CSCI 570 Homework 2

Due Date: Sept. 21, 2023 at 11:59 P.M.

1. In the SGM building at USC Viterbi, there is a need to schedule a series of n classes on a day with varying start and end times. Each class is represented by an interval $[start\_time, end\_time]$, where $start\_time$ is the time when the class begins and end_time is when it concludes. Each class requires the exclusive use of a lecture hall.

   (a) To optimize resource allocation, devise an algorithm using binary heap(s) to determine the minimum number of lecture halls needed to accommodate all the classes without any class overlapping in scheduling. (7 points)

   (b) Analyze and state its worst-case time complexity in terms of n. (3 points)

Input: List of intervals
Output: Minimum number of lecture halls needed
if intervals is empty **then**
    return 0
end if
Initialize an empty min-heap `free_rooms`
Sort `intervals` by start times
Push the end time of the first interval onto `free_rooms`
for each remaining interval in `intervals` **do**
    if the earliest end time in `free_rooms` $\leq$ start time of current interval **then**
        Pop the earliest end time from `free_rooms`
    end if
    Push the end time of the current interval onto `free_rooms`
end for
return the size of `free_rooms` as the minimum number of lecture halls needed

The worst-case time complexity of the algorithm is $O(n \log n)$, where n represents the number of intervals (classes). This complexity arises

primarily from two operations: sorting the intervals based on their start times, which takes $O(n \log n)$ time, and performing heap operations for each interval, also taking $O(n \log n)$ time in total.

2. The Thomas Lord Department of Computer Science at USC Viterbi is working on a project to compile research papers from various departments and institutes across USC. Each department maintains a sorted list of its own research papers by publication date, and the USC researchers need to combine all these lists to create a comprehensive catalog sorted by publication date. With limited computing resources on hand, they are facing a tight deadline. To address this challenge, they are seeking the fastest algorithm to merge these sorted lists efficiently, taking into account the total number of research papers (m) and the number of departments (n).

   (a) Devise an algorithm using concepts of binary heap(s). (7 points)

   (b) Analyze and state its worst-case time complexity in terms of m and n. (3 points)

Use a min heap H of size n.
Use an Array named $CombinedSort[1, \ldots, m]$ to store the combined result (our final answer!).
Use another array named $CP[1, \ldots, n]$ to store the current pointer locations of n different arrays.
Give each department IDs ranging from 1 to n.
Insert the first elements of each sorted array into the heap.

The objects entered into the heap will consist of
$(PublicationDate, ResearchPaperID, DepartmentID)$
with $PublicationDate$ as the key.

for $j = 1$ to n **do**
    Set pointer $CP(j) = 2$
end for
for $i = 1$ to m **do**

$S = \text{ExtractMin}(H)$
$CombinedSort(i) = S.ResearchPaperID$
$j = S.DepartmentID$
Insert element at $CP(j)$ from Department j into the heap
Increment $CP(j)$
end for

The runtime complexity of the given algorithm is $O(m \cdot \log(n))$, where m is the number of research papers and n is the number of departments. This is because the dominant factor in the time complexity is the loop that iterates over m research papers, each of which involves $O(\log(n))$ heap operations.

3. In an interstellar odyssey, a spaceship embarks on a journey from a celestial origin to a distant target star, equipped with an initial fuel capacity of `'currentFuel'` units. Along the cosmic highway, there are space refueling stations represented as an array of `'spaceStations'`, each defined as `[distanceToStationFromOrigin, fuelCapacity]`. There are `'n'` space stations between the celestial origin and the target star. The objective is to determine the minimum number of refueling stops required for the spaceship to reach the target star, which is located `'targetDistance'` light-years away. The spaceship consumes one unit of fuel per light-year traveled. Upon encountering a space station, it can refuel completely by transferring all available `'fuelCapacity'` units from the station. The challenge is to calculate the `'refuelStops'` needed for a successful voyage to the target star or return -1 if reaching the destination remains unattainable with the available fuel resources.

