CSE 101
Practice Quiz 2 solutions, Fall, 2019

1. Answer True or False or short answer with a brief explanation, counterexample or proof.

   (a) Let M be an MST of G. If M contains an edge of weight w, than all MSTs of G must contain an edge of weight w.

   Solution: True: Let T_1 and T_2 be two MSTs. Suppose that an edge of weight w_1 is in T_1 and not in T_2. Add the edge of weight w_1 to T_2. Now you have created a cycle.
   Case 1: w_1 is the heaviest edge in the cycle. Add one of the lighter edges in the cycle to T_1 that will create a cycle and remove w_1 from T_1. Now you have decreased the total weight of T_1 which was assumed to be minimal.
   Case 2: There exists an edge e' in the cycle that has weight greater than w_1. Then we can remove e' from T_2 and now we have decreased the total weight of T_2 which was assumed to be minimal.

   (b) Let G be an undirected connected graph with positive edge weights. If G has two edges with the same weight then it must have more than one MST.

   Solution: False, consider the counterexample:
   A:(B,1)
   B:(A,1),(C,1)
   C:(B,1)

   (c) When using the union-find data structure, any rank k vertex has fewer than 2^k vertices in its subtree.

   Solution: False, from the book and the slides, we proved that each rank k vertex must have equal to or more than 2^k vertices.

   (d) $O(n^{1.585})$ is the best runtime possible for multiplication of two n-bit integers.

   Solution: False, we saw that using Cook-Toom and splitting the numbers into 3 parts, that we could get a runtime of $O(n^{1.465})$.

   (e) Suppose $T(n) = 3T(n/2) + cn$ and $S(n) = 4S(n/6) + cn$ with $T(1) = S(1) = c$. What can you say about $T(n)$ and $S(n)$? Is $T(n) = O(S(n))$? Or is $S(n) = O(T(n))$?

   Solution: By the master theorem:
   $a_T = 3, b_T = 2, d_T = 1$, so $a_T > b_T^{d_T}$, so $T(n) = \Theta\left(n^{\log_{b_T}(a_T)}\right) = \Theta\left(n^{1.585}\right)$.
   $a_S = 4, b_S = 6, d_S = 1$, so $a_S < b_S^{d_S}$, so $T(n) = \Theta\left(n^{d_S}\right) = \Theta\left(n\right)$.
   We can conclude that $S(n) = O(T(n))$.

   (f) Suppose $T(n) = T(n/3) + O(\sqrt{n})$. Use Master Theorem to solve this recurrence.

   Solution: By the master theorem:
   $a = 1, b = 3, d = 1/2$, so $a < b^d$, so $T(n) = O\left(n^d\right) = O\left(\sqrt{n}\right)$.

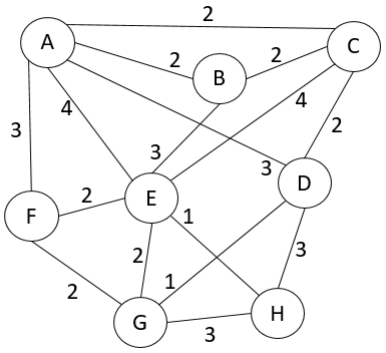
   (g) The expected runtime of QuickSelect (the randomized selection algorithm from class) is $O(n)$

   Solution: True, see slides.

   (h) Suppose $T(n) = 9T(n/3) + O(n^2)$. Use Master Theorem to solve this recurrence.

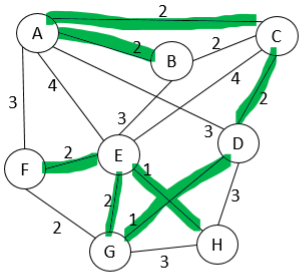
   Solution: By the master theorem:
   $a = 9, b = 3, d = 2$, so $a = b^d$, so $T(n) = O\left(n^d \log(n)\right) = O\left(n^2 \log(n)\right)$.

2. Consider the following graph:



(a) Perform Kruskal on the above graph. When ordering the edges by weight, break ties by alphabetical order of the endpoints.

