PROBLEM SET 2

Zhiqiang Xie 77892769

Problem 1:

1. It's possible to maintain the isoefficiency:
   - $E = \frac{n}{p*(n/p+2\log p)} = \frac{1}{1+\frac{2*p\log p}{n}}$

   To maintain the isoefficiency, $n \in \Omega(p\log p)$.
2. It's impossible:
   - $E = \frac{n^2}{p*(n^2/p+n^3/\sqrt{p})} = \frac{1}{1+n\sqrt{p}}$

   To maintain the isoefficiency, $n \in O(\frac{1}{\sqrt{p}})$, which means the input size should decrease with the number of processors increasing.

Problem 2

1.

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

main(int argc, char** argv[]){
    int myrank, npes, buf;
    int count = 1;
    int tag = 1;
    const int root=0;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
    MPI_Comm_size(MPI_COMM_WORLD, &npes);
    if (myrank == root){
        buf = myrank;
        for (int i=0;i<myrank;i++){
            MPI_Send(&buf, count, MPI_INT, i, tag, MPI_COMM_WORLD)
        }
        for (int i=myrank+1;i<npes;i++){
            MPI_Send(&buf, count, MPI_INT, i, tag, MPI_COMM_WORLD)
        }
    }else{
        MPI_Recv(&buf, count, MPI_INT, root, tag, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        cout << "Process" << myrank << "received number" << buf << "from root\n";
    }
}
```

2.

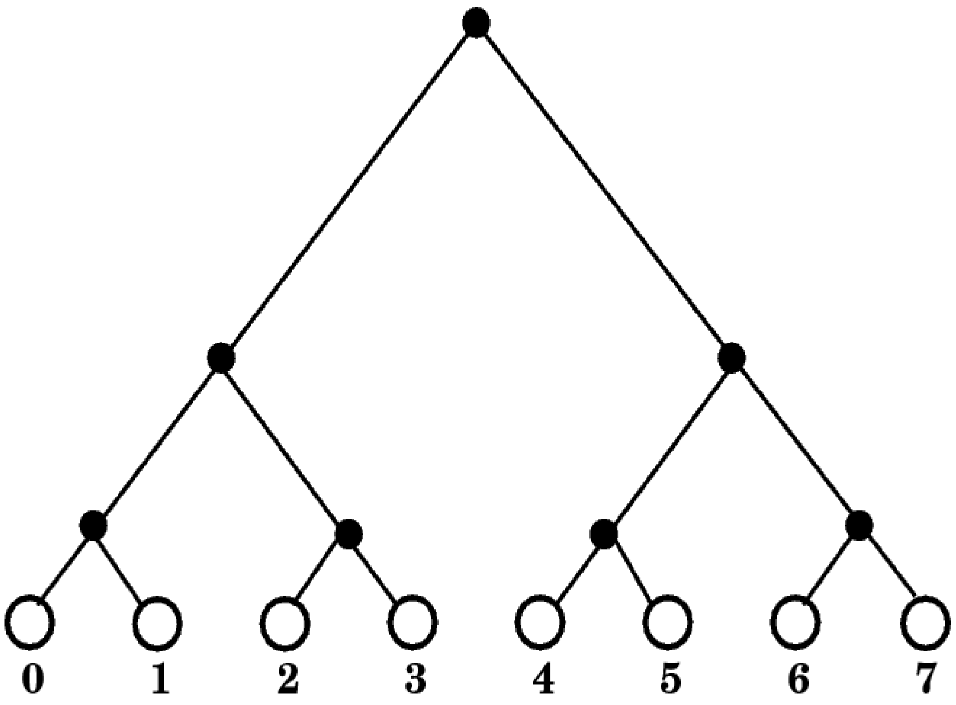
```
#include <mpi.h>
#include <iostream>
using namespace std;