   (a) Devise an algorithm using concepts of binary heap(s). (7 points)

   (b) Analyze and state its worst-case time complexity in terms of n. (3 points)

Initialize an empty max-heap named $maxHeap$
Initialize a variable $refuelStops$ to store the result, set it to 0
Initialize an index $stationIndex$ and set it to 0

while $currentFuel < targetDistance$ **do**
    while $stationIndex <$ length of $spaceStations$
and $spaceStations[stationIndex][0] \leq currentFuel$ **do**
        Push $spaceStations[stationIndex][1]$ onto the $maxHeap$
        Increment $stationIndex$ by 1
    end while
    if $maxHeap$ is empty **then**
        return -1
    end if
    Increment $currentFuel$ by the maximum value popped from $maxHeap$
    Increment $refuelStops$ by 1
end while
return $refuelStops$

The worst-case time complexity of this algorithm is $O(n \log(n))$, where n is the number of space stations. The nested while loop that iterates through the space stations can run at most n times, and each insertion and deletion operation on the max-heap (`maxHeap`) takes $O(\log(n))$ time. Therefore, the overall time complexity is $O(n \log(n))$.

4. You are tasked with performing operations on binomial min-heaps using a sequence of numbers. Follow the steps below:

   (a) Create a binomial min-heap $H1$ by inserting the following numbers from left to right: 3, 1, 13, 9, 11, 5, 7, 15. (2 points)

   (b) Perform one `deleteMin()` operation on $H1$ to obtain $H2$. (2 points)

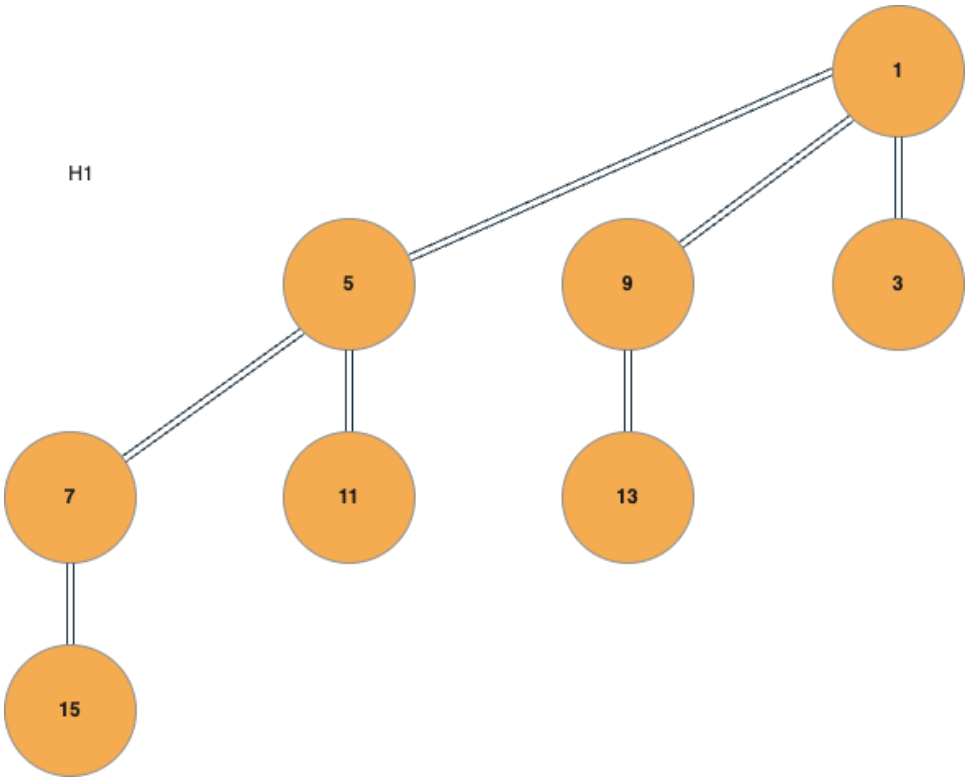
   (c) Create another binomial min-heap $H3$ by inserting the following numbers from left to right: 8, 12, 4, 2. (2 points)

