BFS

```
code = """
#include<iostream>
#include<omp.h>

using namespace std;
int q[100];
int visited[7];
int local_q;

void bfs(int adj_matrix[7][7], int first, int last, int q[], int n_nodes) {
    if(first==last)
      return;

    int cur_node = q[first++];
    cout<<"  "<<cur_node;

    omp_set_num_threads(3);

    #pragma omp parallel for shared(visited)
    for(int i=0; i<n_nodes; i++) {
        if(adj_matrix[cur_node][i] == 1 && visited[i] == 0){
            q[last++] = i;
            visited[i] = 1;
        }
    }

    bfs(adj_matrix, first, last, q, n_nodes);
}

int main() {
    int first = -1;
    int last = 0;
    int n_nodes = 7;

    for(int i=0; i<n_nodes; i++) {
        visited[i] = 0;
    }

    int adj_matrix[7][7] = {
      {0, 1, 1, 0, 0, 0, 0},
      {1, 0, 1, 1, 0, 0, 0},
      {1, 1, 0, 0, 1, 0, 0},
      {0, 1, 0, 0, 1, 0, 0},
      {0, 0, 1, 1, 0, 1, 0},
      {0, 0, 0, 0, 1, 0, 1},
      {0, 0, 0, 0, 0, 1, 0}
    };

    int start_node = 3;
    q[last++] = start_node;
    first++;
    visited[start_node] = 1;

    bfs(adj_matrix, first, last, q, n_nodes);

    return 0;

}
"""
```

```
file_ = open("bfs.cpp", "w")
file_.write(code)
file_.close()
```

```
!g++ -fopenmp bfs.cpp
```

```
!./a.out
```

```
3 4 1 5 2 0 6
```

BINARY SEARCH

```
code = """
#include<mpi.h>
#include<bits/stdc++.h>

using namespace std;

#define n 23
#define KEY_IDX 8

int a[] = {1,2,3,4,7,9,13,24,55,56,67,88, 100, 200, 300, 500, 760, 761, 762, 763, 764, 765};
int key = 2;
//int a[n];

int a2[n];

void generateArray() {
    for(int i = 0; i < n; i++)
        a[i] = i;

    sort(a, a + n);
    key = a[KEY_IDX];
}

int binarySearch(int *array, int start, int end, int value) {
    int mid;

    while(start <= end) {
        mid = (start + end) / 2;
        if(array[mid] == value)
            return mid;
        else if(array[mid] < value)
            start = mid + 1;
        else
            end = mid - 1;
    }
    return -1;
}

void receiveData(int id, int np, MPI_Status status){
      int length;
      // receive the count of elements, from the master
      MPI_Recv(&length, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &status);

      // receive the array elements, from the master
      MPI_Recv(&a2, length, MPI_INT, 0, 0, MPI_COMM_WORLD, &status);

      // apply binary search and find the element
      int position = binarySearch(a2, 0, length, key);

      //cout<<"Pos = "<<position<<endl;
      //cout<<"id = "<<id<<endl;

      if(position != -1) {
              int element_idx = (n / np) * id + position;
              cout<<"Element found at index : "<<element_idx<<endl;
          cout<<"Found by process: "<<id<<endl;
        }

      // return the search results to the master
      //MPI_Send(&position, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);

}


void sendData(int id, int np){
      //int length = n / np;
      //int index = id * length;

      int index = n/np * id;
      int length = (id < np-1) ? n/np : n-index;

      //if(id == (np-1))
        //  length = n - index;

      // sending 'element count' to the child process, buffer size = 1
      MPI_Send(&length, 1, MPI_INT, id, 0, MPI_COMM_WORLD);

      // sending a buffer, which contains the array elements, to the child process, buffer size = 'elements_per_process'
      MPI_Send(&a[index], length, MPI_INT, id, 0, MPI_COMM_WORLD);

}

int main(int argc, char* argv[]) {
    int pid, np, elements_per_process, n_elements_received;

    double start, end;

    MPI_Status status;

    // initialize MPI
    MPI_Init(&argc, &argv);

    // get the node id and store it in 'pid'.
    MPI_Comm_rank(MPI_COMM_WORLD, &pid);

    // get how many processes have been started
    MPI_Comm_size(MPI_COMM_WORLD, &np);

    // if master process
    if(pid == 0) {
        int index;

        // elements_per_process = n / np;
        //generateArray();

        for(int i = 0; i < n; i++)
            cout<<a[i]<<" ";
        //cout<<endl;

        cout<<key<<" "<<endl;

        start = MPI_Wtime();
        // distribute data to the child processes
        for(int i=1; i<np; i++) {
            sendData(i, np);
        }

        int position = binarySearch(a, 0, (n/np)-1, key);

        end = MPI_Wtime();

        if(position != -1) {
          cout<<"Element found at index : "<<position<<endl;
          cout<<"Found by MASTER process"<<endl;
        }

        // in seconds
        cout<<"Execution time(parallel) = "<<(end-start)<<" seconds"<<endl;
    }
    // slave processes
    else {
        receiveData(pid, np, status);
    }

    // terminate the MPI env.
    MPI_Finalize();
    return 0;
```

```
}
"""
```

```
file_ = open("BinarySearch.cpp", "w");
file_.write(code);
file_.close();
```

```
!mpiCC BinarySearch.cpp
```

```
!mpirun --allow-run-as-root -np 4 ./a.out
```

```
1 2 3 4 7 9 13 24 55 56 67 88 100 200 300 500 760 761 762 763 764 765 2 2
Element found at index : 1
Found by MASTER process
Execution time(parallel) = 3.3334e-05 seconds
```

✓ 0s completed at 22:44