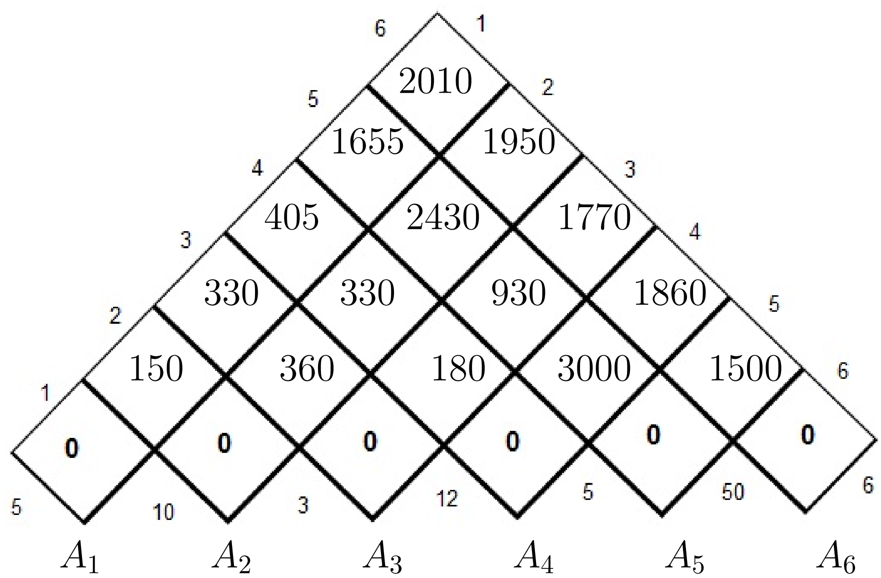
# Chapter 15:
# Matrix Chain Multiplication,
# Bitonic Traveling Salesman Problem,
# and Printing Neatly

(a) Matrix Chain Multiplication

**15.2-1)** Find an optimal parenthesization of a matrix-chain product whose sequence of dimension is: (5, 10, 3, 12, 5, 50, 6)
**Solution:**




**Definition 1** *The minimum cost of parenthesizing the product* $A_iA_{i+1}\cdots A_j$ *becomes*

$$m[i,j] = \begin{cases} 0 & if\ i = j \\ \min\limits_{i\le k<j} \{m[i,k] + m[k+1,j] + p_{i-1}p_kp_j\} & if\ i < j \end{cases}$$

- $m[1,2] = \min\limits_{1\le k<2} \{m[1,k] + m[k+1,2] + p_0p_kp_2\}$
  $= \min\limits_{k=1} \{m[1,1] + m[2,2] + p_0p_1p_2 = 0 + 0 + 5\cdot 10\cdot 3 = 150$

- $m[2,3] = \min\limits_{2\leq k<3} \{m[2,k] + m[k+1,3] + p_1p_kp_3\}$
  $= \min\limits_{k=2} \{m[2,2] + m[3,3] + p_1p_2p_3 = 0 + 0 + 10 \cdot 3 \cdot 12 = 360$
- $m[3,4] = \min\limits_{3\leq k<4} \{m[3,k] + m[k+1,4] + p_2p_kp_4\}$
  $= \min\limits_{k=3} \{m[3,3] + m[4,4] + p_2p_3p_4 = 0 + 0 + 3 \cdot 12 \cdot 5 = 180$
- $m[4,5] = \min\limits_{4\leq k<5} \{m[4,k] + m[k+1,5] + p_3p_kp_5\}$
  $= \min k = 4 \{m[4,4] + m[5,5] + p_3p_4p_5 = 0 + 0 + 12 \cdot 5 \cdot 50 = 3000$
- $m[5,6] = \min\limits_{5\leq k<6} \{m[5,k] + m[k+1,6] + p_4p_kp_6\}$
  $= \min\limits_{k=5} \{m[5,5] + m[6,6] + p_4p_5p_6 = 0 + 0 + 5 \cdot 50 \cdot 6 = 1500$