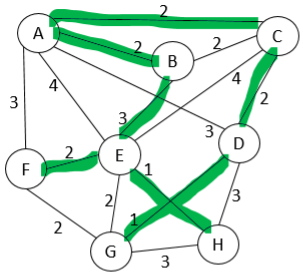
Solution:



(D,G): 1
(E,H): 1
(A,B): 2
(A,C): 2
(B,C): 2
(C,D): 2
(E,F): 2
(E,G): 2
(F,G): 2
(A,D): 3
(A,E): 3
(B,E): 3
(D,H): 3
(G,H): 3
(A,F): 4
(C,E): 4

(b) Perform Prim's on the above graph starting at A. Keep a table of cost and prev for each iteration. Break ties by alphabetical order.

Solution:



A	B	C	D	E	F	G	H
0	∞	∞	∞	∞	∞	∞	∞
	2,A	2,A	3,A	4,A	3,A	∞	∞
		2,A	3,A	3,B	3,A	∞	∞
			2,C	3,B	3,A	∞	∞
				3,B	3,A	1,D	3,D
					2,E	1,D	1,E
					2,E		1,E
					2,E		

3. Suppose that you and your friends are going to hike the John Muir trail this summer. You want to hike as much as possible per day but, you do not want to hike after dark. On a map there are a large set of good stopping points for camping and you design your trip based on the following system: Every

time you come to a potential stopping point, determine whether you can make it to the next one before nightfall. If you can make it then keep hiking, otherwise stop and camp. You claim that with this strategy you will minimize the number of camping stops you must make.

(a) Suppose the stopping points are located at distances $x_1, \ldots, x_n$ from the trailhead. Also assume that your group can hike d distance per day (independent of terrain, weather conditions and so forth.)

Determine the

- Instance,
- Solution format,
- constraints and
- objective function

for this problem.

Solution:
Instance: $(x_1, \ldots, x_n)$ the locations of the stopping points starting from the trail head and d, the maximum distance you can travel in a day.

Solution Format: $[p_1, \ldots, p_k]$ the list of locations you will camp at.

Constraints: $p_{j+1} - p_j \leq d$ for each j.

Objective: Minimize k, the number of stops.

(b) Prove this strategy is optimal.

Proof:(exchange argument:)
Exchange Argument: For some input $I = (x_1, \ldots, x_n; d)$, let g be the first greedy location. Let $OS = [c_1, \ldots, c_k$ be some solution to I that does not include stopping at location g. Then the first stop of OS stops at point c_1. Then by the nature of the greedy choice, you cannot stop any farther than g on your first day, so $c_1 < g$. Create OS' by stopping at g on your first day instead of c_1. And then continuing along the same sequence of stopping points after g in OS. (you might pass by other stopping points of OS on your way to g.)

OS' is valid: We must show that we can always travel between two consecutive stopping points within one day. Suppose that c_j is the stopping point in OS that occurs right before g, then we must show that we can get from g to c_{j+1}. By the validity of OS, we know that $c_{j+1} - c_j \leq d$ and since $c_j < g$, we know that $c_{j+1} - g \leq d$. Therefore OS' is a valid solution.
OS' is just as good or better than OS: At the very least, there is one stop before g in OS and so it follows from the construction of OS' that

$$OS' = (OS - \{\texttt{all stops before } g \texttt{ in } OS\}) \cup \{g\}.$$

Therefore $|OS| \geq |OS'|$.
induction part:
Claim: *For any input of any size $n \geq 1$, the greedy solution is optimal.*
Base Case: For $n = 1$, stop at the only stopping point which will be the end of the trail. This is optimal
Inductive Hypothesis: Suppose that for some $n > 1$, the greedy strategy is optimal for all inputs of size k such that $1 \leq k < n$.
Inductive Step: Suppose $I = (x_1, \ldots, x_n; d)$ is an input of size n. Then let OS be any solution of I. Then by the exchange argument, there exists a solution OS' that stops at g and uses at most as many stops as OS. Therefore we have that:

$$|OS| \geq |OS'| = |\{g\} \cup S(I')| \geq |\{g\} \cup GS(I')| = |GS(I)|$$

Where I' is the set of stops after g.

(c) Implement and determine runtime.

