.
2. Correct O(logn) solution and correctness proof. - 8 marks.

Q6.

<u>Solution:</u>

1. Number of bits is $O(\log_2 n)$
2. Time complexity of algorithm is $O(\sqrt[3]{n})$ if you don't take the time for arithmetic operations into account otherwise O(n^⅓*(log n)^2) or some poly(log n) factor along with n^⅓.
3. No it is not an efficient algorithm
4. A suitable proof is expected. Just to give a brief overview, for all n that can represented as product of three numbers (all greater than or equal to 2), the algorithm will traverse from 2 to $\sqrt[3]{n}$ to check the divisibility. For any such number n (n=a*b*c) even if we assume two of its factors to be greater than $\sqrt[3]{n}$ still the last remaining factor has to be less than $\sqrt[3]{n}$ to maintain the product equal to n. At max (in case n is a cube of some prime number) we might need to traverse till its cube root to find if there exists any prime factor or not.
5. Let's say n=35. It is a composite number with 1,5,7,35 as its factors.
   The algorithm will run from i=2 to i=4 (as 4*4*4=64 which is greater than 35). For all values of i between 2 to 4 (i.e. 2, 3 and 4) none of them divides 35. Finally at i=4 the loop is exited and the algorithm outputs -1.
   Much more similar cases are possible.

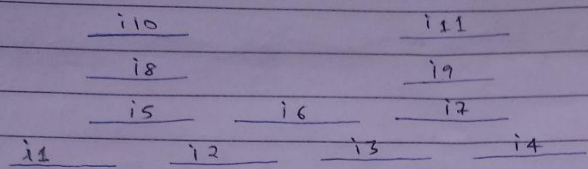
Grading Rubric:

1. (1 marks) $O(\log_2 n)$ or $\log_2 n$
   (0 marks) for any other answer
2. (2 marks) $O(\sqrt[3]{n})$ or $O(n^{1/3} * (\log n)^2)$
   (0 marks) for any other answer
3. (1 marks) For answer NO
   (0 marks) for any other answer
4. (4 marks) Suitable proof and explanation
   (1 marks) shall be deducted if the case where loop will traverse till $\sqrt[3]{n}$ is not discussed.
   (1-2 marks) shall be deducted for improper explanation.
5. (2 marks) Correct example and explanation
   (1 marks) 1 marks shall be deducted if the counterexample case is not properly explained.
   (0 marks) otherwise

Q7. solution:

Interval Scheduling :→

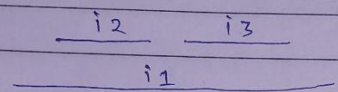
(1.) Ans.



Output of given algorithm = { i_6, i_1, i_4 }

Optimum = { i_1, i_2, i_3, i_4 }

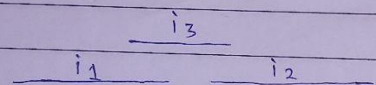
(2.) Ans.



Output of given algorithm = { i_1 }

Optimum = { i_2, i_3 }

(3.) Ans.



Output of given algorithm = { i_3 }

Optimum = { i_1, i_2 }

(4.) Ans. Algorithm :→

1. Sort the intervals in S according to the end time e.
~~2. for every consecutive interval~~

2. Add the first interval to S'.
3. for every consecutive interval {
   (a.) If the start time of current interval is after ~~the~~ ($\geq$) the last selected interval then add it to S'. otherwise skip it.
   (b.) Go to the next interval.

}

Date:

YOUVA

(E) Ans: To prove the correctness of our algorithm, we will compare the ~~ot~~ output of our algorithm and any optimum solution.

Let $ALG = \{i_1, i_2, ---, i_k\}$ be the set of output given by our algorithm.

Let $OPT = \{j_1, j_2, ---, j_m\}$ be the optimum set, ordered by their ~~fin~~ end time.

Our goal is to prove $k = m$ i.e. $|ALG| = |OPT|$

Step 1 :- First we will show that for $1 \leq r \leq k$, $f(i_r) \leq f(j_r)$

$f(i)$ denote end time of i.
$s(i)$ denotes the start time of i.

Proof :- We will prove it by induction.

Base Case, we take $r = 1$. Since we selected the interval with the earliest end time, it certainly must be the case that $f(i_1) \leq f(j_1)$

For $r > 1$, assume the statement is true for $r-1$ and we will prove it for r. The induction hypothesis states that $f(i_{r-1}) \leq f(j_{r-1})$, ~~and~~ So, any interval that are compatible with the first $r-1$ intervals in the OPT are ~~cer~~ certainly compatible with the first $r-1$ intervals of ALGO. Therefore, we could add j_r to our ALGO soln. and since we take the compatible job with the smallest end time, it must be the case that $f(i_r) \leq f(j_r)$.

Step 2 :- The algorithm is optimal i.e., $k = m$.

Proof by Contradiction :-

If the ALG is not optimal, it means $m > k$ and there is a interval j_{k+1} in OPT and j_{k+1} is not in ALG. This request must start after interval j_k ends and hence after i_k ends. But then this interval is compatible with all the intervals in ALG, ~~and~~ So, ALG would have added it ~~during~~ before termination. This is a contradiction and it proves that $m = k$ i.e. the algorithm is correct.