- $m[1,3] = \min\limits_{1\leq k<3} \{m[1,k] + m[k+1,3] + p_0p_kp_3\}$

$$= \min_{1\leq k<3} \left\{ \begin{array}{l} k = 1 \ \{m[1,1] + m[2,3] + p_0p_1p_3 = 0 + 360 + 5 \cdot 10 \cdot 12 = 960 \\ \mathbf{k = 2} \ \{m[1,2] + m[3,3] + p_0p_2p_3 = 150 + 0 + 5 \cdot 3 \cdot 12 = \mathbf{330} \end{array} \right\} = 330.$$

- $m[2,4] = \min\limits_{2\leq k<4} \{m[2,k] + m[k+1,4] + p_1p_kp_4\}$

$$= \min_{2\leq k<4} \left\{ \begin{array}{l} \mathbf{k = 2} \ \{m[2,2] + m[3,4] + p_1p_2p_4 = 0 + 180 + 10 \cdot 3 \cdot 5 = \mathbf{330} \\ k = 3 \ \{m[2,3] + m[3,4] + p_1p_3p_4 = 360 + 0 + 10 \cdot 12 \cdot 5 = 960 \end{array} \right\} = 330.$$

- $m[3,5] = \min\limits_{3\leq k<5} \{m[3,k] + m[k+1,5] + p_2p_kp_5\}$

$$= \min_{3\leq k<5} \left\{ \begin{array}{l} k = 3 \ \{m[3,3] + m[4,5] + p_2p_3p_5 = 0 + 3000 + 3 \cdot 12 \cdot 50 = 4800 \\ \mathbf{k = 4} \ \{m[3,4] + m[5,5] + p_2p_4p_5 = 180 + 0 + 3 \cdot 5 \cdot 50 = \mathbf{930} \end{array} \right\} = 930.$$

- $m[4,6] = \min\limits_{4\leq k<6} \{m[4,k] + m[k+1,6] + p_3p_kp_6\}$

$$= \min_{3\leq k<5} \left\{ \begin{array}{l} \mathbf{k = 4} \ \{m[4,4] + m[4,6] + p_3p_4p_6 = 0 + 1500 + 12 \cdot 5 \cdot 6 = \mathbf{1860} \\ k = 5 \ \{m[4,5] + m[6,6] + p_3p_5p_6 = 180 + 0 + 12 \cdot 50 \cdot 6 = 6600 \end{array} \right\} = 1860.$$

- $m[1,4] = \min\limits_{1\leq k<4} \{m[1,k] + m[k+1,4] + p_0p_kp_4\}$

$$= \min_{1\leq k<4} \left\{ \begin{array}{l} k = 1 \ \{m[1,1] + m[2,4] + p_0p_1p_4 = 0 + 330 + 5 \cdot 10 \cdot 5 = 580 \\ \mathbf{k = 2} \ \{m[1,2] + m[3,4] + p_0p_2p_4 = 150 + 180 + 5 \cdot 3 \cdot 5 = \mathbf{405} \\ k = 3 \ \{m[1,3] + m[4,4] + p_0p_3p_4 = 330 + 0 + 5 \cdot 12 \cdot 5 = 630 \end{array} \right\} = 405.$$

- $m[2,5] = \min\limits_{2\leq k<5} \{m[2,k] + m[k+1,5] + p_1p_kp_5\}$

$$= \min_{2\leq k<5} \left\{ \begin{array}{l} \mathbf{k = 2} \ \{m[2,2] + m[3,5] + p_1p_2p_5 = 0 + 930 + 10 \cdot 3 \cdot 50 = \mathbf{2430} \\ k = 3 \ \{m[2,3] + m[4,5] + p_1p_3p_5 = 360 + 3000 + 10 \cdot 12 \cdot 50 = 9360 \\ k = 4 \ \{m[2,4] + m[5,5] + p_1p_4p_5 = 330 + 0 + 10 \cdot 5 \cdot 50 = 2830 \end{array} \right\} = 2430.$$

