CS-3510-02 Algorithms, Spring 2025 Mid-term 2 take home part
Hayden Hawley Collaborator(s):

Academic Integrity Requirement: You must write your solution in your own words. Do not copy from external sources, classmates, or generative AI tools.

Problem 1 (10 pts)

An independent set I in an undirected graph $G = (V, E)$ is a subset $I \subseteq V$ of vertices such that no two vertices in I are joined by an edge of E. Consider the following greedy algorithm to try to find a maximum size independent set which is based on the general idea that choosing vertices with small degree to be in I will rule out fewer other vertices:

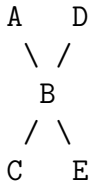
Algorithm 1 GreedyIndependentSet

```
1: I ← ∅
2: while G is not empty do
3:     Choose a vertex v of smallest degree in G        ▷ Not counting deleted edges
4:     I ← I ∪ {v}
5:     Delete v and all its neighbors and their connected edges from G
6:                          ▷ None of the neighbors can be included since v is included
7: end while
8: return I
```

Prove that this algorithm is incorrect by counterexample. Specifically, give an example of a graph on which this algorithm does not produce a largest size independent set. Show both the independent set that the algorithm finds and a larger independent set.

Solution:

The greedy algorithm does NOT always find the biggest possible independent set. Consider this graph:



Edges: A-B, B-C, B-D, B-E

- All outer nodes (A, C, D, E) have degree 1. B has degree 4.
- Greedy would pick A first (lowest degree).
- It then removes A and B.
- We're left with C, D, and E. The final independent set is C, D, E.

But the actual largest solution is A, C, D, E. 4 nodes, and none are connected. Therefore, this greedy algorithm picked a smaller set than was possible.

Problem 2 (10 pts)

Suppose A is an array of n integers that is a strictly decreasing sequence, followed by a strictly increasing sequence such as $[12, 9, 8, 6, 3, 4, 7, 9, 11]$. Give an $O(\log n)$ algorithm to find the minimum element of the array. Justify your algorithm is correct.

Solution:

This is a simple binary search problem. You can find the lowest point by recursively splitting the array in half.

1. Start with the whole array, which can really be thought of as two concatenated pre-sorted arrays. We're trying to find the value which is smaller than the values to its left AND right.
2. Look at the middle (index $\text{len}(A)/2$) element:
   - If it's **greater** than the element after it, the valley is to the **right**. Throw away the left half.
   - If it's **smaller** than the next element, check if it's also smaller than the previous element. If so, we found the minimum.
   - Otherwise, the valley is to the **left**. Throw away the right half.
3. Repeat until we find the minimum.

Since we halve the array at each step, the runtime is $O(\log n)$ like any binary search problem.

Problem 3 (10 pts)

Consider a graph that is a path, where the vertices are $v_1, v_2, \ldots, v_n$, with edges between v_i and v_{i+1}. Suppose that each node v_i has an associated weight w_i. Give an algorithm that takes an n-vertex path with weights and returns an independent set of maximum total weight. The runtime of the algorithm should be polynomial in n. Justify your algorithm is correct.

Solution:

We have a path of weighted nodes, and we want to choose a set of non-adjacent nodes such that the total weight is maximized. We can use dynamic programming to break the problem down.

At each node i, we have two options:

- **Skip** the current node and take the best total from $i - 1$
- **Take** the current node and add its weight $w[i]$ to the best total from $i - 2$

Pseudocode:

```
dp[0] = w[0]
dp[1] = max(w[0], w[1])
for i from 2 to n-1:
    dp[i] = max(dp[i-1], dp[i-2] + w[i])
```

This algorithm runs in $O(n)$ time.