main(int argc, char** argv[]){
    int myrank, npes, buf;
    int count = 1;
    int tag = 1;
    const int root=0;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
    MPI_Comm_size(MPI_COMM_WORLD, &npes);
    int numbers[npes];
    if (myrank == root){
        numbers[myrank] = myrank;
        for (int i=0;i<myrank;i++){
            MPI_Recv(&buf, count, MPI_INT, i, tag, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            numbers[i] = buf;
            cout << "receive number" << buf << "from process" << i << "\n";
        }
        for (int i=myrank+1;i<npes;i++){
            MPI_Recv(&buf, count, MPI_INT, i, tag, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            numbers[i] = buf;
            cout << "receive number" << buf << "from process" << i << "\n";
        }
    }else{
        buf = myrank;
        MPI_Send(&buf, count, MPI_INT, root, tag, MPI_COMM_WORLD)
    }
}
```

Problem 3

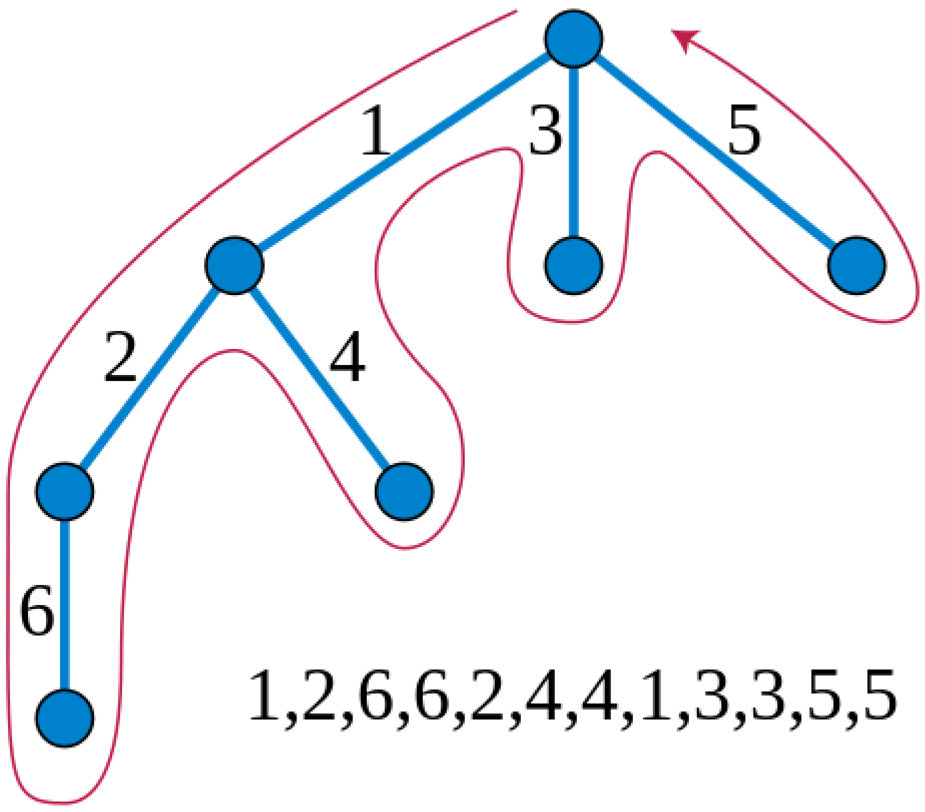
A top-down-like method can achieve the time complexity, I'll illustrate it by the binary tree in the diagram.



- Initially, every nodes hold a $m-$word message.
- Firstly, copy the message of every two nodes in different subtrees (like (0,7), (1,6), etc) and transfer it to each other in the bidirectional channel.
    - All communication take use of the path through root, which make the workload to be $m * p/2$
    - Thus the cost $t_1 = t_s + t_w m * p/2$
- Then run the similar procedures in the two subtree recursively.
    - For example, now the pairs are (0,3), (1,2), (4,7), (5,6). Here the workload of every channels is $2m * p/4 = m * p/2$
    - Thus the cost $t_i = t_s + t_w m * p/2$
- Finally, we finish the all to all broadcasting in $(t_s + t_w mp/2) \log p$ time cost.

Problem 4

The time complexity actually indicates a ring, and here's a technique called Euler-tour representation which can traverse any tree to be a ring representation.



Here the representation meets our need: the bidirectional channels. With applying the ring-broadcasting method, we'll make the workload of every single directional channel to be m.

Therefore, the time cost should be $(t_s + t_w m)(p-1)$