Name:

ID:

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Problem Set 3b (50 points) **Fall 2019, CU-Boulder**

Instructions for submitting your solution:

- The solutions **should be typed** and we cannot accept hand-written solutions. Here's a short intro to Latex.
- You should submit your work through **Gradescope** only.
- If you don't have an account on it, sign up for one using your CU email. You should have gotten an email to sign up. If your name based CU email doesn't work, try the identikey@colorado.edu version.
- Gradescope will only accept **.pdf** files (except for code files that should be submitted separately on Gradescope if a problem set has them) and **try to fit your work in the box provided**.
- You cannot submit a pdf which has less pages than what we provided you as Gradescope won't allow it.
- Verbal reasoning is typically insufficient for full credit. Instead, write a logical argument, in the style of a mathematical proof.
- For every problem in this class, you must justify your answer: show how you arrived at it and why it is correct. If there are assumptions you need to make along the way, state those clearly.
- You may work with other students. However, **all solutions must be written independently and in your own words.** Referencing solutions of any sort is strictly prohibited. You must explicitly cite any sources, as well as any collaborators.

Name:

ID:

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Problem Set 3b (50 points) **Fall 2019, CU-Boulder**

1. (23 pts) Imagine an alternate reality where CU has a small robot that travels around the campus delivering food to hungry students. The robot starts at the C4C and goes to whatever dorm or classroom has placed the order. The fully-charged battery of the robot has enough energy to travel k meters. On campus, there are n wireless charging pods where the robot can stop to charge its battery. Denote by $l_1 < l_2 < \cdots < l_n$ the locations of the charging pods along the route with l_i the distance from the C4C to the ith charging pod. The distance between neighboring charging pods is assumed to be at most k meters. Your objective is to make as few charging stops as possible along the way.

   (a) (10 pts) Write a python program for an optimal greedy algorithm to determine at which charging pods the robot would stop. Your code should take as input k and a *list* of distances of charging pods (first distance in the list is 0 to represent the start point and the last is the destination and not a pod). Print out the charging pods where the robot stops using your greedy strategy.
   Example 1 - If **k = 40** and **Pods = [0, 20, 37, 54, 70, 90]**. Number of stops required is 2 and the output should be **[37, 70]**.
   Example 2 - If **k = 20** and **Pods = [0, 18, 21, 24, 37, 56, 66]**. Number of stops required is 3 and the output should be **[18, 37, 56]**.
   Example 3 - If **k = 20** and **Pods = [0, 10, 15, 18]**. Number of stops required is 0 and the output should be [].

   (b) (3 pts) Provide the time complexity of your python algorithm, including an explanation.
   Solution. The general time mass of this algorithm comes with the single for loop, so general time complexity is O(n). Constant is decided by the cost of the operations inside the loop.

(c) (10 pts) Prove that your algorithm gives an optimal solution.

Solution. Initialization: Before the loop starts, curStop is set as the first value, 0. So when the loop makes its first iteration, it makes the comparison $(a[i+1]-k > curStop)$. In the case of initialization curStop is always 0, so the comparison is $(a[i+1]-k > 0)$ or $(a[i+1] > k)$. So if the comparison registers as true, then that means the distance to the next charging $a[i+1]$ station is greater than the max distance the robot can go, k. So the value of a[i] is the further the robot can go at the start before it runs out of battery.

Maintenance: As the loop continues to iterate, each time a furthest charging spot is found, curStop is set as its distancer. Therefore the comparison $(a[i+1]-k > curStop)$ is checking to see if the difference between the next charging station, one step further than a[i], and curStop is greater than k. If it is, that means the robot cannot make it to a[i+1], so then a[i] is added to path[], and curStop is set as a[i].

Termination: At termination of the loop, is one less than the length of the array of charging ports, because the destination, a[len(a)], is not a charging port, so it isn't included in the list of stops.