   (d) Merge $H2$ and $H3$ to form a new binomial heap, $H4$. (2 points)

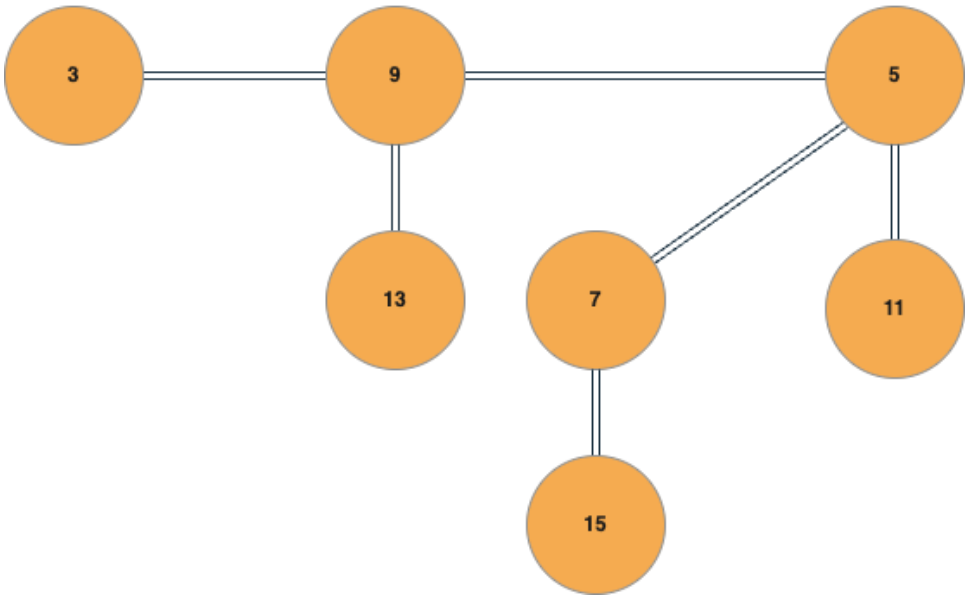
   (e) Perform two `deleteMin()` operations on $H4$ to obtain $H5$. (2 points)

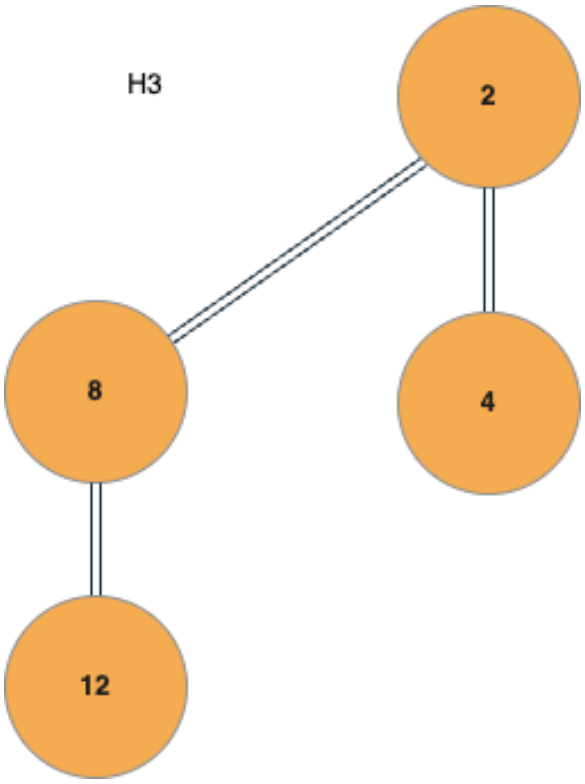
   Note: It is optional to show the intermediate steps in your submission. Only the five final binomial heaps ($H1$, $H2$, $H3$, $H4$, and $H5$) will be considered for grading. So, please ensure that you clearly illustrate your final binomial heaps ($H1$, $H2$, $H3$, $H4$, and $H5$). You can use online tools like draw.io for drawing these heaps.

