SSTF

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

using namespace std;

// Function to find the index of the closest request to the current head
int findClosest(int currentHead, vector<int>& requests, vector<bool>& visited) {
    int minDistance = INT_MAX;
    int index = -1;
    for (int i = 0; i < requests.size(); i++) {
        if (!visited[i] && abs(requests[i] - currentHead) < minDistance) {
            minDistance = abs(requests[i] - currentHead);
            index = i;
        }
    }
    return index;
}

int main() {
    int n, head;
    cout << "Enter number of requests: ";
    cin >> n;

    vector<int> requests(n);
    cout << "Enter the requests: ";
    for (int i = 0; i < n; i++) {
        cin >> requests[i];
    }

    cout << "Enter initial head position: ";
    cin >> head;

    vector<bool> visited(n, false);
    vector<int> seekSequence;
    int totalSeekTime = 0;
    int currentHead = head;

    for (int i = 0; i < n; i++) {
        int idx = findClosest(currentHead, requests, visited);
        visited[idx] = true;
        seekSequence.push_back(requests[idx]);
        totalSeekTime += abs(requests[idx] - currentHead);
        currentHead = requests[idx];
    }

    cout << "\nSeek sequence: ";
```

```
    for (int i : seekSequence) {
        cout << i << " ";
    }
    cout << "\nTotal seek time: " << totalSeekTime << endl;

    return 0;
}
```

C-Look (towards large)

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

int main() {
    int n, head;
    cout << "Enter number of requests: ";
    cin >> n;

    vector<int> req(n);
    cout << "Enter requests: ";
    for (int &x : req) cin >> x;

    cout << "Enter head position: ";
    cin >> head;

    vector<int> left, right;

    for (int x : req) {
        if (x >= head) right.push_back(x);
        else left.push_back(x);
    }

    sort(right.begin(), right.end());
    sort(left.begin(), left.end());

    cout << "\nC-LOOK Order: ";
    int total = 0, prev = head;

    for (int x : right) {
        cout << x << " ";
        total += abs(prev - x);
        prev = x;
    }
```

```
    for (int x : left) {
        cout << x << " ";
        total += abs(prev - x);
        prev = x;
    }

    cout << "\nTotal head movement = " << total << "\n";
    return 0;
}
```

C-LOOK (towards smallest)

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

int main() {
    int n, head;
    cout << "Enter number of requests: ";
    cin >> n;

    vector<int> req(n);
    cout << "Enter requests: ";
    for (int &x : req) cin >> x;

    cout << "Enter head position: ";
    cin >> head;

    vector<int> left, right;

    for (int x : req) {
        if (x < head) left.push_back(x);
        else right.push_back(x);
    }

    // Sort both lists normally (ascending)
    sort(left.begin(), left.end());   // smallest → biggest
    sort(right.begin(), right.end()); // smallest → biggest

    cout << "\nC-LOOK Order (towards smaller first): ";
    int total = 0, prev = head;

    // Visit left side first (but in reverse = descending)
    for (int i = left.size() - 1; i >= 0; i--) {
        cout << left[i] << " ";
        total += abs(prev - left[i]);
        prev = left[i];
```

```
    }

    // Jump to the smallest request on right side
    if (!right.empty()) {
        total += abs(prev - right[0]);
        prev = right[0];
        cout << right[0] << " ";
    }

    //  Now service right side normally (ascending)
    for (int i = 1; i < right.size(); i++) {
        cout << right[i] << " ";
        total += abs(prev - right[i]);
        prev = right[i];
    }