Scan through the locations until one of them exceeds d. Camp at the first stopping point just before the distance d. Then continue from the stopping point in the same manner. (Note that this implementation assumes that there is in fact a solution.)

```
procedure Camps(x_1, ..., x_n; d)
  output = [ ]
  i = 1
  start = 0
  while i ≤ n:
    while x_i − start < d:
      i = i + 1.
    output = output + [x_{i−1}]
    start = x_i
  return output
```

This will run in $O(n)$ time because there is a constant time operation every time i is incremented and i is incremented n times.

4. Suppose you have n friends, $F_1, \ldots, F_n$ visiting during the quarter and you want to have a series of parties with each friend invited to at least one of the parties. For each friend F_i you are given f_i and ℓ_i which are the first and last days F_i will be visiting. (The friend will be available for all dates between f_i and l_i, inclusive.) You want to throw as few parties as possible; any number of friends can be invited to a party. For example, if Abe is visiting from day 1 to 3, Beth from day 2 to 7, Cora from day 1 to 5, and David from day 4 to 6, you can have one party on day 3 with Abe, Beth and Cora, and one on day 4 with just David.

Candidate Greedy Strategy I: Plan a party for the earliest last day of a friends visit. Invite all friends available that day to that party. Then recursively schedule parties for the set of friends who cannot attend that one.

Candidate Greedy Strategy II: Plan a party on a day that has the maximum number of people visiting. Then invite all of those people available on that day. Recursively schedule parties for the set of friends who cannot attend that one.

(a) Assume that you are given the first and last day of each of your n friends. Determine the

- Instance,
- Solution format,
- constraints and
- objective function

for this problem.

Solution

Instance: $((f_1, \ell_1), \ldots, (f_n, \ell_n))$ the first and last days that your n friends are visiting.

Solution Format: $S = (p_1, \ldots, p_k)$ a schedule of days.

Constraints: For all $i = 1 \ldots n$, there exists at least one $p \in (p_1, \ldots, p_k)$ such that $f_i \leq p \leq \ell_i$.

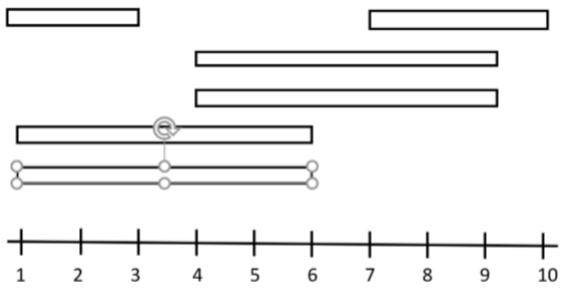
Objective: Minimize k (the number of parties.)

(b) Decide which greedy strategy is optimal

Solution:
Candidate Greedy Strategy I is always optimal and Candidate Greedy Strategy II is not always optimal.

(c) Give a counterexample for the suboptimal strategy.

Solution:
Candidate Greedy strategy II is not always optimal. Say that your friends come for the intervals shown in the diagram:



Then this strategy will put a party on day 5 and then put a party on day 2 and day 8 for a total of 3 parties. But you could have just had parties on day 2 and day 8 to meet the constraints.

(d) Prove the optimal strategy is optimal.

Solution:
Proof:(exchange argument)
ExArg Claim: *For some input $I = (f_1, \ell_1), \ldots, (f_n, \ell_n))$ ordered by ascending last days, the greedy choice g is the day ℓ_1. Suppose that OS is a valid set of days that do not include the day ℓ_1. Then there is a solution OS' that includes the day ℓ_1 and has at most as many party days as OS.*
Proof of ExArg Claim: Suppose that $OS = (p_1, \ldots, p_k)$ be a valid sequence of days that meets the constraints of the problem ordered from earliest to latest that excludes ℓ_1, i.e., $p_1 \neq \ell_1$. Then create OS' by taking all the days of OS and exchanging p_1 for ℓ_1, i.e., $OS' = (\ell_1, p_2, \ldots, p_k)$.