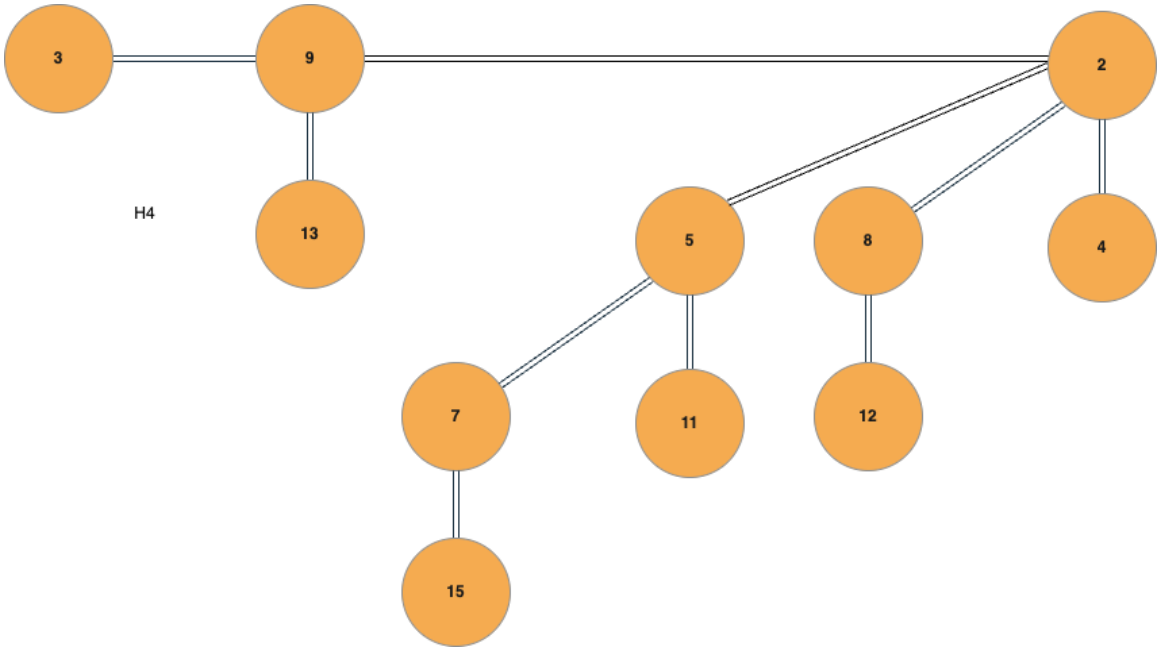
H1

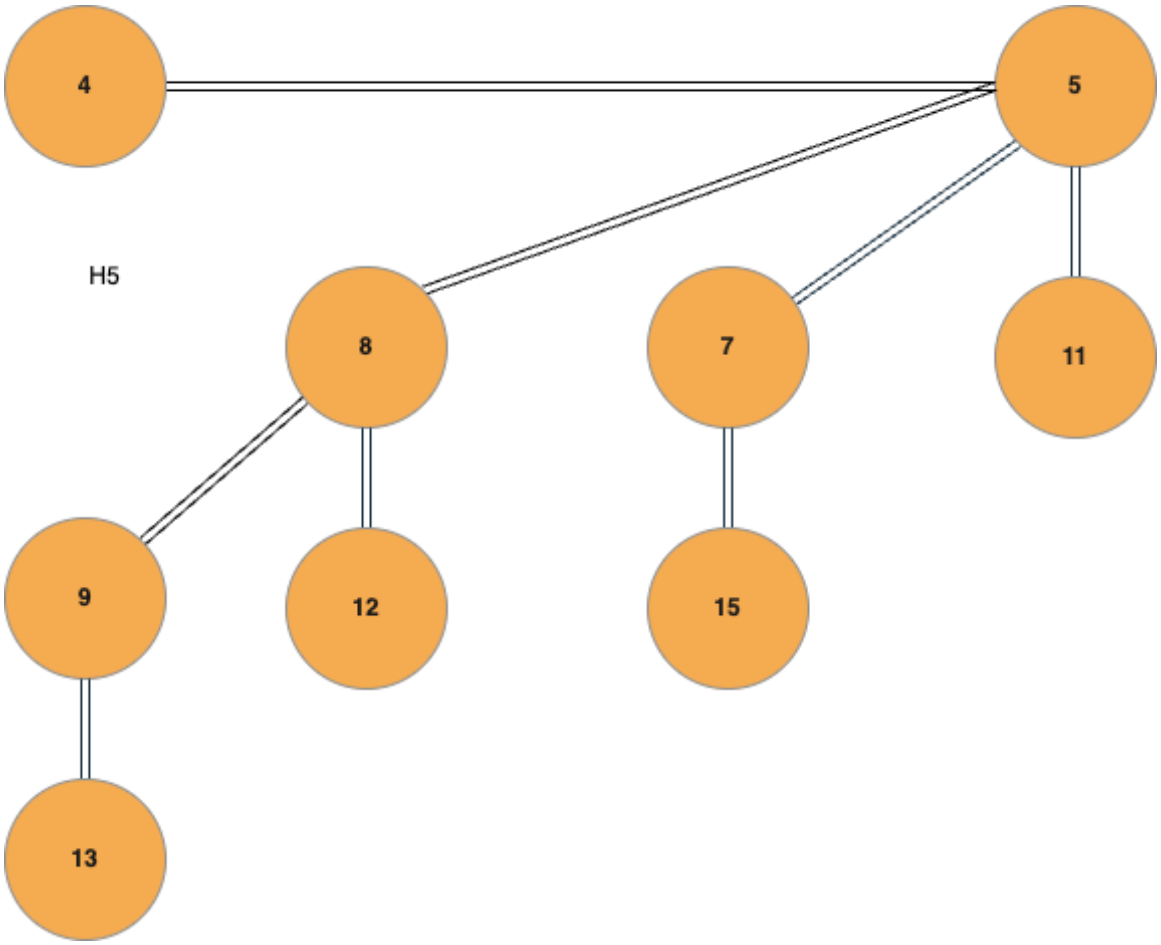


H2




H3
2
8
4
12


3
9
2
H4
13
5
8
4
7
11
12
15



5. If we have a k-th order binomial tree (B_k), which is formed by joining two B_{k-1} trees, then when we remove the root of this k-th order binomial tree, it results in k binomial trees of smaller orders. Prove by mathematical induction. (10 points)

   Proof by Mathematical Induction:

   Base Case (k = 1): If we remove the root node of a B_1 binomial tree, we will indeed get 1 B_0 binomial tree. This is our base case.

   Inductive Hypothesis: Assume that if we remove the root node of a B_{k-1} order binomial tree, we will get $k-1$ binomial trees of smaller orders.

   Inductive Step: We want to prove that when we remove the root of a B_k order binomial tree, it results in k binomial trees of smaller orders.

A B_k order binomial tree is formed by joining two B_{k-1} order binomial trees, and the root node of the B_k order binomial tree comes from either one of the two B_{k-1} order binomial trees.

When we remove the root node of this B_k order binomial tree, we will get:

(a) One binomial tree of order B_{k-1}: This comes from the subtree that did not have the root node of the B_k order binomial tree.

(b) Several smaller order binomial trees: These are equivalent to the binomial trees formed when removing the root node of a B_{k-1} order binomial tree.

