

```

#include <iostream>
#include <vector>
#include <limits>
#include <deque>
#include <queue>

#define PA pair<int, int>

using namespace std;

// Вариант с време O(nk) и памет O(n)
// Без възстановяване на оптималното решение.
int minCostTravel(int n, int k, const vector<int> &M)
{
    vector<int> D(n + 1, numeric_limits<int>::max());
    D[0] = 0;

    for (int i = 1; i <= n; i++)
    {
        for (int j = max(0, i - k); j < i; j++)
        {
            if (D[j] != numeric_limits<int>::max())
            {
                D[i] = min(D[i], D[j] + M[j]);
            }
        }
    }
    return D[n];
}

// Вариант с време O(nk) и памет O(n)
// с възстановяване на оптималното решение.
pair<int, vector<int>> minCostTravelPath(int n, int k, const vector<int> &M)
{
    vector<int> D(n + 1, numeric_limits<int>::max());
    vector<int> prev(n + 1, -1);
    D[0] = 0;

    for (int i = 1; i <= n; i++)
    {
        for (int j = max(0, i - k); j < i; j++)
        {
            if (D[j] != numeric_limits<int>::max() && D[j] + M[j] < D[i])
            {
                D[i] = D[j] + M[j];
                prev[i] = j;
            }
        }
    }

    vector<int> path;
    for (int i = n; i != -1; i = prev[i])
    {
        path.push_back(i);
    }

    return {D[n], path};
}

// Вариант с време O(nk) и памет O(k)
// Без възстановяване на оптималното решение.
int minCostTravelMemoryK(int n, int k, const vector<int> &M)
{
    vector<int> D(k, numeric_limits<int>::max()); // Цикличен буфер с размер k
    D[0] = 0;

    for (int i = 1; i <= n; i++)
    {
        int minPrev = numeric_limits<int>::max();

        // Търсим минимума в последните k елемента
        for (int j = max(0, i - k); j < i; j++)
        {
            minPrev = min(minPrev, D[j % k] + M[j]);
        }

        D[i % k] = minPrev; // Запазваме само последните k стойности
    }

    return D[n % k]; // Връщаме последния записан резултат
}

// Вариант с време O(nlogn) и памет O(n)
// Без възстановяване на оптималното решение.
int minCostTravelNlogN(int n, int k, const vector<int> &M)
{
    priority_queue<PA, vector<PA>, greater<PA>> pq;

    int D = 0;

    for (int i = n - 1; i >= n - k; i--)
    {
        pq.push({M[i], i});
    }

    for (int i = n - k - 1; i >= 0; i--)
    {
        while (!pq.empty() && pq.top().second - i > k)
        {
            pq.pop();
        }
        D = pq.top().first;
        pq.push({M[i] + D, i});
    }

    return D;
}

// Вариант с време O(n) и памет O(k)
// Без възстановяване на оптималното решение.
int minCostTravelDeque(int n, int k, const vector<int> &M)
{
    deque<PA> D; // двусвързан списък

    D.push_back({0, 0});
    D.push_back({M[1], 1});

    for (int i = 2; i <= n; i++)
    {
        while (i - D.front().second > k)
        {
            D.pop_front();
        }
        int tmp = M[i] + D.front().first;
        while (!D.empty() && D.back().first >= tmp)
        {
            D.pop_back(); // поддържа списъка D сортиран
        }
        D.push_back({tmp, i});
    }
    return D.front().first;
    // Сложността по време е O(n),
    // защото всеки елемент на масива най-много веднъж
    // се добавя към списъка D и се премахва от него,
    // а всяка от тези операции се изпълнява
    // за константно време.
    // Сложността по памет е O(k) заради списъка D,
    // който във всеки миг съдържа най-много k + 1 елемента
    // (където k е дадено цяло положително число).
}

int main()
{
    int n, k;
    cout << "Въведете броя на километрите (n) и максималното разстояние на ден (k): ";
    cin >> n >> k;

    vector<int> M(n, 0);
    cout << "Въведете цените на хотелите по шосето: ";
    for (int i = 1; i < n; i++)
    {
        cin >> M[i];
    }

    pair<int, vector<int>> result1 = minCostTravelPath(n, k, M);

    int res1 = minCostTravel(n, k, M);

    int res2 = minCostTravelMemoryK(n, k, M);

    int res3 = minCostTravelDeque(n, k, M);

    int res4 = minCostTravelNlogN(n, k, M);

    for (int v : result1.second)
    {
        cout << v << " ";
    }

    cout << '\n';

    cout << "result1: " << result1.first << " res1: " << res1 << " res2: " << res2 << " res3: " << res3 << " res4: " << res4 << '\n';

    return 0;
}

```