

Answer the questions in the boxes provided on the question sheets. If you run out of room for an answer, add a page to the end of the document.

Name: Kayley Seon

Wisc id: kseon

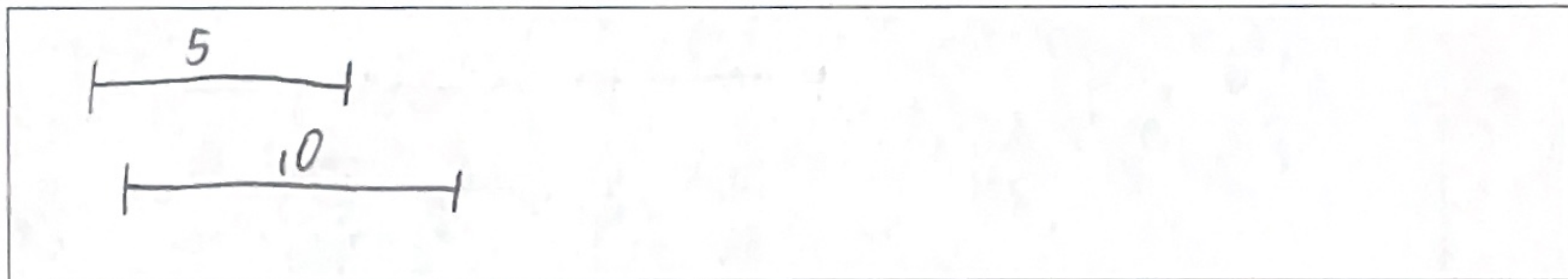
Greedy Algorithms

1. In one or two sentences, describe what a greedy algorithm is. Your definition should be informal, something you could share with a non computer scientist.

Greedy algorithm takes the biggest value/weight, prioritizes and then you will be able to maximize the solution.

2. There are many different problems all described as "scheduling" problems. In the following questions, pay attention to the details of the problem setup, as they will change each time!

- (a) Let each job have a start time, an end time, and a value. We want to schedule as much value of non-conflicting jobs as possible. Use a counterexample to show that Earliest Finish First (the greedy algorithm we used for jobs with all equal value) does NOT work in this case.



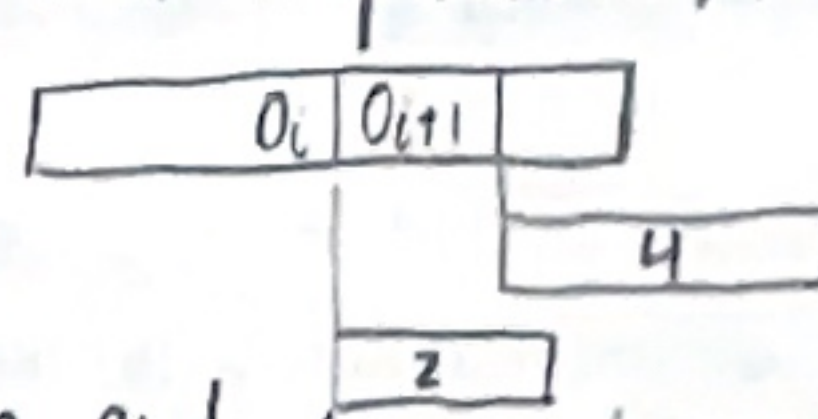
- (b) *Kleinberg, Jon. Algorithm Design (p. 191, q. 7)* Now let each job consist of two durations. A job i must be preprocessed for p_i time on a supercomputer, and then finished for f_i time on a standard PC. There are enough PCs available to run all jobs at the same time, but there is only one supercomputer (which can only run a single job at a time). The completion time of a schedule is defined as the earliest time when all jobs are done running on both the supercomputer and the PCs. Give a polynomial time algorithm that finds a schedule with the earliest completion time possible.

Kind of like
a rest
problem

Sort the finishing times by longest first, and then add this order into the supercomputer. This is because supercomputer is a constant time, so the timing ends up depending on the finishing time at the PC and then you would want the longest finishing time run first while the others complete in parallel.

(c) Prove the correctness and efficiency of your algorithm from part (c).

