MINIMUM VARIATION

Assume that the array of marks is sorted, i.e. $s_1 \le s_2 \le \cdots \le s_n$.

The key observation is that the marks at last can be assumed to be either s_1 or s_n. This is because if s_1 and s_n are both in the prefix of length i, then clearly $d_i = s_n - s_1$, which is the maximum possible value of any discrepancy. Similarly $d_{i+1}, d_{i+2}, \ldots, d_n$ are all equal to $s_n - s_1$. This moving either s_1 or s_n (whichever appears last) to the very end of the array cannot possibly increase the sum of these discrepancies, since they already have the largest possible value.

If we repeat the previous observation, we deduce that for each i, the prefix of length i in an optimal solution forms a contiguous subarray of the sorted array. Therefore, we may solve the problem through dynamic programming: $dp(l,r)$ represents the minimum possible answer if we solve for the subarray $s[l \ldots r]$. Clearly $dp(x,x)=0$, and the transition is given by

$$dp(l,r) = s_r - s_l + \min(dp(l+1,r), dp(l,r-1))$$

Which corresponds to placing either the smallest or the largest element at the end of the sequence. The final answer is $dp(1,n)$. This allows us to solve the problem in $O(n^2)$.

Solution-

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

const int MAX = 2e3 + 5;
```

```cpp
ll mem[MAX][MAX], a[MAX];

ll dp(int l, int r) {
    if(mem[l][r] != -1)
        return mem[l][r];
    if(l == r)
        return 0;
    return mem[l][r] = a[r] - a[l] + min(dp(l + 1, r), dp(l, r - 1));
}

int main() {
    int n;
    cin >> n;

    for(int i = 0; i < n; i++)
        cin >> a[i];

    sort(a, a + n);
    memset(mem, -1, sizeof mem);

    cout << dp(0, n - 1) << '\n';
}
```