We now must show

- OS' is a valid solution.
- $|OS'| \leq |OS|$.
  - OS' **is a valid solution.:** Since OS is already a valid solution, it suffices to show that everyone who came to party p_1 could also come to the party on ℓ_1. Let's assume by contradiction that there is a person P that can come on p_1 but cannot come on ℓ_1. Then ℓ_1 is not in the interval $[f, \ell]$ and by the greedy choice, $\ell_1 \leq \ell$. Therefore $\ell_1 < f$. Since P can come to the party on day p_1, that means that $f \leq p_1 \leq \ell$. This implies that $\ell_1 < p_1$ which means that the first person to leave will not be able to make it to the party p_1. This contradicts the fact that OS is a valid solution.
  - $|OS'| \leq |OS|$: Since we are only exchanging one party for another, we have that $|OS'| = |OS|$.

Induction Part:
Claim: *For any input of any size $n \geq 1$, the greedy solution is optimal.*
Base Case: For $n = 1$, you can just have a party on the last day of your only guest.
Induction Hypothesis: Suppose that for some $n > 1$, the greedy strategy is optimal for all inputs of size k such that $1 \leq k \leq n - 1$.
Induction Step: Suppose $I = (f_1, \ell_1), \ldots, (f_n, \ell_n))$ is an input of size n. Then let OS be any solution. Then by the Exchange Argument, there exists a solution OS' that includes ℓ_1 and has at most as many parties as OS. Therefore, we have that:

$$|OS(I)| \geq |OS'(I)| = |\{\ell_1\} \cup S(I')| \geq |\{\ell_1\} \cup GS(I')| = |GS(I)|$$

(e) Implement and determine runtime.

Solution:
Implementation: First sort the visitors by their last days. Then set the first party day to be the last day of the first person on the list. Loop through the rest of the visitors in the order of their last day until you find somebody who cannot make it to the most recently chosen party day and set their last day as the next party day.

Parties $(f_1, \ell_1), \ldots, (f_n, \ell_n))$
  output $= \ell$
  $D = \ell_1$
  for $i = 2, \ldots, n$:
    if $f_i \leq D$:
      continue
    else:
      put ℓ_i into output
      $D = \ell_i$
  return output

Runtime: Sorting takes $O(n \log n)$ time. Then the loop iterates $n-1$ times with a constant time operation for each iteration so the total runtime of the loop is $O(n)$. The total time of the algorithm is $O(n \log(n))$.

5. Consider the following problem: We have n oxen, $Ox_1, ..Ox_n$, each with a strength rating S_i. We need to pair the oxen up into teams to pull a plow; if Ox_i and Ox_j are in a team, we must have $S_i + S_j \geq P$, where P is the weight of a plow. Each ox can only be in at most one team. Each team has exactly two oxen. We want to maximize the number of teams.

Candidate Greedy Strategy I: Take the strongest and weakest oxen. If together they meet the strength requirement, make them a team. Recursively find the most teams among the remaining oxen.

Otherwise, delete the weakest ox. Recursively find the most teams among the remaining oxen.

Candidate Greedy Strategy II: Take the weakest two oxen, If together they meet the strength requirement, make them a team. Recursively find the most teams among the remaining oxen.

Otherwise, delete the weakest ox. Recursively find the most teams among the remaining oxen.

(a) Assume that you are given the strengths of all oxen in increasing order of strength. Determine the

- Instance,
- Solution format,
- constraints and
- objective function

for this problem.

Solution

Instance: $(S_1, \ldots, S_n)$ the strengths of n oxen and P, the minimum combined strength to tow a plow.

Solution Format: $\{(i_1, j_1), \ldots, (i_k, j_k)$ a set of pairs of indices of oxen such that each oxen appears at most once.

Constraints: For all $m = 1 \ldots k$, $S_{i_m} + S_{j_m} \geq P$.

Objective: Maximize k (the number of pairs.)

(b) Decide which greedy strategy is optimal

Solution:
Candidate Greedy Strategy I is always optimal and Candidate Greedy Strategy II is not always optimal.

(c) Give a counterexample for the suboptimal strategy.

Solution:
Candidate Greedy Strategy II is not always optimal. Suppose that $P = 10$ and there are 4 oxen with strengths: $(1, 2, 8, 9)$. If we try to pair the first two, they cannot pull the plow, so we would delete 1. But then of the three remaining, we'll get at most 1 pair. Alternatively, we can pair $(1, 9)$ and $(2, 8)$ to get two viable pairs.