- $m[3,6] = \min\limits_{3\leq k<6} \{m[3,k] + m[k+1,6] + p_3p_kp_6\}$

$$= \min_{3\leq k<6} \left\{ \begin{array}{l} k = 3 \ \{m[3,3] + m[4,6] + p_2p_3p_6 = 0 + 1860 + 3 \cdot 12 \cdot 6 = 2076 \\ \mathbf{k = 4} \ \{m[3,4] + m[5,6] + p_2p_4p_6 = 180 + 1500 + 3 \cdot 5 \cdot 6 = \mathbf{1770} \\ k = 5 \ \{m[3,5] + m[6,6] + p_2p_5p_6 = 930 + 0 + 13 \cdot 50 \cdot 6 = 1830 \end{array} \right\} = 1770.$$

- $m[1,5] = \min\limits_{1 \le k < 5} \{m[1,k] + m[k+1,5] + p_0 p_k p_5\}$

$$= \min_{1 \le k < 5} \left\{ \begin{array}{l} k = 1 \; \{m[1,1] + m[2,5] + p_0p_1p_5 = 0 + 2430 + 5 \cdot 10 \cdot 50 = 4930 \\ k = 2 \; \{m[1,2] + m[3,5] + p_0p_2p_5 = 150 + 930 + 5 \cdot 3 \cdot 50 = 1830 \\ k = 3 \; \{m[1,3] + m[4,5] + p_0p_3p_5 = 330 + 3000 + 5 \cdot 12 \cdot 50 = 6330 \\ \mathbf{k = 4} \; \{m[1,4] + m[5,5] + p_0p_4p_5 = 405 + 0 + 5 \cdot 5 \cdot 50 = \mathbf{1655} \end{array} \right\} = 1655.$$

- $m[2,6] = \min\limits_{2 \le k < 6} \{m[2,k] + m[k+1,6] + p_1 p_k p_6\}$

$$= \min_{2 \le k < 6} \left\{ \begin{array}{l} \mathbf{k = 2} \; \{m[2,2] + m[3,6] + p_1p_2p_6 = 0 + 1770 + 10 \cdot 3 \cdot 6 = \mathbf{1950} \\ k = 3 \; \{m[2,3] + m[4,6] + p_1p_3p_6 = 360 + 1860 + 10 \cdot 12 \cdot 6 = 2940 \\ k = 4 \; \{m[2,4] + m[5,6] + p_1p_4p_6 = 330 + 1500 + 10 \cdot 15 \cdot 6 = 2130 \\ k = 5 \; \{m[2,5] + m[6,6] + p_1p_5p_6 = 2430 + 0 + 10 \cdot 50 \cdot 6 = 5430 \end{array} \right\} = 1950.$$

- $m[1,6] = \min\limits_{1 \le k < 6} \{m[1,k] + m[k+1,6] + p_0 p_k p_6\}$

$$= \min_{1 \le k < 6} \left\{ \begin{array}{l} k = 1 \; \{m[1,1] + m[2,6] + p_0p_1p_6 = 0 + 1950 + 5 \cdot 10 \cdot 6 = 2250 \\ \mathbf{k = 2} \; \{m[1,2] + m[3,6] + p_0p_2p_6 = 150 + 1770 + 5 \cdot 3 \cdot 6 = \mathbf{2010} \\ k = 3 \; \{m[1,3] + m[4,6] + p_0p_3p_6 = 330 + 1860 + 5 \cdot 12 \cdot 6 = 2550 \\ k = 4 \; \{m[1,4] + m[5,6] + p_0p_4p_6 = 405 + 1500 + 5 \cdot 5 \cdot 6 = 2055 \\ k = 5 \; \{m[1,5] + m[6,6] + p_0p_5p_6 = 1655 + 0 + 5 \cdot 50 \cdot 6 = 3155 \end{array} \right\} = 2010.$$

multiplication sequence:
s[1,6]=2 $(A_1A_2)(A_3A_4A_5A_6)$
s[3,6]=4 $(A_1A_2)((A_3A_4)\ (A_5A_6))$

**optimal parenthesization of the matrix-chain product:** $((A_1A_2)((A_3A_4)\ (A_5A_6)))$

**15.2-2)** Give a recursive algorithm MATRIX-CHAIN-MULTIPLY(A,s,i,j) that actually performs the optimal matrix -chain multiplication, given the sequence of matrices $(A_1, A_2, ..., A_n)$, the s table computed by MATRIX-CHAIN-ORDER, and the indices i and j. (The initial call would be MATRIX-CHAIN-MULTIPLY(A,s,1,n)).

