```
* Only relevant answers are graded.
```

Q1

a) When you consider: i) recursive and ii) immediate data decompositions? Explain.

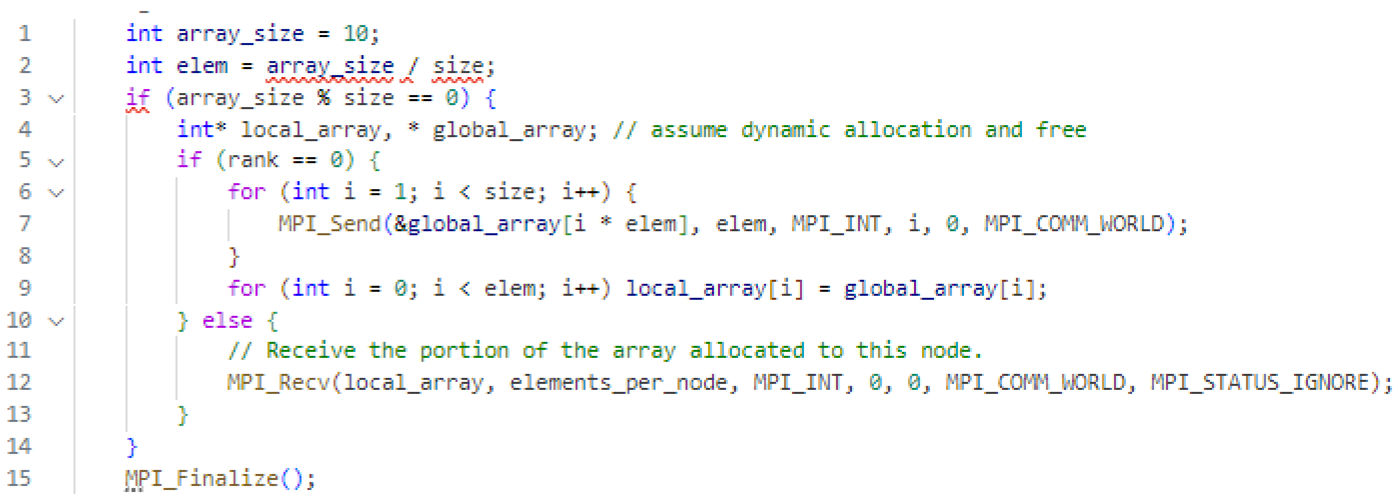
b) How distributed memory computations benefit for task interaction graph? Give an example.

c) What benefits are there in keeping two file chunks on the same rack in HDFS?

d) How Namenode ensure accurate record of data-blocks? Explain with the assumption that there are at least 2-3 node failures per minutes in a 1500 node cluster.

Q2

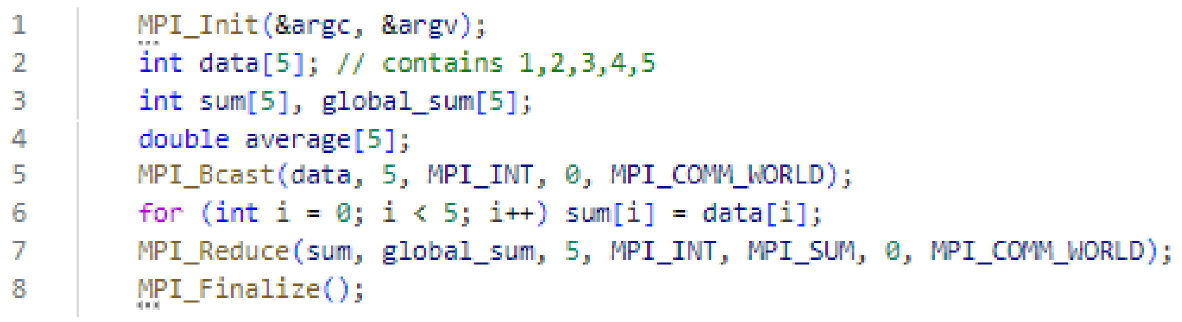
a) What this program is doing? Explain the purpose of both if statement and each MPI_???? calls.



```
int array_size = 10;
int elem = array_size / size;
if (array_size % size == 0) {
    int* local_array, * global_array; // assume dynamic allocation and free
    if (rank == 0) {
        for (int i = 1; i < size; i++) {
            MPI_Send(&global_array[i * elem], elem, MPI_INT, i, 0, MPI_COMM_WORLD);
        }
        for (int i = 0; i < elem; i++) local_array[i] = global_array[i];
    } else {
        // Receive the portion of the array allocated to this node.
        MPI_Recv(local_array, elements_per_node, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    }
}
MPI_Finalize();
```

Write MPI C code snippet where each nodes determines if its rank is odd or even. Each even node sends its 10-element array to each odd nodes. All nodes sum theirs received numbers and print local total. Do not write template code and use only MPI_Send and MPI_Recv. Hint: if (x % 2) { /* x is odd */ }

Explain what the following program is doing. Write small sentences as labels, draw different versions of "data array" as its values changed during processing, and show values of sum and global_sum after line # 7. Assume 3 processes for this MPI program.



```
1    MPI_Init(&argc, &argv);
2    int data[5]; // contains 1,2,3,4,5
3    int sum[5], global_sum[5];
4    double average[5];
5    MPI_Bcast(data, 5, MPI_INT, 0, MPI_COMM_WORLD);
6    for (int i = 0; i < 5; i++) sum[i] = data[i];
7    MPI_Reduce(sum, global_sum, 5, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
8    MPI_Finalize();
```