Now, according to our inductive hypothesis, when we remove the root node of a B_{k-1} order binomial tree, we get $k-1$ smaller order binomial trees.

Therefore, the total number of binomial trees we get after removing the root node of a B_k order binomial tree is:

1 (B_{k-1} binomial tree) + $(k-1)$ (smaller order binomial trees from the inductive hypothesis) $= k$.

So, we have shown that when we remove the root of a B_k order binomial tree, it indeed results in k binomial trees of smaller orders, as required. This completes the proof by mathematical induction.

6. Given a weighted undirected graph with all distinct edge costs. Design an algorithm that runs in $O(V+E)$ to determine if a particular edge e is contained in the minimum spanning tree of the graph. Pseudocode is not required, and you can use common graph algorithms such as DFS, BFS, and Minimum Spanning Tree Algorithms as subroutines without further explanation. You are not required to prove the correctness of your algorithm. (10 points)

Let $e = (u, v)$. Delete all edges of weight greater than or equal to the cost of e. Run DFS on the new graph to see if there is a path between u and v. This can be done by checking if u and v are in the same connected component. Edge e is contained in the minimum spanning tree in the original graph if and only if there is no path between u and v. Deleting

edges takes $O(E)$, and running DFS takes $O(V + E)$, so the runtime is $O(V + E)$.

7. Given a weighted undirected graph with all distinct edge costs and $E = V + 10$. Design an algorithm that outputs the minimum spanning tree of the graph and runs in $O(V)$. Pseudocode is not required, and you can use common graph algorithms such as DFS, BFS, and Minimum Spanning Tree Algorithms as subroutines without further explanation. You are not required to prove the correctness of your algorithm. (10 points)

   Start with the original graph. Run DFS to detect a cycle. Delete the edge with the greatest cost in the cycle. Repeat the process (11 times) until the graph becomes a tree (no cycles). The runtime is $O(11(V + E)) = O(11(2V + 10)) = O(V)$

8. There are N people with the i-th person's weight being w_i. A boat can carry at most two people under the max weight limit of $M \geq \max_i w_i$. Design a greedy algorithm that finds the minimum number of boats that can carry all N people. Pseudocode is not required, and you can assume the weights are sorted. Use mathematical induction to prove that your algorithm is correct. (10 points)

Let the heaviest and the lightest person share a boat if allowed; otherwise, Let the heaviest person take a boat alone. Repeat the process until everyone is assigned to a boat. Here is the pseudocode.

Algorithm 1 Problem 8

```
1: weights.sort(); // not required if the weights are sorted
2: i = 0, j = N − 1, boat = 0;
3: while i ≤ j do
4:     if i < j and weights[i] + weights[j] ≤ M then
5:         i ← i + 1;
6:     end if
7:     j ← j − 1;
8:     boat ← boat + 1;
9: end while
10: return boat
```

We prove the correctness by induction on the number of people.

Base Case (N= 1) The algorithm outputs 1 and it is optimal

Inductive Hypothesis: Suppose the algorithm is correct for any $N < k$.

Inductive Step: We want to prove the algorithm is correct for $N = k$. If the heaviest person cannot take a boat with the lightest person, the heaviest person is not able to take a boat with any other person. Therefore, the heaviest person must take a boat alone in every solution. Thus, the algorithm is optimal as it is optimal for arranging the remaining $k - 1$ people by the induction hypothesis.

Otherwise, our algorithm lets the heaviest person take a boat with the lightest person. If in an optimal solution, the heaviest person takes a boat alone or takes a boat with the lightest person. Then, our algorithm is again optimal by the induction hypothesis. If not, suppose in the optimal solution the heaviest person takes a boat with another person called p. In this case, One can exchange p with the lightest person in our solution so

that the solution is still valid. This is because p can take a boat with the heaviest person given by the optimal solution, and the lightest person can take the boat p originally takes as the lightest person is not heavier than p. Therefore, our solution is again optimal by the induction hypothesis.