**Solution:**

```
MATRIX-CHAIN-MULTIPLY(A, s, i, j)
{
    if(i<j)
   {
        X= MATRIX-CHAIN-MULTIPLY(A, s, i, s[i,j]);
        Y= MATRIX-CHAIN-MULTIPLY(A, s, s[i,j]+1, j);
        return X * Y;
   }
    else
        return A[i];
}
```

where A is the array containing $A_1, A_2, ..., A_n$,
s is an $n$ x $n$ array such that s[i,j]=$k$ for $m[i,j] = m[i,k] + m[k+1,j] + p_{i-1}p_kp_j$,
and X $*$ Y denotes matrix multiplication

(b) Bitonic Traveling Salesman Problem

**15-1)** Describe an O($n^2$)-time algorithm for determining the optimal bitonic tour. You may assume that no two points have the same x-coordinate. Hint: scan left to right, maintaining optimal possibilities for the two parts of the tour

**Solution:** Let $\{p_1, p_2, p_3, \ldots, p_n\}$ be a set of points on a cartesian plane. A **tour** is a cycle where $p_1, p_2, \ldots,$ and $p_n$ are all part of the cycle. That is, "it is a simple path with no repeated vertices or edges other than the starting and ending vertices" (Wikipedia). A tour is bitonic if the path starting from the left most point, moves strictly from left to right back to the rightmost point and then strictly from right to left. A **bitonic tour** is said to be optimal if it is a bitonic tour of minimal length. Let $p_i = (x_i, y_i)$, $n$ be the number of points to be connected on the plane and $p_1, p_2, \ldots, p_n$ be the list of points sorted in ascending order according to their $x$-coordinate.

**Claim 1**: $p_n$ and $p_{n-1}$ are neighbors in any bitonic tour containing points $p_1, p_2, \ldots, p_n$.
Proof: Assume $p_{n-1}$ is not a neighbor of $p_n$. Let $p_i$ and $p_j$ be the two neighbors of $p_n$. Then, the points must be visited in the order $p_i, p_n, p_j, p_{n-1}$. The path $p_i \rightsquigarrow p_n$ moves from left to right, the path $p_n \rightsquigarrow p_j$ moves from right to left, and the path $p_j \rightsquigarrow p_{n-1}$ moves from left to right. Thus, this tour is not bitonic. Therefore, $p_n$ and $p_{n-1}$ are neighbors in any bitonic tour containing points $p_1, p_2, \ldots, p_n$.

Necessarily, a minimal bitonic tour must contain edge $\overline{p_{n-1}p_n}$. Let $P$ be a minimal bitonic path from $p_{n-1}$ to $p_n$ obtained by removing the edge $\overline{p_{n-1}p_n}$. Since $\overline{p_{n-1}p_n}$ exists in any bitonic tour, finding the minimal bitonic tour from $p_{n-1}$ to $p_n$ is equivalent to finding the minimal bitonic path from $p_{n-1}$ to $p_n$. Note that the bitonic path, $P$, visits points $p_1, p_2, \ldots, p_n$. Let $b_k$ be the neighbor of $p_n$. By removing $p_n$, we are left with the minimal bitonic path $P'$ that visits $p_1, p_2, \ldots, p_k, \ldots, p_{n-1}$.