Running time Analysis :- Assume n intervals.

Sorting + Checking each interval (1 for loop)

$O(n \log n) + O(n)$

$= O(n \log n)$

Grading Rubrics :-

Part 1 : 2 Marks
Part 2 : 2 Marks
Part 3 : 2 Marks.
Part 4 : 4 Marks.
Part 5 : 4 Marks (3 Marks Proof + 1 Mark for Time Complexity)

Q8 solution:

1) By max flow min cut theorem, there exists a cut in graph G of value f. There are two cases e* belongs to this cut or not. If e* belongs to this cut then min cut in G' might increase by at most 1. Hence, f' is at most 1 greater than f by max flow min cut theorem for G'.
2) Draw residual graph with respect to flow f and graph G. start with s and check the maximum possible flow from s to t with following constraints:
a) Flow on an edge doesn't exceed the given capacity of the edge.
b) Incoming flow is equal to outgoing flow for every vertex except s and t.
It will need one time iteration so time complexity will be O(n+m)

3) From the Ford-Fulkerson algorithm, we know that if a flow is not maximum, then there exists a positive path from s to t in the residual graph. From one, it is sure that we have to find such a path in the residual graph at most once. And finding a path takes O(n+m) time

Rubrics

1) For all part correct -> 3 ,partial-> 1 and 2

Q9 solution:

a) For the first part, a feasible flow of value n is obtained by pushing a flow of value 1 on every edge from s to a vertex in L, value 1 on every edge from a vertex in R to t, and 1/d on every edge between L and R. Check that this is a valid flow and satisfies the capacity constraints on every edge and conservation constraints on every vertex.

b) For the second part, we note that for the given flow network, all edge capacities are integral. Moreover, we know that there is a flow of value n as shown in the earlier part. In fact, this is also the maximum flow since s has n edges going out, and each has capacity 1, so flow can never exceed n. Now, by the max-flow min-cut theorem that we saw in class, we get that there is an integral flow in the network of value n. Any such integral flow must have all the edges going out of s to have flow value 1, all incoming edges in t to have flow value 1. Also, some of the edges between L and R will have value 0 and some will have value 1 (since the capacity is 1, and flow is integral). Just focus on the edges between L and R that have value 1. Now, the claim is that these edges must form a perfect matching.

To see this, note that in the max integral flow above, each vertex in L has incoming flow of value 1 from s, and each vertex in R has one unit flow going to t. Moreover, we know that edges between L and R carry flow of value 1 or 0. So, every vertex in L and R has exactly one such edge of flow value 1 incident to it, and this gives us a perfect matching.

Rubrics Q9)

a)

Correct : 3.5 marks

Generated correct flow: 1.5 marks

Capacity and constraint check verified: 2 marks

Incorrect : 0 marks

b)

Correct : 3.5 marks

Argument about integral flow: 2 marks

Argument about perfect matching from integral flow value of n : 1.5 marks

Incorrect : 0 marks

Q10 solution:

Rubrics:
Part 1

a. Algorithm (6 marks)

Part 2

a. Reduction from feasibility to optimization (2 marks)
b. Algorithm (4 marks)

Solutions should be in polynomial time
Proof of correctness is not required

Part 1

Proof of correctness:

Suppose F is satisfiable and truth values of any x_i in $\{x_1, \cdots x_n\}$ is y_i where ~~[illegible]~~ $y_i \in \{true, false\}$ or $y_i \in \{0, 1\}$

Then there exists atleast one assignment $y_1 \cdots y_n$ which makes F satisfiable.

Therefore, if we replace x_i with its truth value y_i, F should remain satisfiable otherwise it will contradict that $y_1 \cdots y_n$ gives a satisfying assignment.

We use this fact to devise our algorithm.

That is if by replacing x_i by y_i in F, F still remains satisfiable then there exists at least one satisfying assignment of F in which truth value of x_i is y_i. □

Running time

~~Since~~ We were given check_SAT takes constant time.

We run the loop for n iterations and it clearly visible that in each loop we perform constant time operations.

Thus, running time of the algorithm is $O(n)$. □

Note: Other variations are possible with polynomial running times.

Part 2: For the vertex cover question, we again have to output a vertex cover of minimum size and not just the size of the minimum vertex cover.

Here again, something like the SAT algorithm works. One way is to do the following.

1. Suppose you have found the size of the minimum vertex cover. Say, it is k*. This can be computed using binary search and calls to the decision subroutine.

2. Let V ={v_1, v_2, …, v_n}.

3. Initialization: k = k*, S be the empty set, G' = G (copy of the original graph)

4. For i = 1, 2, …, n,
    - Set G_1 = G'\{v_i}. (graph obtained by deleting v_i, and all its associated edges from G)
    - Check if G_1 has a vertex cover of size at most k-1, using the decision subroutine for VC
        - If 'yes', set k = k-1, S = S \union {v_i}, G' = G_1 and i = i+1 . Continue the loop
        - If 'no', set i = i+1. Continue the loop.

For the proof of correctness, the invariant in step 4 is the following: let G', k, S be the variables in the loop at the end of iteration i. Then, for every i, G' has a minimum vertex cover of size k, and for every vertex cover U of G' of size k, U \union S is a vertex cover of size at most k* of the original graph G.

The proof of this is via induction on i.