**Median explain:** the basic idea is to find the median for each array if the median of the first median bigger than the second the first array will be taken from the most left index to the middle index, and the second array from the middle index to the most right index. Otherwise, if the second median is bigger the first array will be taken from the middle to the most right index, and the second array from the left to the middle index.

If both have the same median, then one of those medians will be returned

At the end we will take the maximum index if the array length has reached two and the minimum from the second index add them, then divide by tow and return the median of the tow array.

**Source code:**

public class TextFormating{

int MAX = Integer.MAX\_VALUE;

//To print the solution

int print(int arr[], int n){

int s;

if(arr[n] == 1)

s=1;

else

s=print(arr, arr[n]-1) + 1;

System.out.println("Line number" + " " + s + " : " + "From word number" +" "+ arr[n] + " " + "to" + " " + n);

return s;}

// l[] represents lengths of different words in input sequence. For example, int l[] = {6,3,6,3,8,2,10}; represents the given sentence "CSC311 The Design and Analysis of Algorithms".

// n is size of l[], and M is line width: (maximum no. of characters that can fit in a line)

void solution(int l[], int n, int M){

int extra[][] = new int[n+1][n+1]; // extra[i][j] will have number of extra spaces if words from i to j are put in a single line

int cost[][] = new int[n+1][n+1]; // cost[i][j] will have cost of a line which has words from i to j

int total[] = new int[n+1]; // total[i] will have total cost of optimal arrangement of words from 1 to i

int p[] = new int[n+1]; // p[] is used to print the solution.

//calculate extra spaces in a single line.

for (int i = 1; i <= n; i++){

extra[i][i]= M - l[i-1];

for (int j = i+1; j <= n; j++)

extra[i][j] = extra[i][j-1] - l[j-1] - 1;}

for (int i = 1; i <= n; i++){

for (int j = i; j <= n; j++){

if (extra[i][j] < 0)

cost[i][j] = MAX;

else if (j == n && extra[i][j] >= 0)

cost[i][j] = 0;

else

cost[i][j] = extra[i][j]\*extra[i][j];}

}

// Calculate minimum cost and find minimum cost arrangement.

total[0] = 0;

for (int j = 1; j <= n; j++){

total[j] = MAX;

for (int i = 1; i <= j; i++){

if (total[i-1] != MAX && cost[i][j] != MAX && (total[i-1] + cost[i][j] < total[j])){

total[j] = total[i-1] + cost[i][j];

p[j] = i;}}}

print(p, n);}

public static void main(String args[]){

TextFormating t = new TextFormating();

int l[] = {6, 3, 6,3,8,2,10};

int n = l.length;

int M = 13;

t.solution (l, n, M);}}

**Time and space complexity:**

Time complexity: O(n^2)

space complexity: O(n^2)

**Sample run:**

Table

Description automatically generated

**Challenges:**

We believe that we faced a little problem in the beginning, on how to decide to implement the code. and then using an analysis skill we figure out how to do it in the right way, we also retuned to several books that describe the algorithm which was hard to understand at first.

Method explain:

To solve the problem we need to calculate

Extra space by the provided formula we can calculate the extra space for each line

Note: the extra space can be a negative number.

contains words i through j, where i ≤ j

Then we compute costs of all possible lines in a 2D table cost[][], The value lc[i][j] indicates the cost to put words from i to j in a single line where i and j are indexes of words in the input sequences.

|  |  |  |
| --- | --- | --- |
| cost[i,j]=∞ | If extra[I,j] < 0 | Word dose not fit |
| cost[i,j] = 0 | If j=n and extra[i,j] ≥ 0 | Last line cost 0 |
| cost[i,j] = |  |  |

We need to try all the different possibilities after calculating the cost to find the minimum cost from 1 to j thus we can define total where total is an optimized solution:

|  |  |
| --- | --- |
| Total[ 0 ] = 0 | If j = 0 |
| Total[ j ] = Minimum (total[i-1]+cost[i , j]) where 1<I ≤ j | If j > 0 |

The method printSolution() uses p[] to print the solution, its keep track of what words go on what line, so we can keep a parallel p array that points to where each total value came from. The last line starts at word p[n] and goes through word n. The previous line starts at word p[p[n]] and goes through word p[n] – 1, etc.

Psudocode:

//calculate extra spaces in a single line.

**for** i = 1 to n;   
 extra[ i , i ]= **M** – l [i-1]  
  
 **for** j = i+1 to n  
 extra[ I , j ] = extra[ i , j-1] - l[ j-1] - 1  
  
// Calculate cost  
 **for** i = 1 to n   
  
 **for** j = i to n  
 **if** extra[ i , j ] < 0  
 cost[ i , j ] = ∞  
 **else if** j == n && extra[i][j] >= 0  
 cost[ i , j ] = 0  
 **else**  
 cost[ i , j ] =   
   
// Calculate minimum cost and find minimum cost arrangement.

total[0] = 0;  
 **for** j = 1 to n

total[j] = ∞  
 **for** i = 1 to j

**if** total[i-1] != ∞ **AND** cost[ i , j ] != ∞ **AND** (total[i-1] + cost[ i , j ] < total[j])  
 total[j] = total[i-1] + cost[ i , j ]  
 p[j] = i

**return** total and p

references:

Cormen, T., Leiserson, C., Rivest, R. and Stein, C., 2022. *Introduction to Algorithms, Fourth Edition*. Cambridge: MIT Press.

Winston, W. and Roe, A., 1997. *User's guide for LINDO and LINGO, Windows versions to accompany Operations research : applications and algorithms, third edition and Introduction to mathematical programming : applications and algorithms, second edition*. Belmont, CA [etc.]: Duxbury Press.