**Claim 2**: The minimal bitonic path has an optimal substructure. (A minimal bitonic path containing points $p_1, p_2, \ldots, p_n$. contains a minimal bitonic subpath containing points $p_1, p_2, \ldots, p_{n-1}$)
Proof: let $\mathcal{P}$ be the minimal bitonic path with endpoints $p_i$ and $p_j$ with $i < j$. Let $p_k$ be a neighbor of $p_j$. Then the path $\mathcal{P}'$ obtained by removing $p_j$ from $\mathcal{P}$ is a normal bitonic path with endpoints $p_i$ and $p_k$. Necessarily, $p_{j-1} \in \{p_i, p_k\}$.

Proof: Assume $p_{j-1} \notin \{p_i, p_k\}$
$\implies$ path $p_i p_{j-1} p_k p_j$ ($p_k$ and $p_j$ are neighbors and $p_i$ and $p_j$ are endpoints).
$\implies$ $p_i \rightsquigarrow p_{j-1}$ moves from left to right because $i < j-1 < j$ and $i \neq j-1$,
$p_{j-1} \rightsquigarrow p_k$ moves from right to left because $k < j-1 < j$, and
$p_k \rightsquigarrow p_j$ moves from left to right because $k < j$
$\implies$ the path $p_i p_{j-1} p_k p_j$ is not bitonic. This is a contradiction and thus, $p_{j-1} \in \{p_i, p_k\}$.

Path $\mathcal{P}'$ is the minimal bitonic path containing points $p_1, p_2 \ldots, p_{j-1}$.

Proof: Assume $\mathcal{P}'$ is not a minimal bitonic path. Then there exists a shorter bitonic path, $\mathcal{P}''$, with endpoints $p_i$ to $p_k$. We can construct $\mathcal{P}'''$, a path from $p_i$ to $p_j$ by appending $\overline{p_k p_j}$ to $\mathcal{P}''$. Note that $\mathcal{P}$ is a minimal bitonic path from $p_i$ to $p_j$. We assumed that $\mathcal{P}''$ is shorter than $\mathcal{P}'$

$\implies \ell(\mathcal{P}'') < \ell(\mathcal{P}')$
$\implies \ell(\mathcal{P}'') + ||\overline{p_k p_j}|| < \ell(P') + ||\overline{p_k p_j}||$
$\implies \ell(P''') = \ell(P'') + ||\overline{p_k p_j}|| < \ell(P') + ||\overline{p_k p_j}|| = \ell(P)$
$\implies \ell(P''') < \ell(P)$
This is a contradiction because $\mathcal{P}$ is a minimal bitonic path.
Therefore, $\nexists$ a shorter normal bitonic path with endpoints $p_i$ and $p_j$ different from $\mathcal{P}'$. Thus, $\mathcal{P}'$ is the minimall bitonic subpath with endpoints $p_i$ and $p_k$ from path $\mathcal{P}$ with endpoints $p_i$ and $p_j$ where $p_{j-1} \in \{p_i, p_k\}$.
$\therefore$ the minimal bitomic path has an optimal substructure.

We now have two cases. From the previous claim, $p_{j-1} \in \{p_i, p_k\}$
case 1: if $j-1 \neq i$, then $j-1 = k$
case 2: if $j-1 = i$, then $1 \leq k < i$

**Claim 3**: Let $\ell(i,j)$ be the length of the bitonic path with endpoints $p_i$ and $p_j$ where $i < j$ and let $k$ be a neighbor of $p_j$. Then the following is true:

$$\ell(i,j) = \begin{cases} ||\overline{p_1 p_2}|| & i = 1, j = 2 \\ \ell(i, j-1) + ||\overline{p_{j-1} p_j}|| & i < j-1 \\ \min\limits_{1 \leq k < i} \{\ell(k,i) + ||\overline{p_k p_j}||\} & i = j-1 \end{cases}$$

Proof:

- The minimal bitonic path with endpoints $p_1$ and $p_2$ only visits two points, necessarily $p_1$ and $p_2$. Thus, for the path to be simple, it must consist of the only edge between $p_1$ and $p_2$.
- From case 1, we have the minimal bitonic path $p_i \rightsquigarrow p_{k=j-1} \to p_j$. This implies that $\ell(i,j)$ must be equal to the length of the minimal bitonic path from $p_i$ to $p_{j-1}$, $\ell(i, j-1)$, plus the length of the edge $\overline{p_{j-1} p_j}$.
- From case 2, since $1 \leq k < i$, we have the minimal bitonic path $p_{i=j-1} \rightsquigarrow p_k \to p_j$. This means $\ell(i,j)$ must be equal to the length of the minimal bitonic path from $p_i$ to $p_k$, $\ell(i,k)$, plus the length of the edge $\overline{p_k p_j}$. However, $1 \leq k < i$. Necessarily, $\ell(i,j) = \ell(k,i) + ||\overline{p_k p_j}||$ for the value $k$ that yields the shortest value. Thus, the optimal bitonic path from $p_i$ to $p_j$ is the minimum of the lengths for $1 \leq k < i$.

To compute $\ell(n-1, n)$, we must compute the length of the optimal bitonic subpaths. This requires a dynamic programming bottom up approach using the formula in claim three, where we begin with $\ell(1,2), \ell(1,3), \ldots, \ell(n-1,n)$.

**Claim 4**: The dynamic programming algorithm for computing the shortest bitonic tour is an $O(n^2)$ time algorithm.
Proof: The following values will be computed:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $\ell(1,2)$ | | | | | | | |
| $\ell(1,3)$ | $\ell(2,3)$ | | | | | | |
| $\ell(1,4)$ | $\ell(2,4)$ | $\ell(3,4)$ | | | | | |
| $\ell(1,5)$ | $\ell(2,5)$ | $\ell(3,5)$ | $\ell(4,5)$ | | | | |
| $\ell(1,6)$ | $\ell(2,6)$ | $\ell(3,6)$ | $\ell(4,6)$ | $\ell(5,7)$ | | | |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | | | |
| $\ell(1,n-1)$ | $\ell(2,n-1)$ | $\ell(3,n-1)$ | $\ell(4,n-1)$ | $\ell(5,n-1)$ | $\dots$ | $\ell(n-2,n-1)$ | |
| $\ell(1,n)$ | $\ell(2,n)$ | $\ell(3,n)$ | $\ell(4,n)$ | $\ell(5,n)$ | $\dots$ | $\ell(n-2,n)$ | $\ell(n-1,n)$ |
| $n-1$ | $n-2$ | $n-3$ | $n-4$ | $n-5$ | $\dots$ | 2 | 1 |

Assume that all operations on real numbers take unit time. Since all calculations are done using lengths of bitonic subpaths, which only requires a simple table lookup, $\ell(i,j) = 1$. The running time is equivalent to

$n-1+n-2+n-3+n-4+n-5+\ldots+2+1 = \sum_{1}^{n-1} i = \frac{1}{2}n\cdot(1+n-1) = \frac{n(n-1)}{2}$

$\implies O(n^2)$

The set of points can be sorted in $O(n\ log\ n)$. The total running time is then $O(n^2) + O(n\ log\ n) = O(n^2)$ $\square$

(c) Printing Neatly

**15-2)** Give a dynamic programming algorithm to print a paragraph of $n$ words neatly on a printer. Analyze the runtime and space requirements for your algorithm.

**Solution**:

Assume a monospaced font (all characters having the same width). The input text is a sequence of $n$ words of length $l_1, l_2, ..., l_n$ measured in characters. We want to print this paragraph neatly on a number of lines that hold a maximum of $M$ characters each.

Printing $n$ number of words can be accomplished in a greedy manner by compressing as many words (seperated by a single space) in the first line before moving on to the next, and doing the same for the proceeding lines until the last word has been reached. This solution presents the problem of "unneatness" (long words may cause a larger number of spaces at the end of the lines, making the paragraph look ragged). Printing neatly can thus be accomplished by "redistributing the number of spaces at the end of each line as evenly as possible". To do this, we focus on printing as many words on each line while minimizing the number of white spaces left over at the end of each line. We must therefore define a cost to each line associated with the number of spaces left over at the end of the line. Let this cost be the cube of the number of white spaces left over. It does not matter how many spaces the last line has so we define the cost of the last line as zero. To print neatly, we must find the optimal (minimum) sum of costs of all lines.