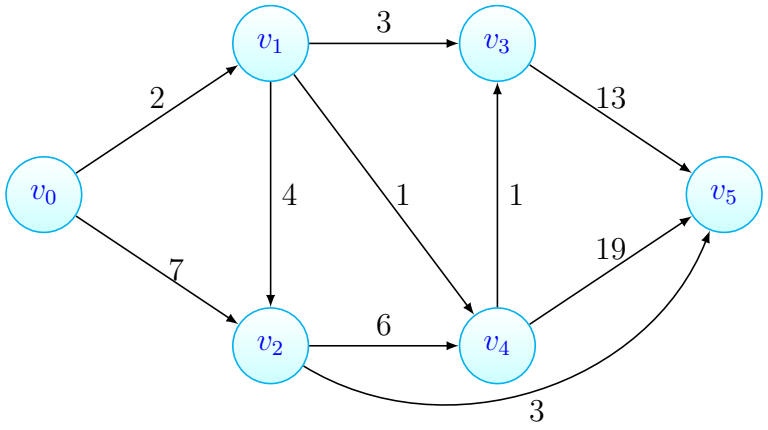
Name: _____

ID: _____

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**

Problem Set 3b (50 points) **Fall 2019, CU-Boulder**

2. (7 pts) Using Dijkstra's algorithm, determine the length of the shortest path from v_0 to each of the other vertices in the graph. Clearly specify the distances from v_0 to each vertex **after each iteration** of the algorithm.



iteration	visited	v_0	v_1	v_2	v_3	v_4	v_5
1	v_0	0	2	7	inf	inf	inf
2	v_1	0	2	6	5	3	inf
3	v_4	0	2	6	4	3	22
4	v_3	0	2	6	4	3	17
5	v_2	0	2	6	4	3	10
6	v_5	0	2	6	4	3	10

Solution.

Name:

ID:

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Problem Set 3b (50 points) **Fall 2019, CU-Boulder**

3. (20 pts) After years of futility, the Colorado Rockies have decided to try a new approach to signing players. Next year, they have a target number of wins, n, and they want to sign the fewest number of players who can produce exactly those n wins. In this model, each player has a win value of $v_1 < v_2 < \cdots < v_r$ for r player types, where each player's value v_i is a positive integer representing the number of wins he brings to the team. (Note: In a real-world example, All-Star third baseman, Nolan Arenado, contributed 4.5 wins this year beyond what a league-minimum player would have contributed to the team.) The team's goal is to obtain a set of counts $\{d_i\}$, one for each player type (so d_i represents the quantity of players with valuation v_i that are recruited), such that $\sum_{i=1}^{r} d_i = k$ and where k is the number of players signed, and k is minimized.

   (a) (10 pts) Write a greedy algorithm that will produce an optimal solution for a set of player win values of $[1, 2, 4, 8, 16]$ and prove that your algorithm is optimal for those values. Your algorithm need only be optimal for the fixed win values $[1, 2, 4, 8, 16]$. You do **not** need to consider other configuration of win values.

   Solution.

```
def signPlayers(n):
a = [1, 2, 4, 8, 16]
players = []*5
i = len(a)-1
while (i>-1):
    if(n-a[i]>=0):
        players[i] += 1
        n-=a[i]
    else:
        i -= 1
return players
```

   When input n is 30, this algorithm returns [0,1,1,1,1]. This proof is optimal for the set [1,2,4,8,16] because this set is a geometric sequence (2^n). So a[i] = 2*a[i-1]. When the loop starts, the index is the last in the set of values, and it only decrements when no more a[i]'s fit in n. If it were to do it any other way, for example start with the element at the index before a[i], it would just be counting twice as many players as it needs to (starting at 8 would cause it to count 2 players of v = 8 as opposed to 1 player of v = 16).

[Additional space for solving Q3a]

(b) (10 pts) Find a set of win values where your algorithm does not produce the optimal solution and show where your algorithm fails for those values.

Solution. The set [1, 5, 8] does not work when the input value n is 20. it would output [4, 0, 2] (That's 4 player values of 1, 0 player values of 5, and 2 player values of 8).

This is because the algorithm starts by adding two players of value 8, because 8*2 is less than 20. Then it moves to 5, but adding 5 would make it go over 20. Then it moves to 1, and adds 4 players of value 1. That's 6 total players, but it could be done with 4 players of value 5. This is because the set is not geometric, so starting by checking the greatest value player does not give the optimal number of players.

Name:

ID:

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Problem Set 3b (50 points) **Fall 2019, CU-Boulder**

Ungraded questions - These questions are for your practice. We won't grade them or provide a typed solution but we are open to discuss these in our OHs and you should take feed backs on your approach during the OHs. These questions are part of the syllabus.

1. Suppose we have a directed graph G, where each edge e_i has a weight $w_i \in (0, 1)$. The weight of a path is the product of the weights of each edge.

   (a) Explain why a version of Dijkstra's algorithm cannot be used here. [**Hint:** We may think about transforming G into a graph H, where the weight of edge i in H is $\ln(w_i)$. It is equivalent to apply Dijkstra's algorithm to H.]
   Solution.

   (b) What conditions does each edge weight w_i need to satisfy, in order to use Dijkstra's algorithm?
   Solution.