$G = \{g_1, g_2, g_3, \dots, g_n\}$ g sorted descending finishing times. ^{greedy solution}
 $O = \{o_1, o_2, o_3, \dots, o_n\}$ O is any other feasible algorithm/solution

Assume there is an inversion O_i and O_{i+1} . O_{i+1} has a greater finishing time than O_i . Fixing this inversion would make the finishing time no worse. Because the computers run in parallel, this would mean that O_{i+1} is finishing after O_i .
 As such, we know that it will be the slowest time, and the maximum time possible

$$P_1 + P_2 + \dots + P_i + f_i$$

$$P_1 + P_2 + P_i + P_{i+1} + f_i + 1$$

$$P_1 + P_2 + \dots + P_{i-1} + P_{i+1} + P_i + f_i$$

$$P_1 + P_2 + \dots + P_{i-1} + P_{i+1} + f_{i+1}$$

$$f_i \leq f_{i+1}$$

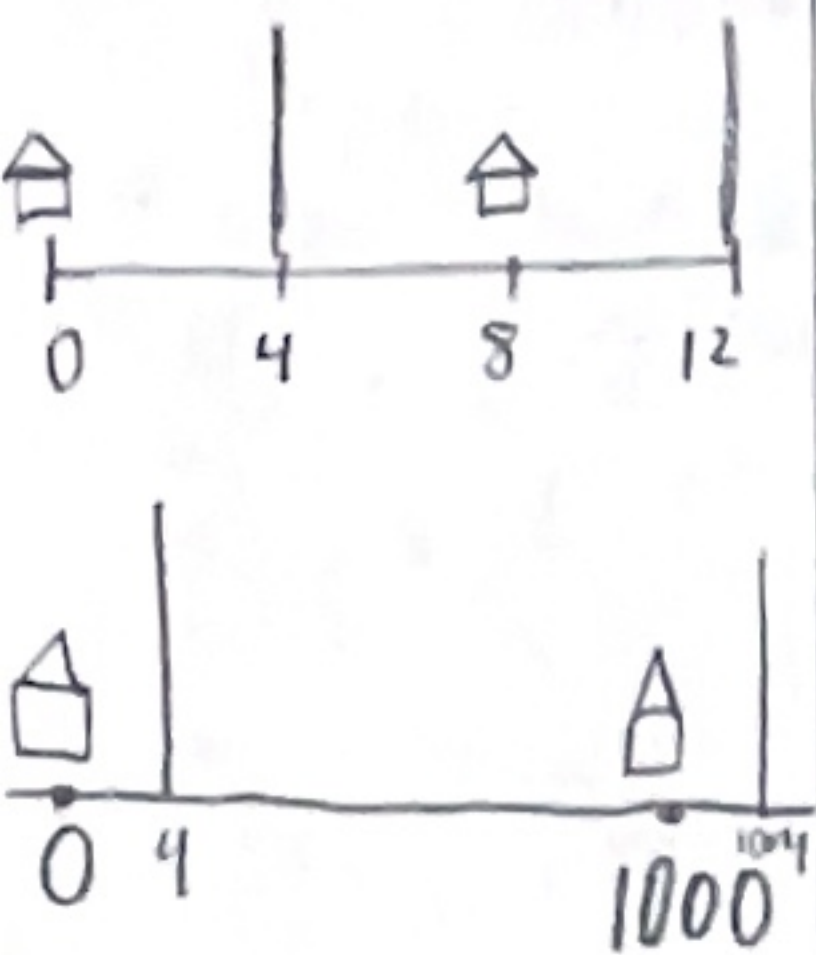
makes it no worse than before.

With O , we would be able to, by induction, prove that by this inversion, we can continue to invert O until it becomes the most optimal, which would give us G , proving that it is our optimal solution.

Runtime is $O(n \log n)$ due to sorting

3. Kleinberg, Jon. *Algorithm Design* (p. 190, q. 5)

- (a) Consider a long straight road with houses scattered along it. We want to place cell phone towers along the road so that every house is within four miles of at least one tower. Give an efficient algorithm that achieves this goal using the minimum possible number of towers.



Tower must be placed within 4 miles, no more than 4 miles away from the house. Going from left to right, we would put the tower 4 miles right of the first house, and place another tower if and only if there is a house without coverage.

- (b) Prove the correctness of your algorithm. *Stay ahead*

only way to fail is if there is redundancy

Greedy solution $S: (f_1, f_2, f_3, \dots, f_k)$ - best
 optimal solution $S': (g_1, g_2, g_3, \dots, g_m)$ let $k \geq m$

Claim $k_i = 1 \dots n, f_i \geq g_i$

Base Case: $i=1$, by nature of greedy algorithms, $f_1 \geq g_1$

Induction Hypothesis: holds for $i=k$

Greedy algorithm selects closest house that is outside range of previous tower $[f_k + 1, k + n]$ while optimal selects break point $[g_{k+1}, f_{k+n}]$.

By induction hypothesis, $f_{k+n} \geq g_{k+n} \rightarrow \text{so } f_{k+1} \geq g_{k+1}$

\therefore greedy is optimal

\nexists a tower greedy uses \geq optimal, whenever greedy places tower, a dead end will have greedy tower boundary, greedy most definitely done

4. Kleinberg, Jon. *Algorithm Design* (p. 197, q. 18) Your friends are planning to drive north from Madison to the town of Superior, Wisconsin over winter break. They have drawn a directed graph with nodes representing potential stops and edges representing the roads between them.

