# Lab Report - 03

# Array Manipulation

1. **Problem Statement :** In this problem, you are given an array of integers, and you need to make it *fashionable*. An array is considered fashionable if the sum of its minimum and maximum elements is divisible by 2, meaning their sum is even. To achieve this, you are allowed to remove any number of elements from the array. Your task is to find the minimum number of removals required so that the remaining array satisfies this condition. You must repeat this process for multiple test cases and output the minimum removals for each.

   **Thought Process:** Since the problem has low constraints, I used a brute force approach to find the minimum count of removals needed to make the array fashionable.

   **Source Code :**

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    long long t;
    cin >> t;
    while (t--) {
        int n;
        cin >> n;
        vector<int> v(n);
        for (int i = 0; i < n; i++) cin >> v[i];
        sort(v.begin(), v.end());
        int mn = INT_MAX;
        for (int i = 0; i < n; i++) {
            for (int j = n - 1; j >= i; j--) {
                if ((v[i] + v[j]) % 2 == 0) mn = min(mn, i + (n - 1 - j));
            }
        }
        cout << mn << '\n';
    }
}
```

**Input :**

```
2
2
5 2
7
3 1 4 1 5 9 2
```

**Output :**

```
1
0
```

2. **Problem Statement :** The Lever performs iterations where it decreases elements of array $a$ that are greater than corresponding elements in $b$, and increases elements of $a$ that are less than corresponding elements in $b$. Each iteration consists of one decrease and one increase step (if possible). The process stops when no element in $a$ is greater than the corresponding element in $b$. Given arrays $a$ and $b$, determine the fixed number of iterations needed until the Lever stops. Output this number for each test case.

**Thought Process :** The iteration ends only when no element in $a$ is greater than its counterpart in $b$, so we focus on step 1. Each element $ai$ greater than $bi$ will be decreased exactly $ai-bi$ times. Therefore, the total number of step 1 executions is the sum of all positive differences $(a_i, b_i)$, and the iteration count is this sum plus one.

**Source Code :**

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

int main(void) {
    int t;
    cin >> t;

    while (t--) {
        int n;
        cin >> n;

        vector<long long> v1(n), v2(n);
        for (int i = 0; i < n; i++) cin >> v1[i];
        for (int i = 0; i < n; i++) cin >> v2[i];

        long long ans = 1;
        for (int i = 0; i < n; i++) {
             ans += max(0, v1[i] - v2[i]);
        }
        cout << ans << '\n';
    }
}
```

**Input :**

```
2
2
7 3
5 6
3
3 1 4
3 1 4
```

**Output :**

```
3
1
```

3. **Problem Statement :** You need to build an array of length $n$ where adjacent elements have opposite signs and every subarray of length two or more has a positive sum. Among all such arrays, choose the one with the smallest absolute values in lexicographical order. Output this optimal array for each test case.

   **Thought Process:** For $n = 2$, the array is always $[-1, 2]$. For larger $n$, the pattern alternates between $-1$ and $3$, except for the last element which is $2$ if needed to maintain conditions.

   **Source Code :**

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

int main() {
    long long t;
```

```
    cin >> t;
    while (t--) {
        int n;
        cin >> n;

        if (n == 2) {
            cout << "-1 2\n";
            continue;
        }

        vector<int> v(n);
        if (n % 2 == 1) {
            for (int i = 0; i < n; ++i)
                v[i] = (i % 2 == 0) ? -1 : 3;
        }
        else {
            for (int i = 0; i < n; ++i) {
                if (i < n - 2) v[i] = (i % 2 == 0) ? -1 : 3;
                else if (i == n - 2) v[i] = -1;
                else v[i] = 2;
            }
        }

        for (int x : v) cout << x << ' ';
        cout << '\n';
    }
}
```

**Input :**

```
2
2
3
```

**Output :**

```
-1 2
-1 3 -1
```

# Prefix Sum

1. **Problem Statement :** Given an array of $n$ integers, your task is to process $q$ queries of the form: what is the sum of values in range $[a, b]$?

   **Thought Process :** Solve using prefix sum.

   **Source Code :**

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

int main(void) {
    int n, q;
    cin >> n >> q;

    vector<long long> v(n + 1, 0);
    for (int i = 1; i <= n; i++) cin >> v[i];

    for (int i = 1; i <= n; i++) v[i] += v[i - 1];
```

```
    while (q--) {
        int l, r;
        cin >> l >> r;
        cout << v[r] - v[l - 1] << '\n';
    }
}
```

**Input :**

```
8 4
3 2 4 5 1 1 5 3
2 4
5 6
1 8
3 3
```

**Output :**

```
11
2
24
4
```

2. **Problem Statement :** Given string $s$, and you need to answer $q$ queries. each query need to print occurrences of $'a'$ from index $L$ to index $R$.

