5.Array prefix Sum and related problems

1. Equilibrium index of an array

```
int equilibrium(int arr[], int n)
{
    int i, j;
    int leftsum, rightsum;

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

        leftsum = 0;
        for (j = 0; j < i; j++)
            leftsum += arr[j];

        rightsum = 0;
        for (j = i + 1; j < n; j++)
            rightsum += arr[j];

        if (leftsum == rightsum)
            return i;
    }

    /* return -1 if no equilibrium index is found */
    return -1;
}

// Driver code
int main()
{
    int n;
    scanf("%d",&n);
    int arr[n];
    for(int i=0;i<n;i++)
    {
```

```
        scanf("%d",&arr[i]);
    }
    int arr_size = sizeof(arr) / sizeof(arr[0]);
    printf("%d", equilibrium(arr, arr_size));

    getchar();
    return 0;
}
```

2.Special index:

```
#include <stdio.h>

int countSpecialIndices(int arr[], int n) {
    int totalSpecialIndices = 0;

    for (int i = 1; i < n - 1; i++) {
        int leftSum = 0;
        for (int j = 0; j < i; j++) {
            leftSum += arr[j];
        }

        int rightSum = 0;
        for (int j = i + 1; j < n; j++) {
            rightSum += arr[j];
        }

        if (leftSum == rightSum) {
            totalSpecialIndices++;
        }
    }

    return totalSpecialIndices;
}

int main() {
```

```
    int n;

    printf("Enter the number of elements in the array: ");
    scanf("%d", &n);

    int arr[n];
    printf("Enter the elements of the array:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    int result = countSpecialIndices(arr, n);
    printf("Total number of special indices: %d\n", result);

    return 0;
}
```

3.Pick from both sides:

```
#include <stdio.h>


int findSumPair(int arr[], int n, int target) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (arr[i] + arr[j] == target) {
                return arr[i] + arr[j];
            }
        }
    }
    return -1;
```

```
}

int main() {
    int n;
    printf("Enter the number of elements in the array: ");
    scanf("%d", &n);

    int arr[n];
    printf("Enter the elements of the array:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    int target;
    printf("Enter the target sum: ");
    scanf("%d", &target);

    int result = findSumPair(arr, n, target);

    if (result != -1) {
        printf("Sum: %d\n", result);
    } else {
        printf("No two numbers sum up to %d\n", target);
    }
```

```
    return 0;
}
```

4. Range Sum Query

```
#include <stdio.h>
#include <stdlib.h>

typedef struct SegmentTree {
    int *tree;
    int n;
} SegmentTree;

SegmentTree* createSegmentTree(int nums[], int n) {
    SegmentTree *tree = (SegmentTree*)malloc(sizeof(SegmentTree));
    tree->n = n;
    tree->tree = (int*)malloc(sizeof(int) * 4 * n);
    buildTree(tree, nums, 0, 0, n - 1);
    return tree;
}

void buildTree(SegmentTree *tree, int nums[], int node, int start, int
end) {
```

```
    if (start == end) {
        tree->tree[node] = nums[start];
        return;
    }

    int mid = (start + end) / 2;
    buildTree(tree, nums, 2 * node + 1, start, mid);
    buildTree(tree, nums, 2 * node + 2, mid + 1, end);
    tree->tree[node] = tree->tree[2 * node + 1] + tree->tree[2 * node + 2];
}

void update(SegmentTree *tree, int node, int start, int end, int index, int val) {
    if (start == end) {
        tree->tree[node] = val;
        return;
    }

    int mid = (start + end) / 2;
    if (index <= mid) {
        update(tree, 2 * node + 1, start, mid, index, val);
    } else {
        update(tree, 2 * node + 2, mid + 1, end, index, val);
    }
```

```
    tree->tree[node] = tree->tree[2 * node + 1] + tree->tree[2 * node +
2];
}

int query(SegmentTree *tree, int node, int start, int end, int left, int
right) {
    if (start > right || end < left) {
        return 0;
    }
    if (left <= start && end <= right) {
        return tree->tree[node];
    }

    int mid = (start + end) / 2;
    int leftSum = query(tree, 2 * node + 1, start, mid, left, right);
    int rightSum = query(tree, 2 * node + 2, mid + 1, end, left, right);
    return leftSum + rightSum;
}

int main() {
    int n;
    printf("Enter the number of elements in the array: ");
    scanf("%d", &n);

    int *nums = (int*)malloc(sizeof(int) * n);
```

```
    printf("Enter the elements of the array:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &nums[i]);
    }

    SegmentTree *tree = createSegmentTree(nums, n);

    int updateIndex, updateValue;
    printf("Enter the index to update and the new value: ");
    scanf("%d %d", &updateIndex, &updateValue);
    update(tree, 0, 0, n - 1, updateIndex, updateValue);

    int left, right;
    printf("Enter the range for the sum query (left right): ");
    scanf("%d %d", &left, &right);
    int result = query(tree, 0, 0, n - 1, left, right);

    printf("Sum of elements between indices %d and %d after updating
index %d: %d\n", left, right, updateIndex, result);

    free(nums);
    free(tree->tree);
    free(tree);

    return 0;
```

```
}
```

5. Product array puzzle

```
#include <stdio.h>
#include <stdlib.h>

int* productExceptSelf(int* nums, int numsSize, int* returnSize) {
    int* product = (int*)malloc(sizeof(int) * numsSize);
    int* left = (int*)malloc(sizeof(int) * numsSize);
    int* right = (int*)malloc(sizeof(int) * numsSize);

    left[0] = 1;
    for (int i = 1; i < numsSize; i++) {
        left[i] = left[i - 1] * nums[i - 1];
    }

    right[numsSize - 1] = 1;
    for (int i = numsSize - 2; i >= 0; i--) {
        right[i] = right[i + 1] * nums[i + 1];
    }

    for (int i = 0; i < numsSize; i++) {
        product[i] = left[i] * right[i];
    }
```

```
    free(left);
    free(right);
    *returnSize = numsSize;
    return product;
}

int main() {
    int numsSize;

    printf("Enter the number of elements in the array: ");
    scanf("%d", &numsSize);

    int* nums = (int*)malloc(sizeof(int) * numsSize);
    printf("Enter the elements of the array:\n");
    for (int i = 0; i < numsSize; i++) {
        scanf("%d", &nums[i]);
    }

    int resultSize;
    int* result = productExceptSelf(nums, numsSize, &resultSize);

    printf("Product of all elements except self:\n");
    for (int i = 0; i < resultSize; i++) {
        printf("%d ", result[i]);
    }
```

```
    printf("\n");

    free(nums);
    free(result);

    return 0;
}
```