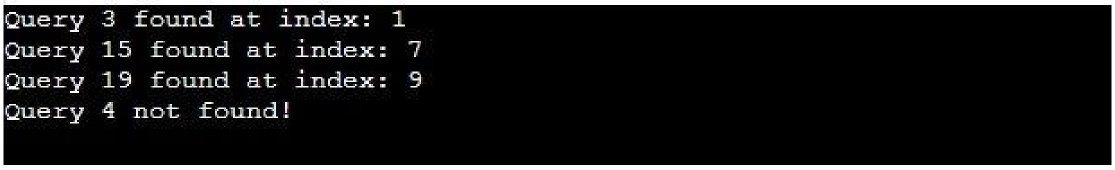
```
            cout<<"Query " << queries[i] <<" not 
found!"<<endl;
         }
     }
     return 0;
}
```

Output:



```
Query 3 found at index: 1
Query 15 found at index: 7
Query 19 found at index: 9
Query 4 not found!
```

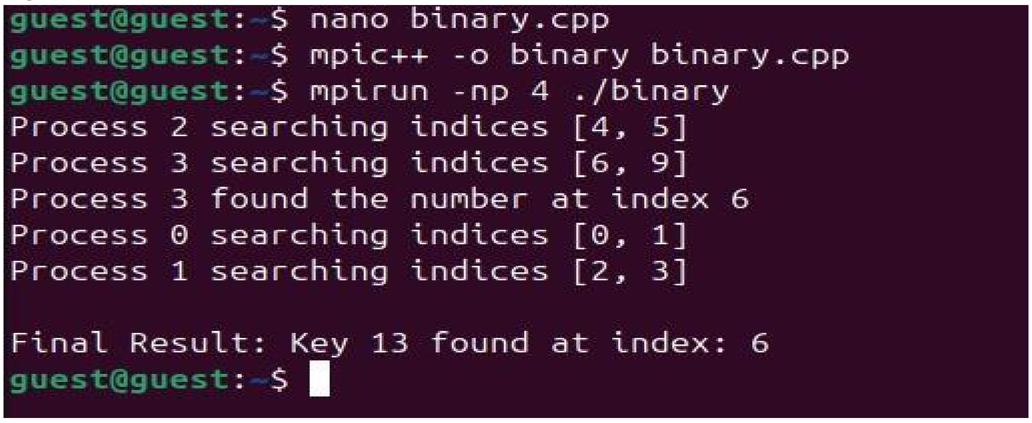
1. DISTRIBUTED IMPLEMENTATION (MPI)

Code:

```
#include <iostream>
#include <mpi.h>
using namespace std;

void mpiBinarySearch(int argc, char** argv, int arr[], int n, 
int key) {
    int rank, size;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    // Divide the array into chunks for each process
    int chunkSize = n / size;
    int start = rank * chunkSize;
    int end = (rank == size - 1) ? (n - 1) : (start + 
chunkSize - 1);
    cout << "Process " << rank << " searching indices [" << 
start << ", " << end << "]" << endl;
    int foundIndex = -1;
    int low = start;
    int high = end;
    // Perform binary search within the assigned chunk
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (arr[mid] == key) {
            foundIndex = mid;
            break;
        } else if (arr[mid] < key) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
    if (foundIndex != -1) {
        cout << "Process " << rank << " found the number at 
index " << foundIndex << endl;
    }
    // Reduce results using MPI_MAX to find the largest valid 
index (or -1 if not found)
    int globalIndex = -1;
    MPI_Reduce(&foundIndex, &globalIndex, 1, MPI_INT, MPI_MAX, 
0, MPI_COMM_WORLD);
    if (rank == 0) {
        cout << "\nFinal Result: ";
        if (globalIndex != -1)
            cout << "Key " << key << " found at index: " << 
globalIndex << endl;
        else
            cout << "Key " << key << " not found in array." << 
endl;
    }
  MPI_Finalize();
}
int main(int argc, char** argv) {
    int arr[] = {1, 3, 5, 7, 9, 11, 13, 15, 17, 19};
    int n = sizeof(arr) / sizeof(arr[0]);
    int key = 13; // You can change this as needed
    mpiBinarySearch(argc, argv, arr, n, key);
    return 0;
}
```

Output:



```
guest@guest:~$ nano binary.cpp
guest@guest:~$ mpic++ -o binary binary.cpp
guest@guest:~$ mpirun -np 4 ./binary
Process 2 searching indices [4, 5]
Process 3 searching indices [6, 9]
Process 3 found the number at index 6
Process 0 searching indices [0, 1]
Process 1 searching indices [2, 3]

Final Result: Key 13 found at index: 6
guest@guest:~$ 
```