$f(n) = 2(f(n-1)) + 1$ $f(n) = 2^2 \cdot f(n-2) + 2$

$f(n-1) = 2 f(n-2) + 1$ $\vdots$ $f(n-n)$.

$\vdots$

$f(n) = 2^n + 1$.

Name: ____________________

Student ID: ____________________

1. [10 points]

$\cancel{f(n)=}$ $n(n-1) \cdots 3 \cdot 2 \cdot 1$

(a) Prove formally that $2^n = O(n!)$. $2^n \leq c\, n!$.

(b) You work for a company and one of your colleagues claims that he (or she) invented a new sorting algorithm whose running time is $f(n) = 16f(\frac{n}{16}) + \frac{1}{2}n$ where n is the size of input to the algorithm and that this algorithm is way better than all existing comparison based sorting algorithm (which is $n \log n$ at best) in terms of asymptotic running time. Is he or she right or wrong? State your answer and prove it formally.

(a). To show $2^n \leq c\,n!$ for some c, $\forall n \geq n_0$.

let $c=2$, $n_0 = 2$

$RHS = 2 \cdot n! = \underbrace{n(n-1)(n-2) \cdots 3 \cdot 2 \cdot 2}$

n terms, and ith term ≥ 2 $\forall i \in n$.

$\therefore RHS \geq \underbrace{2 \cdot 2 \cdots 2}_{n} = 2^n = LHS$

$\therefore 2^n = O(n!)$

(b). $f(n) = 16 \cdot f(\frac{n}{16}) + \frac{1}{2}n$.

$f(\frac{n}{16}) = 16 \cdot f(\frac{n}{16^2}) + \frac{n}{2 \times 16}$

$f(\frac{n}{16^2}) = 16 \cdot f(\frac{n}{16^3}) + \frac{n}{2 \times 16^2}$

$f(\frac{n}{16^3}) = 16 \cdot f(\frac{n}{16^4}) + \cdots$

$f(n) = 16^2 \cdot f(\frac{n}{16^2}) + \underbrace{\frac{1}{2}n + \frac{n}{2}}_{n}$

$= 16^3 \cdot f(\frac{n}{16^3}) + 16^2 \cdot \frac{n}{2 \times 16^2} + n$.

$\underbrace{}_{\frac{3}{2}n}$

$f(n) = 16^k \cdot f(\frac{n}{16^k}) + \frac{k}{2}n$ $\Rightarrow$ reduce n times. as $n \to \infty$

$f(n) = \frac{1}{2}n^2$ $f(\frac{n}{16^n}) \to 0$

$n \log n = O(f(n))$ $\therefore$ NO.

Name: ______________________

Student ID: ______________________

2. **[20 points]** Your millionaire-friend decide to open pizza-parlors on a particular stretch of highway from Los Angeles to Las Vegas, since he knows that there are no pizza restaurants on this deserted highway stretch, and many drivers who love pizza go through it. He wants to open at least one pizza-parlor within 10 mile distance of each gas-station on the highway in order to dominate the inferior food quality offerings at gas stations, and he wants to dominate that particular highway stretch with high quality pizza offerings. Since he heard that you are taking an algorithms class at UCLA, he asks you for an algorithm to places as few pizza-parlors as possible to cover each gas station.

   Explain the algorithm that you would use to decide where on the highway to place pizza restaurants for your rich friend. He says that he will accept your solution only if you could prove to him that it is an absolute minimum number of restaurants to cover all gas stations, so you should prove that as well.

[gas-10 , gas+10] .

Suppose there are n gas stations with coordinate on x-axis, $g_1, \dots, g_n$.

We define interval. $I_1 = [g_1-10, g_1+10]$, . . . , $I_n = [g_n-10, g_n+10]$.

We try to insert pizza house $P_1, \dots, P_K$ to intervals, so that there are at least one pizza house in each I, and # of pizza house is minimum.

(G) - Algorithm: for each pizza house P_i, we assign it to the earliest unmatche interval.

claim: if $\exists$ a optimal solution. our algorithm G can find it

Pf: [by contradiction]

Suppose A is optimal, but our algorithm did not find A.

$\Rightarrow$ For some P_i, A matches $P_i \to I_i = [g_i-10, g_i+10]$. but G maps it to $P_i \to I_k = [g_k-10, g_k+10]$.

We have : $g_k-10 \le g_i-10 \le P_i \le g_i+10 \le g_k+10$.

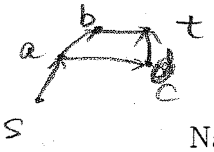
This also means, A maps another $P_k \ne P_i$ to $[g_i-10, g_i+10]$ and $P_k \ge P_i$

$\therefore$ we have $g_k-10 \le g_i-10 \le P_i \le P_k \le g_i+10 \le g_k+10$.

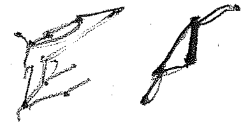
so in A, we can also change the matching to match P_i to I_i and P_k to I_k, since this also satisfy the relation above

and so for every difference matchings, we can argue in the same way.