9. Given $N > 1$ integer arrays with each array having at most M numbers, you are asked to select two numbers from two *distinct* arrays. Your goal is to find the maximum difference between the two selected numbers among all possible choices. Provide an algorithm that finds it in $O(NM)$ time. Pseudocode is not required, and you can use common operations for arrays, such as min and max, without further explanation. Prove that your algorithm is correct. You may find proof by contradiction helpful when proving the correctness. (10 points)

   Let $v_{\max}$ be the maximum value among all the arrays and assume $v_{\max}$ first appears in the $i_{\max}$-th array (may appear in other arrays as well). Then the answer is the maximum of the following:

   (a) $v_{\max} - v'_{\min}$, where $v'_{\min}$ is the minimum value among all the arrays other than the $i_{\max}$-th array.

   (b) $v'_{\max} - v''_{\min}$, where $v'_{\max}$ is the maximum value among all the arrays other than the $i_{\max}$-th array, and $v''_{\min}$ in the minimum value in the $i_{\max}$-th array.

   All of these numbers can be found in $O(NM)$ time. To prove the correctness of the algorithm, it is not hard to see our solution is valid. It remains to show the difference is maximum. Suppose the difference is suboptimal and an optimal solution is $v^\star_{\max} - v^\star_{\min}$, where $v^\star_{\max}$ is found in the $i^\star_{\max}$-th array and $v^\star_{\min}$ is found in the $i^\star_{\min}$-th array. We consider two cases: $i^\star_{\min} = i_{\max}$ and $i^\star_{\min} \neq i_{\max}$. If $i^\star_{\min} = i_{\max}$, we have

$$\begin{aligned} v^\star_{\max} - v^\star_{\min} &\leq v^\star_{\max} - v''_{\min} && (v''_{\min} \text{ is minimum value in array } i_{\max}) \\ &\leq v'_{\max} - v''_{\min}. && (i^\star_{\min} = i_{\max} \neq i^\star_{\max}) \end{aligned}$$

If $i^\star_{\min} \neq i_{\max}$, we have

$$\begin{aligned} v^\star_{\max} - v^\star_{\min} &\leq v_{\max} - v^\star_{\min} && (v_{\max} \text{ is maximum among all arrays}) \\ &\leq v_{\max} - v'_{\min}. && (i^\star_{\min} \neq i_{\max}) \end{aligned}$$

It contradicts the assumption that the solution is suboptimal, which finishes the proof.

10. There are N cities (city 1, to city N) and some flights between these cities. Specifically, there is a direct flight from every city i to city $2i$ (no direct flight from city $2i$ to city i) and another direct flight from every city i to city $i-1$ (no direct flight from city $i-1$ to city i). Given integers a and b, determine if there exists a sequence of flights starting from city a to city b. If so, find the minimum number of flights required to fly from city a to city b. For example, when $N = 10$, $a = 3$, and $b = 9$, the answer is 4 and the corresponding flights are $3 \to 6 \to 5 \to 10 \to 9$. You are not required to prove the correctness of your algorithm. (10 points)

The only case that there doesn't exist a solution is when $a \neq b$, $b = N$, and N is odd. If $b \leq a$, the answer is $a - b$. Otherwise, do the calculation backward: Start from b. Greedily divide by 2 when the number is even or add 1 when the number is odd until $b \leq a$. Let $b' \leq a$ be the resulting number after n backward operations. Then the answer is $n + a - b'$.

Algorithm 2 Problem 10

```
1: if a ≠ b and b = N and N%2 = 1 then
2:     return "no solution"
3: end if
4: count = 0;
5: while b > a do
6:     if b%2 = 0 then
7:         b ← b/2;
8:     else
9:         b ← b + 1
10:    end if
11:    count ← count + 1
12: end while
13: return count + a − b
```