Carnival Editorial

Explanation:

Definitions:

- f[i][j]: the maximum score we can get from interval [i, j]
- sum[i][j]: the sum of Value[i] to Value[j]
- lf[m][n]: the maximum value for all f[i][j] + sum[i][j] where i == m and j in range [m, n]
- rf[m][n]: the maximum value for all f[i][j] + sum[i][j] where j == n and i in range [m, n]

Consider the following DP equation:

- f[i][j] = max(sum[i][m] + f[i][m]) if sum[i][m] < sum[m+1][j] for all m from i to j - 1
- f[i][j] = max(sum[m+1][j] + f[m+1][j]) if sum[i][m] > sum[m+1][j] for all m from i to j - 1
- f[i][j] = max(sum[i][m] + f[i][m], sum[m+1][j] + f[m+1][j]) if sum[i][m] < sum[m+1][j] for all m from i to j - 1

We can find the maximum m from i to j - 1 using binary search with O(log(n)) which keeps sum[i][m - 1] <= sum[m+1][j], then we get:

- For f[i][j]
    - if sum[i][m] == sum[m+1][j]: f[i][j] = max(lf[i][m], rf[m+1][j])
    - if sum[i][m] < sum[m+1][j]: f[i][j] = max(lf[i][m], rf[m+2][j])
- lf[i][j] = max(lf[i][j-1], f[i][j] + sum[i][j])
- rf[i][j] = max(rf[i+1][j], f[i][j] + sum[i][j])

  Time Complexity: O(n^2log(n))

```cpp
#include <bits/stdc++.h>


using namespace std;

typedef long long int ll;


int getsum(vector<int> &sum, int l, int r)

{

   if (l > r)

       return 0;

   if (l == 0)

       return sum[r];

   return sum[r] - sum[l - 1];

}


int main()

{

   int n;

   cin >> n;

   vector<int> value(n, 0);

   for (int i = 0; i < n; i++)

   {

       cin >> value[i];

   }

   vector<int> sum(n, 0);
```

```
    for (int i = 0; i < n; ++i)

    {

        if (i > 0)

            sum[i] += sum[i - 1];

        sum[i] += value[i];

    }

    vector<vector<int>> f = vector<vector<int>>(n, vector<int>(n,
0));

    vector<vector<int>> lf = vector<vector<int>>(n, vector<int>(n,
0));

    vector<vector<int>> rf = vector<vector<int>>(n, vector<int>(n,
0));



    for (int i = 0; i < n; ++i)

    {

        lf[i][i] = rf[i][i] = value[i];

    }



    for (int i = n - 1; i >= 0; --i)

    {

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

        {

            int segsum = getsum(sum, i, j);

            int l = i - 1, r = j;

            while (l < r - 1)
```

```
        {
            int mid = (l + r) / 2;
            int left = getsum(sum, i, mid);
            if (left * 2 <= segsum)
            {
                l = mid;
            }
            else
            {
                r = mid;
            }
        }
        if (l >= i)
            f[i][j] = max(f[i][j], lf[i][l]);
        int rst = l;
        if (getsum(sum, i, l) * 2 < segsum)
        {
            rst += 2;
        }
        else
        {
            rst += 1;
        }
        if (rst <= j)
```

```
                f[i][j] = max(f[i][j], rf[rst][j]);

            lf[i][j] = max(lf[i][max(i, j - 1)], f[i][j] + segsum);

            rf[i][j] = max(rf[max(0, i + 1)][j], f[i][j] + segsum);

        }

    }

    cout << f[0][n - 1] << "\n";

    return 0;

}
```