They have also found a weather forecasting site that can accurately predict how long it will take to traverse one of the edges on their graph, given the starting time t . This is important because some of the roads on their graph are affected strongly by the seasons and by extreme weather. It's guaranteed that it never takes negative time to traverse an edge, and that you can never arrive earlier by starting later.

- (a) Design an algorithm your friends can use to plot the quickest route. You may assume that they start at time $t = 0$, and that the predictions made by the weather forecasting site are accurate.

algorithm outline $t=0$

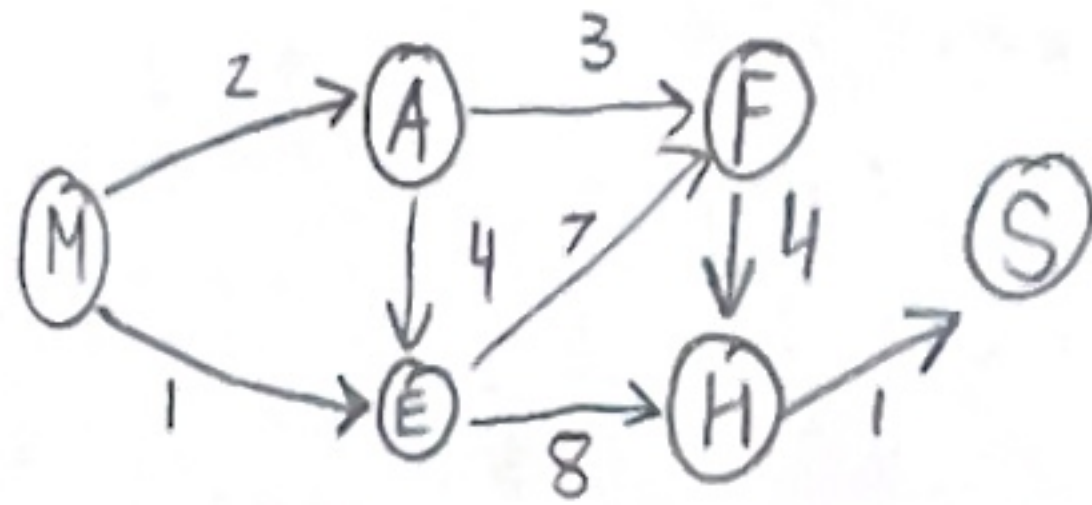
$Q = [] \rightarrow$ integer max
 $dis(v) = \infty$ // source \rightarrow vertex
 $predecessor(v) = null$
 $dis(u) = 0$

while ($Q \neq \emptyset$) {
 $u = \text{Extract min}()$ // take vertex out
 all neighbors v of u :
 $dis(v) > dis(u) + w(u, v)$
 $dis(v) = dis(u)$
 $prev(v) = u$
}

- $m \log n$ is priority que, $m \times n$ edges-vertices
- priority queue
- calculate all the shortest paths, add all of the vertices

- (b) Demonstrate how your algorithm works using a small example with 6 nodes. Your demonstration should include any data structures you maintain during the execution of your algorithm and any queries you make to the weather forecasting site. For example, if your algorithm maintains a "current path" that grows from (M)adison to (S)uperior, you might show something like the following table:

Path	Total time
M	0
M,A	2
M,A,E	5
M,A,E,F	6
M,A,E	5
M,A,E,H	10
M,A,E,H,S	13



nodes	0	1	2	3	4	5
M	<u>0</u>	0	0	0	0	0
A	∞	2	2	2	2	2
E	∞	<u>1</u>	1	1	1	1
F	∞	∞	8	<u>5</u>	5	5
H	∞	∞	9	9	<u>9</u>	9
S	∞	∞	∞	∞	10	<u>10</u>

Coding Question

5. Implement the optimal algorithm for interval scheduling (for a definition of the problem, see the Greedy slides on Canvas) in either C, C++, C#, Java, or Python. Be efficient and implement it in $O(n \log n)$ time, where n is the number of jobs.

The input will start with an positive integer, giving the number of instances that follow. For each instance, there will be a positive integer, giving the number of jobs. For each job, there will be a pair of positive integers i and j , where $i < j$, and i is the start time, and j is the end time.

A sample input is the following:

```
2
1
1 4
3
1 2
3 4
2 6
```

The sample input has two instances. The first instance has one job to schedule with a start time of 1 and an end time of 4. The second instance has 3 jobs.

For each instance, your program should output the number of intervals scheduled on a separate line. Each output line should be terminated by a newline. The correct output to the sample input would be:

```
1
2
```