PA2 code :

```
struct chrd
{int start; int end;};
```

Calculate the independent chords.

```
int findK( int key, int *chart);
void MISCalc(int **MIS ,chrd *input,int length ,int **statecheck,int
*chart);
void Chords(int start,int end,chrd * input, int **checkcase , chrd
*output, int *chart);
void mergesrt(chrd *output ,int begin, int ending);
void merge(chrd *output ,int begin,int mid1,int mid2, int ending);
```

function will be used

```
 //file input
  fstream fin(argv[1]);
  fstream fout;
  fout.open(argv[2],ios::out);
  //int getnumber;
  int num;
  vector<int> original_data;
  while (fin  >> num)
     original_data.push_back(num);
 int length=original_data[0];
  int datasize =original_data.size()/2-1;
```

get number and length

```
int result;
  int **MIS,*MISrow,**statecheck,*statecheckrow,**input,*inputrow;
  int **outputX ,*outputX_row;
  int **output,*outputrow;
  int i,j;
  MIS = (int**)malloc(length*sizeof(void*));  MISrow =
(int*)malloc(length*length*sizeof(int));
  statecheck = (int**)malloc(length*sizeof(void*));  statecheckrow =
(int*)malloc(length*length*sizeof(int));
  input=(int**)malloc(datasize*sizeof(void*));  inputrow=(int*)malloc(da
tasize*2*sizeof(int));
  for (i = 0; i < length; ++i, MISrow += length)   {MIS[i] = MISrow;}
  for (i = 0; i < length; ++i, statecheckrow += length)   {statecheck[i]
= statecheckrow;}
```

```
  for (i = 0; i < datasize; ++i, inputrow += 2)   {input[i] = inputrow;}
  chrd *inputC=new chrd[datasize];
  for(i=0; i<datasize; i++)
    {inputC[i].start=-1; inputC[i].end=-1;}
  for(int ig=1; ig<original_data.size()-1; ig+=2)
   {
      inputC[ig/2].start=original_data[ig];
      inputC[ig/2].end=original_data[ig+1];
   }
  int *findkarr=new int[length];
  for(int zx=0; zx<length;  zx++)
  {
       findkarr[zx]=-1;
  }
  for(int zx=0; zx<datasize; zx++)
  {
    findkarr[inputC[zx].start]=inputC[zx].end;
    findkarr[inputC[zx].end]=inputC[zx].start;
  }
  for (i = 0; i < length; ++i)
{
  for (j = 0; j < length; ++j)
  MIS[i][j] = 0 ,statecheck[i][j]=0;
}
  for(i=0; i<datasize; i++)
  {
    for(j=0; j<2; j++)
        input[i][j]=-1;
  }
  for(int ig=1; ig<original_data.size()-1; ig++)
   {  int mm=ig-1;
      input[mm/2][mm%2]=original_data[ig];
   }
```

Allocate space of upcoming caculation

```
MISCalc(MIS,inputC,length,statecheck,findkarr);
```

Draw mis chart and state check

```
void MISCalc(int **MIS ,chrd *input,int length ,int **statecheck,int
*chart)
```

```
{
     int k=0;
     //MIS is an (0 to length-1)*(0 to length -1) matrix with all zero
inside
     //dis means the distance between j-i mis[i][j] since dis=0;
mis[i][j]=0
     for(int dis=1; dis<length; dis++)//find all distance
     {
        // cout<<"distance  "<<dis<<endl;
        int  i=0,j=i+dis;
        while(j!=length){

            k=findK(j,chart);//find line witch k===
            if(k<i  || k>j)
            {
              MIS[i][j]=MIS[i][j-1];
              statecheck[i][j]=1;
            }
            else if(k > i && k < j)
            {
              if(MIS[i][j-1] >= (MIS[i][k-1] + MIS[k+1][j-1] + 1)) //if
the prvious is not ok
              {
                MIS[i][j]=MIS[i][j-1];
                statecheck[i][j]=1;
              }
              else
              {
                MIS[i][j]=MIS[i][k-1]+1+MIS[k+1][j];
                statecheck[i][j]=2;
              }

            }
            else if(k==i)
            {
              if(dis==1)
              {
                    MIS[i][j]=1;
```

```
                    statecheck[i][j]=3;
                }
                else

                  MIS[i][j]=MIS[i+1][j-1]+1;
                  statecheck[i][j]=3;
              }
              i++;j++;
          }
      }
}
```

