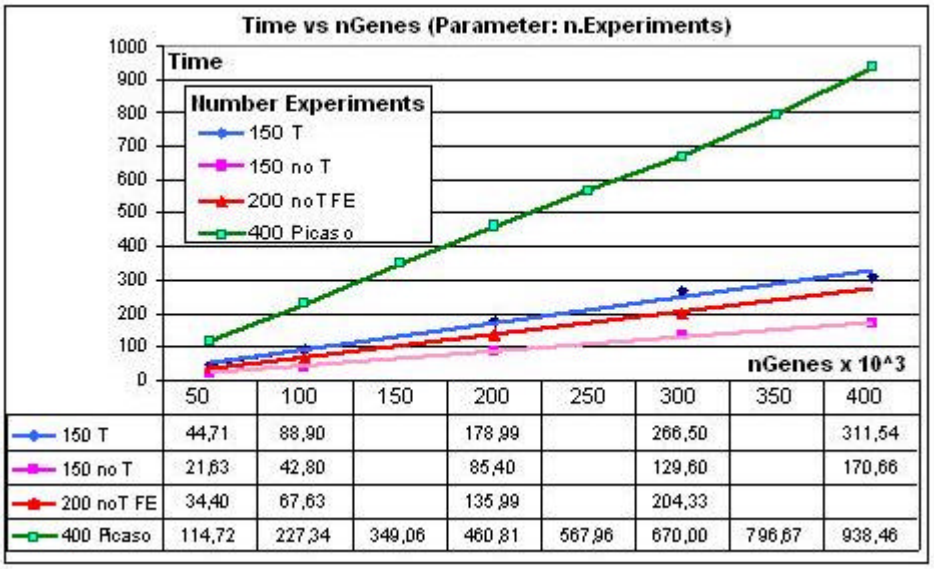
Input data:
4 GPR files containing more than 18 thousand genes were used to generate a test dataset. From each file, two input files were produced, one for each Mean signal in 635 and 532 channels.

Then, the files were themselves replicated to simulate more than 450.000 genes.

The basic "List of Files" (the input file) contains:

```
@ are used as comment lines
@ line format: fileName[TAB]nGenes[TAB]FileType[NEWLINE]
@ Types of files: t:text, one velue in each line
@
../files/GPR100/Exp1RB1-RT1cv.txt        450000 t
../files/GPR100/Exp1RB1-RT1cmean.txt     450000 t
../files/GPR100/Exp1RB1-RT1tv.txt        450000 t
../files/GPR100/Exp1RB1-RT1tmean.txt     450000 t
../files/GPR100/Exp1RB1-RT2cv.txt        450000 t
../files/GPR100/Exp1RB1-RT2cmean.txt     450000 t
../files/GPR100/Exp1RB1-RT2tv.txt        450000 t
../files/GPR100/Exp1RB1-RT2tmean.txt     450000 t
@
@ repeat the same files for testing
@
...
```

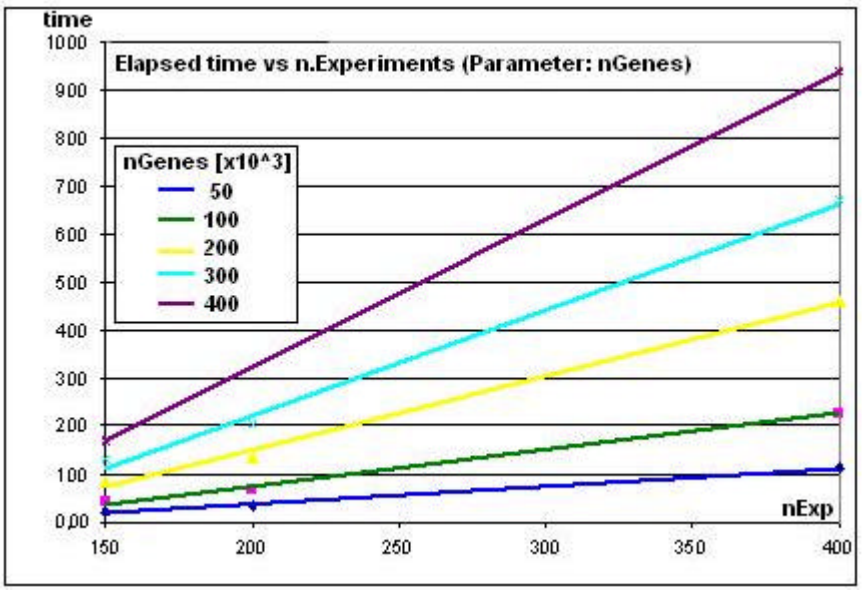
Benchmarking can be performed managing the command line paramenters to force a given number of genes (-g parameter) and a number of experiments (-e parameter).



	50	100	150	200	250	300	350	400
150 T	44,71	88,90		178,99		266,50		311,54
150 no T	21,63	42,80		85,40		129,60		170,66
200 noT FE	34,40	67,63		135,99		204,33		
400 Picaso	114,72	227,34	349,06	460,81	567,96	670,00	796,67	938,46

The first two runs correspond to 150 experiments and different number of genes (ranging from 50 to 400 thousand with increases of 50t). In this case the use or not of option –T (transpose the binary-matrix file into a tabulated-text file). Since this is not a mandatory effect of the program, and the influence is marked, all the rest of results will be obtained without using transposition (no option –T was used).
A second minor comment is that Step 1 consume much more time than step 2 (which in fact is good for us, since its parallelization is trivial and could be so efficient.

Obvously elapsed-time will evolve as Qsort complexity evolve. Basically linear with the number of genes and also linear with the number of experiments (keeping constant the number of genes)



Thus let's concentrate in the first step to distribute the first step (then next step will be addressed).

In the table the parallel code (under OMP) for the first step (sorting each experiment and produce the average column vector.

```
   //   Memory allocation and initializations

#pragma omp parallel shared(...)  private(tid, ...From, To, Range)
{
   tid = omp_get_thread_num();
   #pragma omp sections nowait
   { // first parallel section [1]
    #pragma omp section
    {
    // LOAD DISTRIBUTION-----------------
    Range = nE / nP;
    From = tid * Range;
    To   = (tid+1)*Range;
    if (To > nE) To = nE;

    for (i=From; i< To; i++) { // Qnorm for each datafile: STEP 1
                               // Replace  for (i=0; i< nE; i++)

        LoadFile(fList, i, dataIn);
        Perform ordering of the fList[i].fName
        AccumulateRow(AvG, dataIn , nG);
        Manage the Index in memory or send it to disk
   }

   } // End omp section [1]

   // complete in a second parallel section (by rows)
   for (i=0;i<nG;i++)    // Row average
       AvG[i].Av /=AvG[i].num;
   // produce the ORDERED output file [STEP 2]
   Prepare Output file and one column 'dataOut' array
   for (i=0;i<nE;i++) {
        Get the column index (from memory or from disk)
        for (j=0;j<nG;j++) { // complete the output vector
          dataOut[dIndex[j]]=AvG[j].Av;
          File positioning and writing the vector
        }
   }

   if (p->Traspose) TransposeBin2Txt(p);

  } // End omp sections nowait
} // omp parallel


}
```