

CS 1510 Midterm 1
Fall 2005

Note that the only difference in the first two problems is the definition of *total penalty*.

1. The input to this problem consists of an ordered list of n words. The length of the i th word is w_i , that is the i th word takes up w_i spaces. (For simplicity assume that there are no spaces between words.) The goal is to break this ordered list of words into lines, this is called a layout. Note that you can not reorder the words. The length of a line is the sum of the lengths of the words on that line. The ideal line length is L . No line may be longer than L , although it may be shorter. The penalty for having a line of length K is $L - K$. *The total penalty is the **sum** of the line penalties.* The problem is to find a layout that minimizes the total penalty.

Prove or disprove that the following greedy algorithm correctly solves this problem. A proof of correctness must use an exchange argument.

```
For i= 1 to n
    Place the ith word on the current line if it fits
    else place the ith word on a new line
```

2. The input to this problem consists of an ordered list of n words. The length of the i th word is w_i , that is the i th word takes up w_i spaces. (For simplicity assume that there are no spaces between words.) The goal is to break this ordered list of words into lines, this is called a layout. Note that you can not reorder the words. The length of a line is the sum of the lengths of the words on that line. The ideal line length is L . No line may be longer than L , although it may be shorter. The penalty for having a line of length K is $L - K$. The total penalty is the **maximum** of the line penalties. The problem is to find a layout that minimizes the total penalty.

Prove or disprove that the following greedy algorithm correctly solves this problem. A proof of correctness must use an exchange argument.

```
For i= 1 to n
    Place the ith word on the current line if it fits
    else place the ith word on a new line
```

3. We consider the problem of computing the longest increasing subsequence of a sequence x_1, \dots, x_n of integers. We considered this problem several times during class.
 - (a) (10 points) Give C-like pseudo-code to compute the length of the longest increasing subsequence. The running time of your code should be $O(n^2)$. Include some explanation of your variables and data structures, and of your code in general. I suggest you read the following items in this question before you answer this one.
 - (b) (5 points) Explain how to in $O(n)$ time compute the actual subsequence, that is the longest increasing subsequence, from the data structure constructed by your code in part a.

- (c) (5 points) Give C-like pseudo-code to compute the length of the longest increasing subsequence in $O(n^2)$ time and $O(n)$ space. So you can not afford to use an n by n array.
4. Give an algorithm for the following problem whose running time is polynomial in $n + L$, where $L = \max(\sum_{i=1}^n v_i^3, \prod_{i=1}^n v_i)$.
- Input: positive integers v_1, \dots, v_n
- Output: 1 if there exists a subset S of the integers such that $\sum_{v_i \in S} v_i! = \prod_{v_i \in S} v_i^3$. And 0 otherwise.
- That is, you want to determine if there exists a subset S where the sum of the factorials of the numbers in S is equal to the product of the the cubes of the numbers in S .
5. Give an algorithm for the following problem whose running time is polynomial in n . The input is an n sided convex polygon. Assume that the polygon is specified by the Cartesian coordinates of its vertices. The output should be the triangulation of the polygon into $n - 2$ triangles that minimizes the sums of the perimeters of the into triangles. Note that this is equivalent to minimizing the length of the cuts required to create the triangle. This is exactly the problem that was in the homework.