So G is as good as A after changes [P.S. # of P needed in A won't change after swapping, since no increase or decrease] . $\therefore$ G is also optimal.



sabt
sact



Name: ______________________

Student ID: ______________________

3. **[20 points]** Often, there are multiple shortest paths between two nodes of a graph. These shortest paths may share edges between them. That is s → a → b → t and s → a → d → t are two distinct shortest paths, even though they both use the edge s → a. Give a linear-time algorithm for the following problem:

   Input: An undirected graph G = (V,E) with unit edge length (=1), nodes s and t.

   Output: The number of distinct shortest paths from s to t.

   Give a outline of algorithm, its proof of correctness, and the running time analysis.

A: start from s, check if $(s,t) \in E$

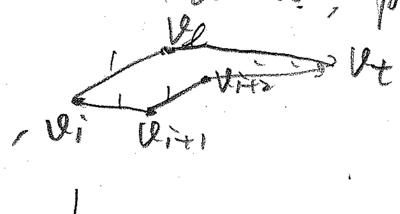
then go to all incident edges in s, check if the ~~other~~ ~~endpoint~~ of incident edges form an edge with v_t. ~~repeat~~

repeat,

If a direct edge is found, we check all possible edges formed and count the number → result

Pf: Suppose $s \to v_1 \to v_2 \to \cdots \to v_k \to t$ found by A is not ~~the~~ shortest, then in some place $\underbrace{v_i \to v_{i+1} \to \cdots \to v_t}_{n}$ can be replaced by $\underbrace{v_i \to \cdots \to t}_{l}$

and $l < n$, for example.



here $v_i \to v_l = 1 < v_i \to v_{i+1} + v_{i+1} \to v_{i+2}$.

but at v_i, we checked all endpoints incident edges, and pick the shortest. thus, this contradicts with A.

$\therefore$ A is optimal.

Find the path $O(|V|) = O(n)$. and all the checking is const time

$\therefore$ total $= O(n)$.

Name: ______________________________

Student ID: ______________________________

4. [**10 points**] Your high-school buddy goes to lunch with you one day, and confidentially tells you that he made an amazing discovery: that he can reduce in polynomial time the Minimum Spanning Tree (MST) problem to Traveling Salesman Problem (TSP). That is, MST $\leq_p$ TSP. He plans to write up his solution carefully and send it to the most prestigious journal in computer science, but he does not want to share with you any details of his intricate solution.

   Assume that he did not make any mistakes, and proved correctly that MST $\leq_p$ TSP. Do you feel that this result is important enough to be published in prestigious journal in computer science? Explain your answer in detail.

No. There are several algorithms for finding a MST in a given graph G (E,V) already. For example Prim's algorithm, if implemented using a priority queue, it can run in $O(m \cdot \log n)$, where $m = |E|$ $n = |V|$. which is of polynomial time.

Whereas TSP is NP-complete. by his reduction, no useful result will be proved any further. So it's not important.

Name: ______________________

Student ID: ______________________

5. [**20 points**] Recall the Traveling Salesman problem, TSP:

   Input: A matrix of distances D (all distances are positive integers), a budget B.
   Output: A tour which passes through all the cities and has length less than or equal to B, if such a tour exists.

   The optimization version of this problem asks directly for the shortest tour TSP-OPT:

   Input: A matrix of distances D(all distances are positive integers).
   Output: The shortest path which passes through all the cities.

   Show that if TSP can be solved in polynomial time, then so can TSP-OPT.

We want to show TSP-OPT $\leq_p$ ~~OP~~ TSP

Pf: We construct an instance of TSP-OPT with the length of the shortest path $= l$.

Then we pass this instance to TSP. with D, the matrix to be the same, and the maximum budget of TSP be l

The way TSP makes the decision is: it finds a path with budget $= l$, then it claim this is the maximum budget (i.e. there might be shorter path, or this is the shortest), otherwise it says no)

So if TSP can solve the passed-in instance, then there will be a path with length $= l$. To mark the path, we just make TSP keep track of the vertices it went through, which is less than polynomial time

So for each l passed in, TSP can find the path in poly-time.

To find the exact shortest path, we iterate l from 2 to $|E|$, which is $O(m)$, so the ~~p~~total will be m-times polynomial time in m,

So the total is still in polynomial time

Name: ______________________

Student ID: ______________________

6. [**20 points**] Show that for any problem Q in NP, there exists an algorithm which solves Q in time $O(2^{p(n)})$, where n is the size of the input and $p(n)$ is a polynomial dependent on input n.

$Q \in NP \Rightarrow$ given a solution. it can be verified in polynomial time. So $P(n)$ here is the possible guesses based on the input size n. For any problem, each guess is either right or wrong. $\therefore$ For each of $P(n)$ possible guesses. there are 2 results.

So in ~~total~~ there will be $2^{P(n)}$, so every $Q \in NP$ can be solved in $O(2^{P(n)})$.