Stefanie Shidoosh

CS180: HW #5

1. Give an algorithm to print a paragraph of n words on a printer in a way that minimizes the above sum. Analyze the running time and space requirements of your algorithm.

    Let M[i, j] be a 2D array, Let i and j be indexes of the input with respect to each word
    Each cell value represents the cost to store words i … j in a single line
    Compute the costs of all possible lines in M
        Calculate extra space for each line
        Calculate the line that is associated with the aforementioned space
        Calculate the minimum cost

    Create an array that stores the optimal cost of arranging the words i..j, recursively recording by each line:

$$array[j] = \begin{cases} 0 \quad (\text{if j} = 0) \\ \min\,(c[i-1] + l_c[i,j] \quad (\text{if j} > 0) \\ 1 \leq i \ \leq j \end{cases}$$

    Output the words with the calculated, optimally specified ranges of words i …j with respect to the array

    The time complexity of this algorithm is $O(n^2)$, as the costliest operation is iterating through the 2D array to calculate the extra space, and then to calculate the line associated with that space, requiring a nested loop capability.

Stefanie Shidoosh

2a. Design a divide-and-conquer algorithm for solving the longest ascending subsequence problem in time $O(n^2)$.

```
Divide array A into 2 arrays, $A_0$ and $A_1$.
Find the longest consecutive subsequence, $s_0$, of sub array $A_0$
Find the longest consecutive subsequence, $s_1$, of sub array $A_1$
If the longest consecutive sub sequence is split between $A_0$ and $A_1$ as a resulting of dividing
    The consecutive sub sequence, $s_3$, must contain A1[n/2] and A2[n/2-(n/2)-1))]
The longest subsequence of the three subsequences, $s_0, s_1, s_2$, is the longest subsequence of A.
```

This algorithm has a time complexity of $O(n^2)$, as every time the algorithm visits an element in A, it has to compare that element with the next element in A to check if it may result in an ascending sequence.

2b. Design a dynamic programming algorithm in time $O(n\log n)$

```
Create an array, $A_0$
Let T be an empty tree
Iterate through A for all n elements
        If current element is larger than the last element in $A_0$, append this element to $A_0$ (If $A_0$ is empty, insert as first element)
                Insert the current element to T such that the last element in $A_0$ is its parent (If T is empty, insert the current element as the root).
        If the current element is smaller than the last element $A_0$, replace the last element with the current element
Use binary search to take the last element on $A_0$ and follow its parent pointers until root node is reached
The nodes visited in this order is the increasing ascending subsequence.
```

Stefanie Shidoosh

3. Given a weighted undirected graph $G = (V, E)$, where $|V| = n$ and edge weight are distinct. Design an algorithm to solve MST in $O(n\log n)$ *rounds*. Values in the input to the problem are of size $O(\log n)$ bits.

```
For all subgraphs, $G_i$ of G
    Iterate through all v in V($G_i$)
    If v has yet to be selected,
        Mark the cheapest outgoing edge of all such v in V($G_i$)
        Choose only one cheapest edge for each vertex
    Append cheapest edges to tree T, ultimately resulting in the MST
```

The algorithm takes $O(nlogn)$ rounds as its input is $O(logn)$ bits, so at each assessment of "cheapness," the algorithm considers $\log(n)$ bits. It needs to do so for all vertices, which in total is n. Therefore, $O(n) * O(logn) = O(nlogn)$.

Stefanie Shidoosh

4. What will the obvious greedy algorithm be for Knapsack? Give a counter example to show that Knapsack is not amenable to a greedy solution. Show that nevertheless if the Hiker is in a hurry and she fills the Knapsack using the greedy algorithm then the value she carries is greater equal than half of the optimal value.

The obvious greedy algorithm would be:
As long as there does not exist an item that can be put in the knapsack such that its addition will not exasperate the knapsack weight capacity:

```
    Iterate through all items, find the lowest weight/value (for all items
    weight=value)
    Put the item in the knapsack
```

This algorithm is not amenable to the Knapsack problem, as demonstrated by the following counter-example: Say the max capacity is 25, and the following items are given,

Weight	1	2	5	8	10
Volume	1	2	5	8	10

The better items to take would be 2, 5, 8, and 10 for a total of 25. The greedy algorithm would select 1, 2, 5, and 8 for a total of 16. Therefore, this algorithm is not an effective solution for the knapsack problem.

The current value is greater than or equal to the optimal value/2 in this case ($12.5 \leq 16$). This is generally the case as with each selection of the cheapest item, the selection of the most expensive item is thrown out. Therefore, the maximum possible value must be greater than or equal to the half of the optimal value as with each iteration the choices are reduced by half.