- Let $s_i = \sum_{k=1}^{i} l_k$ for $i \leq j$, then $s_j - s_i = \sum_{k=i}^{j} l_k$. We can compute $s_1, s_2, ..., s_n$ in $O(n)$. The number of extra white spaces left at the end of the line containing words $i$ through $j$ is $M - (j-i) - \sum_{k=i}^{j} l_k = M - j + i - s_j + s_i$. Thus, the cost incurred by a line containing words $i$ through $j$ is given by:

$$lineCost(i,j) = \begin{cases} 0 & j = n, \ and \ 0 \leq M - j + i - s_j + s_i \\ (M - j + i - s_j + s_i)^3 & 0 \leq M - j + i - s_j + s_i \\ \infty & otherwise \end{cases}$$

This can be computed in constant time, $O(1)$, since $s_i$ and $s_j$ have already been computed.

- Let $C(n)$ be the optimal cost of printing words 1 through $n$ neatly. If all the words fit on one line, the cost $C(n) = lineCost(1, n) = 0$. Otherwise, let the last line contain words $i$ through $n$. Then the cost of printing $n$ words is equal to the cost of printing the first $i-1$ words plus cost of printing the words on the last line. Each line holds a maximum of $M$ characters and each word of length $l$ is separated by a space. Thus, any line can hold a maximum of $\frac{M}{2}$ words. The last line can thus hold up to words $i = j - \frac{M}{2}$ to $j$. We can then summarize the cost to print the first $j$ words as:

$$C(j) = \begin{cases} \min\limits_{1 \leq j - \frac{M}{2} \leq i \leq j} C(i-1) + lineCost(i,j) \end{cases}$$

- We can compute $C(n)$ using a "bottom up" approach by computting $C(1)$, $C(2)$, $C(3)$, ... , $C(n)$ and saving the $i$ for which $C(j)$ is minimum to $L_j$. That is $L_j$ is the index of the first word of the line that ends with letter $j$. Since a line contains a maximum of $\frac{M}{2}$ words and we must compute *lineCost* for any number of words that fit on the last line, it takes $O(\frac{M}{2})$ to compute $C(j)$, $j \in \{1, 2, .., n\}$. Since we must compute for all $j \in \{1, 2, ..., n\}$, the running time to compute $C(n)$ is thus $O(\frac{M}{2}n) + O(n) + O(1) = O(Mn)$.

- We need space for $s_1, s_2, ..., s_n$ , $L_1, L_2, .., L_n$ and $C_1, C_2, ..., C_n \implies$ space required is $O(3n) = O(n)$.

- PRINT-NEATLY(l, n, M)
  1. $C(0) = 0$, $s_i = 0$
  2. for $i = 1, ..., n$
     $s_i = s_i + l_i$
  3. for j=1,...,n
     $C(j) = \min\limits_{1 \leq j - \frac{M}{2} \leq i \leq j} \{C(i-1) + lineCost(i,j)$
     ($k = i$ for which $C(j)$ is minimum)
     $L_j = k$
  4. return L

- $L_j$ contains the index of the first word of the line that ends with the $j^{th}$ word. We can transverse backwards to neatly print words 1 through $n$ starting with $L_n$.

## References

[1] Cormen, Thomas. H., Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. *Introduction to Algorithms, Third Edition.* MIT Press, Cambridge, MA, 2009.

[2] http://www.cs.ust.hk/∼dekai/271/notes/L12/L12.pdf

[3] http://www.cs.huji.ac.il/course/2004/algo/Solutions/bitonic.pdf

[4] http://www.csee.umbc.edu/ kalpakis/Courses/441-sp03/hws/hw5-sol.pdf