```
int partition (List<Int> arr, int start, int end) {
    int pivot = arr.get(end), i = start - 1;
    for (int j = start; j < end; j++) {
        if (arr.get(j) < pivot) {
            i++;
            Collections.swap(arr, i, j);
        }
    }
    Collections.swap(arr, i+1, end);
    return i+1;
}
```

QS

```
boolean isValidColor(int node, int color, boolean[][] matrix,
                     List<Int> currentColors) {
    for(int i=0; i<currentColors.size(); i++)
        if(matrix[node][i] && currColors.get(i) == color)
            return false;
    return true;
}
```

K-COLOR

```
List<Int> primes(int id, int nrProcs, int N, List<Int> primeSqrt) {
    List<Int> res = new ArrayList<>();
    int lastPrime = primeSqrt.get(primeSqrt.size()-1);
    int intervalSize = (N - lastPrime) / nrProcs, left = (N - lastPrime) % nrProcs;
    int offset = left, toAdd = 0;
    if (id < left) { offset = id; toAdd = 1; }
    int begin = (lastPrime+1) + id * intervalSize + offset;
    int end = (lastPrime+1) + (id+1) * intervalSize + offset + toAdd;
    for(int i=begin; i<end; i++) {
        int k=0;
        for(; k<primeSqrt.size() && i%primeSqrt.get(k)!=0; k++);
        if(k==primeSqrt.size() && i!=1) res.add(i);
    }
    return result;
}
```

PRIMES

```
void killAll(int nrProcs) {
    for(int i=1; i<nrProcs; i++) MPI.COMM_WORLD.Send(new int[]{0}, 0, 1, MPI.INT, i, 0);
}
void master(int n, int k, int nrProcs) {
    sout(back(0, n, k, nrProcs, new ArrayList<List<Int>>()));
    killAll(nrProcs);
}
int back(int id, int n, int k, int nrProcs, List<List<Int>> sol) {
    if(sol.size() == k) return 1;
    int child = id + nrProcs/2, sum = 0;
    if(nrProcs >= 2 && child < nrProcs) {
        List<Int> toSend = new ArrayList<>(sol);
        MPI.COMM_WORLD.Send(new int[]{1}, 0, 1, MPI.INT, child, 0);
        MPI.COMM_WORLD.Send(new Object[]{toSend}, 0, 1, MPI.OBJECT, child, 0);
        int last=0;
        if(!sol.isEmpty()) {
            last = sol.get(sol.size()-1);
            if(last%2==1) last++;
        }
        List<Int> temp = new ArrayList<>(sol);
        for(int i=last; i<n; i+=2)
            if(!temp.contains(i)) {
                temp.add(i);
                sum += back(id, n, k, nrProcs/2, temp);
                temp.remove(temp.size()-1);
            }
        int childSum = new int[1];
        MPI.COMMWORLD.Recv(childSum, 0, 1, MPI.INT, child, 0); sum += childSum[0];
    }
```

COMBINATIONS

```
else {
    int last=0;
    if(!sol.isEmpty()) last=sol.get(sol.size()-1);
    for(int i=last; i<n; i++)
        if(!sol.contains(i)) {
            sol.add(i); sum+=back(id,n,K,1,sol);
        { sol.remove(sol.size()-1);
    {
return sum;
void worker(int id, int n, int K, int nrProcs) {
    while(true) {
        int[] alive = new int[1];
        MPI.COMM_WORLD.Recv(alive,0,1,MPI.INT,MPI.ANY_SOURCE,0);
        if(alive[0]==0) return;
        Object[] received = new Object[1];
        Status status = MPI...Recv(received,0,1,MPI.OBJECT,ANYS,0);
        List<Int> sol = (List<Int>) received[0];
        int last=1, sum=0;
        if(!sol.isEmpty()) {
            last = sol.get(sol.size()-1);
        { if(last%2==0) last++;
        List<Int> temp = new ArrayList<>(sol);
        for(int i=last; i<n; i+=2)
            if(!temp.contains(i)) {
                temp.add(i); sum+=back(id,n,K,nrProcs,temp);
            { temp.remove(temp.size()-1);
        MPI...send(new int[]{sum},0,1,MPI.INT,status.source,0);
{ {
void back(int n, int T, List<Int> sol) {
    if(sol.size()==n) return 1; sum.addAndGet(1);
    if(T==1) {
        for(int i=0; i<n; i++) {
            if(!sol.contains(i)) {
                sol.add(i); back(n,1,sol);
                sol.remove(sol.size()-1);
    { { {
    else {
        List<Int> temp = new ArrayList<>(sol);
        executorService.submit(() -> {
            for(int i=1; i<n; i+=2) {
                if(!temp.contains(i)) {
                    temp.add(i); back(n,T/2,temp);
                    temp.remove(temp.size()-1);
        });
        for(int i=0; i<n; i+=2)
            if(!sol.contains(i)) {
                sol.add(i); back(n,T-T/2,sol);
                sol.remove(sol.size()-1);
    { {
AtomicInteger sum = new AtomicInteger(0);
back(n,100,new ArrayList<Int>());
```

```
int size = n*n, chunk = size/(nrProc-1)
int begin = chunk*(id-1)
int end = chunk*id
if(id==nrProc-1) end = max(end, size)

int i = begin/n, j = begin%n
while(begin<end)
    r[i][j] = a[i][j] + b[i][j]
    j++;
    if(j>=n) {
        j=0;
        i++;
    {
    begin++;
```

MATRIX SUM

```
void run(String[] args) {
    MPI.Init(args);
    int rank = MPI.COMMWORLD.Rank();
    int nrProcs = ... .Size();
    if(rank==0) master(4,2,nrProcs);
    else worker(4,2,rank,nrProcs);
    MPI.FINALIZE();
```

COMBINATIONS

```
flatten matrix
    matrix.forEach(myList::addAll);
ExecutorService x =
    Executors.newFixedThreadPool(100);
Future<T> y = x.submit(()->{});
y.get() => get result when finished
x.shutdown();
```

condVar.await() releases its mutex

convolution: split N, each thread does

```
for(j=0, n-1, ++)
    c[i] += a[j] * b[i-j]
```

, i = part of the interval

```
big num prod = convolution +
                => list c
List<Int> res = new
int carry = 0;
for(i=c.size-1, 0, --)
    c[i] += carry
    res.add(c[i]%10)
    if(c[i]>9) carry = c[i]/10
    else carry = 0
while(carry>0)
    res.add(carry%10)
    carry /= 10;
```