//MIS function

```
result=MIS[0][length-1];
  chrd *outputC=new chrd[result];
  for(i=0; i<result; i++)
    {outputC[i].start=-1; outputC[i].end=-1;}
```

Output memory allocation

```
Chords(0,length,inputC,statecheck,outputC,findkarr);
void Chords(int start,int end,chrd * input, int **checkcase ,chrd
*output, int *chart ){

  while(end-start>1)
  {
   // cout<<checkcase[start][end]<<endl;
    if(checkcase[start][end]==3)
    {
      int k=findK(end,chart);
      //cout<<start<<" "<<k<<" "<<end<<endl;
    //cout<<"time  "<<ptr<<"  case"<< checkcase[start][end]<<endl;
     output[ptr].start=k;output[ptr].end=end;
   // cout<<answer[ptr][0]<<" "<<answer[ptr][1]<<" "<<endl;
    ptr++;
    Chords(start+1, end-1, input, checkcase,output,chart );
    end=start-1;

    }
    else if (checkcase[start][end]==2)//case2 M[i][j]=MIS[i][k-1]+1+MIS
    {
```

```
        int k=findK(end,chart);
        //cout<<ptr<<endl;
        output[ptr].start=k;output[ptr].end=end;
       // cout<<answer[ptr][0]<<" "<<answer[ptr][1]<<" "<<endl;
        ptr++;
        //Chords(start, k-1, input, checkcase, answer);
        Chords(k, end-1, input, checkcase,output,chart);
        end=k-1;
    }
    else  //case1
    {
      // M[i][j]=M[i][j-1]
        end--;
    }
  }
}
```

Calculate chords

```
 mergesrt(outputC ,0, result-1);
int fptr=0;
void mergesrt(chrd *output  ,int low, int high)
{
    if(low<high)
    {
        int mid1=(low+high)/2;
        int mid2=mid1+1;
        mergesrt(output,low,mid1);
        mergesrt(output,mid2,high);
        merge(output,low,mid1,mid2,high);
    }
}
void merge(chrd *output ,int low,int mid1,int mid2, int high)
{
    // TODO : Please complete the function
    int n_left=mid1-low+1,n_right=high-mid2+1;
    chrd *left_arr = new chrd[n_left];
    chrd *right_arr = new chrd[n_right];
    for(int i=0; i<n_left; i++ )
```

```
    {
      left_arr[i].start=output[low+i].start;
      left_arr[i].end=output[low+i].end;
    }

    for(int i=0; i<n_right; i++ )
    {
     right_arr[i].start=output[mid2+i].start;
      right_arr[i].end=output[mid2+i].end;
    }

    int lptr=0 ,rptr=0;
    int kptr=low;
    while (lptr<n_left && rptr<n_right){
        if (left_arr[lptr].start <= right_arr[rptr].start)  {
            output[kptr].start = left_arr[lptr].start;
            output[kptr].end = left_arr[lptr].end;
            lptr++;
        }
        else{
            output[kptr].start = right_arr[rptr].start;
            output[kptr].end = right_arr[rptr].end;
            rptr++;
        }
        kptr++;
    }
    while (lptr<n_left)  {
        output[kptr].start = left_arr[lptr].start;
        output[kptr].end = left_arr[lptr].end;
        lptr++;  kptr++;
    }
    while (rptr<n_right){
        output[kptr].start = right_arr[rptr].start;
         output[kptr].end = right_arr[rptr].end;
        rptr++;  kptr++;
    }
}
```

Modified merge sort meant to sort the first number.

```
fout<<result<<endl;
 for(int ix=0; ix<result; ix++)
 {
    fout<<outputC[ix].start<<" "<<outputC[ix].end<<endl;
 }
 fin.close();
 fout.close();
```

data output