(d) Prove the optimal strategy is optimal.

Solution:
Proof:(exchange argument)
ExArg Claim: *For some input $I = (S_1, \ldots, S_n)$ ordered by ascending strength and some minimum plow strength P, the greedy choice g is the pair (S_n, S_g) such that S_g is the weakest strength ox with the property: $S_n + S_g \geq P$. Suppose that OS is a valid set of pairs of oxen that does not pair (S_n, S_g). Then there is a solution OS' that includes the pair (S_n, S_g) and has at least as many pairs as OS.*
Proof of ExArg Claim: Suppose that OS be a valid set of pairs that does not pair (S_n, S_g). Then in OS there are 4 possibilities:

- Neither S_n nor S_g are paired with another ox.

  In this case, define $OS' = OS \cup \{S_n, S_g\}$. Since OS is valid and $S_n + S_g \geq P$, OS' must be valid. And $|OS'| = |OS| + 1$.
- S_n is not paired and S_g is paired with S_j for some $j \neq n$.

  In this case, swap out the pair (S_g, S_j) with (S_n, S_g). Since OS is valid and $S_n + S_g \geq P$, OS' must be valid. And $|OS'| = |OS|$.
- S_g is not paired and S_n is paired with S_j for some $j \neq g$.

  In this case, swap out the pair (S_n, S_j) with (S_n, S_g). Since OS is valid and $S_n + S_g \geq P$, OS' must be valid. And $|OS'| = |OS|$.
- S_n is paired with S_j and S_g is paired with S_i.

  In this case, swap out the two pairs (S_n, S_j) and (S_g, S_i) with (S_n, S_g) and (S_i, S_j). Since OS is valid and $S_n + S_g \geq P$, it remains to be shown that $S_i + S_j \geq P$ in order to show that OS' is valid. We know that $S_i + S_g \geq P$ and that $S_j \geq S_g$. Therefore $S_i + S_j \geq P$. And $|OS'| = |OS|$.

Induction Part:
Claim: *For any input of any size $n \geq 2$, the greedy solution is optimal.*
Base Case: For $n = 2$, The greedy choice will choose the two oxen if they make a valid pair and that will be optimal.
Induction Hypothesis: Suppose that for some $n > 2$, the greedy strategy is optimal for all inputs of size k such that $2 \leq k \leq n - 1$.
Induction Step: Suppose $I = (S_1, \ldots, S_n)$ is an input of size n in increasing order.
Case 1: $S_1 + S_n < P$: then there is no way for S_1 to pair with any other ox and so by the inductive hypothesis, the greedy strategy will find the optimal solution of $(S_2, \ldots, S_n)$.
Case 2: $S_1 + S_n \geq P$. Then let OS be any solution. Then by the Exchange Argument, there exists a solution OS' that includes (S_1, S_n) and has at least as many pairs as OS. Therefore, we have that:

$$|OS(I)| \leq |OS'(I)| = |(S_1, S_n) \cup S(I')| \leq |(S_1, S_n) \cup GS(I')| = |GS(I)|$$

where $I' = I - \{(S_1, S_n)\}$.

(e) Implement and determine runtime.

Solution:
Implementation: If there are less than two oxen, then return the empty set. Otherwise sort the oxen by their strength. If (S_1, S_n) is not a valid pair then recurse on $(S_2, \ldots, S_n)$. Otherwise, recurse on $(S_3, \ldots, S_{n-1})$ and add in the pair (S_1, S_n).

Teams $((S_1, \ldots, S_n), P)$ sorted in increasing order
  if $n < 2$:
    return $\emptyset$
  if $S_1 + S_n < P$:
    return **Teams**$((S_2, \ldots, S_n), P)$
  if $S_1 + S_n \geq P$:
    return **Teams**$((S_2, \ldots, S_{n-1}), P) \cup \{(S_1, S_n)\}$

Runtime: Sorting takes $O(n \log n)$ time. Then in the worst case, the recursive call is of size $n - 1$ and the non-recursive part takes $O(1)$ time. So $T(n) \leq T(n-1) + O(1)$ which simplifies to $T(n) = O(n)$. All in all with the sorting, the algorithm will take $O(n \log n + n) = O(n \log n)$.