   **Thought Process :** Solved using prefix sum,

   **Source Code :**

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

int main(void) {
    string s;
    cin >> s;
    int n = (int)s.length();
    vector<int> prefix(n + 1, 0);
    for (int i = 0; i < n; i++) {
        prefix[i + 1] = prefix[i] + (s[i] == 'a' ? 1 : 0);
    }
    int q;
    cin >> q;
    while (q--) {
        int l, r;
        cin >> l >> r;
        cout << prefix[r] - prefix[l - 1] << '\n';
    }
}
```

**Input :**

```
abcaabc
6
1 3
2 4
1 6
3 5
```

```
4 7
1 7
```

**Output :**

```
1
1
3
2
2
3
```

3. **Problem Statement :** Polycarpus needs to find **k** consecutive fence planks with the smallest total height to carry his piano easily. Given the heights of **n** planks, the goal is to find the starting index of such a segment with minimal sum. The fence is linear (not cyclic), so only consecutive segments within the array count. Output the starting position of that minimal-sum segment.

   **Thought Process :** Solved using prefix sum.

   **Source Code :**

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(void) {
    int n, k;
    cin >> n >> k;

    vector<long long> v(n + 1, 0);
    for (int i = 1; i <= n; i++) {
        cin >> v[i];
        v[i] += v[i - 1];
    }
    long long ans = LLONG_MAX;
    int ind = 1;
    for (int i = 1; i <= n - k + 1; i++) {
        long long curr = v[i + k - 1] - v[i - 1];
        if (curr < ans) {
            ans = curr;
            ind = i;
        }
    }
    cout << ind << '\n';
}
```

**Input :**

```
7 3
1 2 6 1 1 7 1
```

**Output :**

```
3
```

# Merge Sort

Source Code :

```cpp
#include <bits/stdc++.h>
using namespace std;

vector<int> v;

void merge(int l, int mid, int r) {
    int size_l = mid - l + 1;
    int size_r = r - mid;
    vector<int> v_l(size_l), v_r(size_r);

    for (int i = 0; i < size_l; i++) v_l[i] = v[i + l];
    for (int i = 0; i < size_r; i++) v_r[i] = v[i + mid + 1];

    int index_l = 0, index_r = 0, index_m = l;
    while (index_l < size_l && index_r < size_r) {
        if (v_l[index_l] <= v_r[index_r]) {
            v[index_m] = v_l[index_l];
            index_l++;
        }
        else {
            v[index_m] = v_r[index_r];
            index_r++;
        }
        index_m++;
    }
    while (index_l < size_l) {
        v[index_m] = v_l[index_l];
        index_l++;
        index_m++;
    }
    while (index_r < size_r) {
        v[index_m] = v_r[index_r];
        index_r++;
        index_m++;
    }
}

void merge_sort(int l, int r) {
    if (l == r) return;
    int mid = (l + r) / 2;
    merge_sort(l, mid);
    merge_sort(mid + 1, r);
    merge(l, mid, r);
}

int main(void) {
    int n;
    cin >> n;
    v.resize(n);

    for (int i = 0; i < n; i++) cin >> v[i];
    merge_sort(0, n - 1);

    for (int i = 0; i < n; i++) {
        cout << v[i] << ((i == n - 1) ? '\n' : ' ');
    }

    return 0;
}
```

**Input :**

```
6
10 2 12 3 32 4
```

**Output :**

```
2 3 4 10 12 32
```

# Insertion Sort

**Source Code :**

```cpp
#include <bits/stdc++.h>
using namespace std;

void insertion_sort(vector<int>& v) {
    int n = v.size();
    for (int i = 1; i < n; i++) {
        int item = v[i];
        int j = i - 1;
        while (j >= 0 && v[j] > item) {
            v[j + 1] = v[j];
            j--;
        }
        v[j + 1] = item;
    }
}
int main() {
    int n;
    cin >> n;

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

    insertion_sort(v);

    for (int i = 0; i < n; i++) {
        cout << v[i] << (i == n - 1 ? '\n' : ' ');
    }

    return 0;
}
```

**Input :**

```
6
10 2 12 3 32 4
```

**Output :**

```
2 3 4 10 12 32
```

# Selection Sort

**Source Code :**

```cpp
#include <bits/stdc++.h>
using namespace std;

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

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

    for (int i = 0; i < n - 1; i++) {
        int min_index = i;
        for (int j = i + 1; j < n; j++) {
            if (v[j] < v[min_index]) min_index = j;
        }
        if (min_index != i) swap(v[i], v[min_index]);
    }

    for (int i = 0; i < n; i++) {
        cout << v[i] << (i == n - 1 ? '\n' : ' ');
    }

    return 0;
}
```

**Input :**

```
6
10 2 12 3 32 4
```

**Output :**

```
2 3 4 10 12 32
```

# Bubble Sort

**Source Code :**

```cpp
#include <bits/stdc++.h>
using namespace std;

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

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

    int cnt = 1;
    while (cnt < n) {
        for (int i = 0; i < n - cnt; i++) {
            if (v[i] > v[i + 1]) swap(v[i], v[i + 1]);
        }
        cnt++;
    }

    for (int i = 0; i < n; i++) cout << v[i] << ' ';
    cout << '\n';
```

```
    return 0;
}
```

**Input :**

```
6
10 2 12 3 32 4
```

**Output :**

```
2 3 4 10 12 32
```