    cout << "\nTotal head movement = " << total << "\n";
    return 0;
}
```

Best Fit

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

using namespace std;

int main() {
    int n, m; // n = memory blocks, m = processes
    cout << "Enter number of memory blocks: ";
    cin >> n;
    vector<int> blockSize(n);
    cout << "Enter the size of each block: ";
    for (int i = 0; i < n; i++) cin >> blockSize[i];

    cout << "Enter number of processes: ";
    cin >> m;
    vector<int> processSize(m);
    cout << "Enter the size of each process: ";
    for (int i = 0; i < m; i++) cin >> processSize[i];

    vector<int> allocation(m, -1); // stores block allocated for each process
    vector<int> internalFrag(n, 0); // internal fragmentation per block

    // Best Fit Allocation
    for (int i = 0; i < m; i++) {
        int bestIdx = -1;
```

```
        for (int j = 0; j < n; j++) {
            if (blockSize[j] >= processSize[i]) {
                if (bestIdx == -1 || blockSize[j] < blockSize[bestIdx]) {
                    bestIdx = j;
                }
            }
        }
        if (bestIdx != -1) {
            allocation[i] = bestIdx;
            internalFrag[bestIdx] = blockSize[bestIdx] - processSize[i];
            blockSize[bestIdx] -= processSize[i];
        }
    }

    // Calculate total internal fragmentation
    int totalInternal = 0;
    for (int i = 0; i < n; i++) totalInternal += internalFrag[i];

    // Calculate external fragmentation
    int totalFree = 0;
    for (int i = 0; i < n; i++) totalFree += blockSize[i];

    int totalExternal = 0;
    for (int i = 0; i < m; i++) {
        if (allocation[i] == -1 && processSize[i] <= totalFree)
            totalExternal += processSize[i]; // external fragmentation
    }

    // Output
    cout << "\nProcess No.\tProcess Size\tBlock Allocated\tInternal Frag\n";
    for (int i = 0; i < m; i++) {
        cout << "  " << i+1 << "\t\t" << processSize[i] << "\t\t";
        if (allocation[i] != -1)
            cout << allocation[i]+1 << "\t\t" << internalFrag[allocation[i]] << endl;
        else
            cout << "Not Allocated\t0" << endl;
    }

    cout << "\nTotal Internal Fragmentation: " << totalInternal << endl;
    cout << "Total External Fragmentation: " << totalExternal << endl;

    return 0;
}
```

FIFO (Page Replacement)

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

int main() {
    int n, f;
    cout << "Pages: ";
    cin >> n;

    vector<int> pages(n);
    for (int &x : pages) cin >> x;

    cout << "Frames: ";
    cin >> f;

    queue<int> q;
    unordered_set<int> s;
    int faults = 0;

    for (int p : pages) {
        if (s.find(p) == s.end()) {
            if (s.size() == f) {
                int old = q.front(); q.pop();
                s.erase(old);
            }
            s.insert(p);
            q.push(p);
            faults++;
        }
    }

    cout << "Page Faults = " << faults << "\n";
    return 0;
}
```

LRU

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

int main() {
    int n, f;
    cout << "Pages: ";
    cin >> n;
```

```cpp
    vector<int> pages(n);
    for (int &x : pages) cin >> x;

    cout << "Frames: ";
    cin >> f;

    list<int> frame;
    unordered_map<int, list<int>::iterator> pos;

    int faults = 0;

    for (int p : pages) {
        if (pos.find(p) == pos.end()) {
            faults++;
            if ((int)frame.size() == f) {
                int last = frame.back();
                frame.pop_back();
                pos.erase(last);
            }
        } else {
            frame.erase(pos[p]);
        }
        frame.push_front(p);
        pos[p] = frame.begin();
    }

    cout << "Page Faults = " << faults << "\n";
    return 0;
}
```

Optimal

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

int main() {
    int n, f;
    cout << "Pages: ";
    cin >> n;

    vector<int> pages(n);
    for (int &x : pages) cin >> x;

    cout << "Frames: ";
    cin >> f;
```

```
    vector<int> frame;
    int faults = 0;

    for (int i = 0; i < n; i++) {
        int p = pages[i];

        if (find(frame.begin(), frame.end(), p) != frame.end())
            continue;

        faults++;

        if (frame.size() < f) {
            frame.push_back(p);
            continue;
        }

        int far = -1, idx = -1;

        for (int j = 0; j < f; j++) {
            int next = -1;
            for (int k = i + 1; k < n; k++) {
                if (frame[j] == pages[k]) { next = k; break; }
            }
            if (next == -1) { idx = j; break; }
            if (next > far) { far = next; idx = j; }
        }

        frame[idx] = p;
    }

    cout << "Page Faults = " << faults << "\n";
    return 0;
}
```