Exercises for lab 4 of CS2101a

Instructor: Marc Moreno Maza

October 2, 2012

1 Exercise 1

What does the following program do?

```
#include <stdio.h>
#include <stdlib.h>

/* Transpose naively an n-by-n matrix */
void transpose_matrix(int* a,  int n)
{
  int i,j, tmp;

  for(i=0;i<n;i++) {
    for( j=i+1; j<n; j++) {
      tmp = a[i*n + j];
      a[i*n + j] = a[j*n + i];
      a[j*n + i] = tmp;
      }
  }
}

/* Print an n-by-n matrix */
void print_matrix(int* a,  int n)
{
  int i,j;
  for(i=0;i<n;i++) {
    for( j=0; j<n; j++) {
      printf("%d ", a[n*i+j]);
      if (j == n-1) printf("\n");
    }
  }
  printf("\n");
}

/* Create a random n-by-n matrix */
```

```
void random_matrix(int* a,  int n)
{
  int i,j;

  for(i=0;i<n;i++) {
    for(j=0;j<n;j++) {
      a[i*n + j] = rand()%n;
    }
  }
}

int main() {
  int n, s;
  int* a;
  printf("n = ");
  scanf("%d", &n);
  printf("\n");
  s = n * n;
  if (s < 1000000000) {
    printf("s = %d\n", s);
    a = (int *) malloc(s * sizeof(int));
    random_matrix(a,n);
    if (n < 10) print_matrix(a,n);
    transpose_matrix(a,n);
    if (n < 10) print_matrix(a,n);
  }
  free(a);
  return 0;
}
```

Using the UNIX `time` command, measure the running time of this program when $n = 2^k$ for $k = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14$.

2 Exercise 2

We investigate another approach for computing the transpose ${}^tA$ of a square matrix A. This approach is based on a divide and conquer scheme. In the formula below, we assume that n is a power of 2 and that $A_{1,1}, A_{1,2}, A_{2,1}, A_{2,2}$ denote square blocks of order $n/2$.

$$ {}^tA = \begin{cases} \begin{pmatrix} {}^tA_{1,1} & {}^tA_{2,1} \\ {}^tA_{1,2} & {}^tA_{2,2} \end{pmatrix} & \text{if} \quad A = \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \\ A & \text{if} \quad n = 1 \end{cases} \tag{1} $$

Write a C program that successively

- reads a positive integer value n from the user,
- generate an $n \times n$ matrix `a` with random entries of type `int` with values in the range $0 \cdots n-1$.
- transpose the matrix in place using this divide-and-conquer approach.

Using the UNIX `time` command, measure the running time of this program when $n = 2^k$ for $k = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14$.

3 Exercise 3

A drawback of the approach of Exercise 2 is the overhead due to the recursive calls. One way to reduce this negative impact is to modify the above formula as follows

$$ {}^tA = \begin{cases} \text{naive\_Transpose } A & \text{if } n \leq B \\ \begin{pmatrix} {}^tA_{1,1} & {}^tA_{2,1} \\ {}^tA_{1,2} & {}^tA_{2,2} \end{pmatrix} & \text{if} \quad A = \begin{pmatrix} A_{1,1} & A_{1,2} \\ {}^tA_{2,1} & A_{2,2} \end{pmatrix} \quad \text{else} \end{cases} \tag{2} $$

where

- B is a *base-case*, which is typically a power of 2 in the range $16 \cdots 256$,
- naive_Transpose refers to the algorithm of Exercise 1.

1. Modify the program of Exercise 2 so as to use a base-case.
2. Determine what is the best base-case for your machine.
3. Using the UNIX `time` command, measure the running time of this program when $n = 2^k$ for $k = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14$.
4. In principle, this new program should perform better than the one of Exercise 1. Explain why.

4 Exercise 4

Another way to implement the approach of Exercise 3 is to use a blocking strategy. Let b be a positive integer dividing n.

1. We decompose the matrix A into $b \times b$-blocks.

$$ \begin{pmatrix} B_{1,1} & \cdots & B_{1,n/b} \\ \vdots & \vdots & \vdots \\ B_{n/b,1} & \cdots & B_{n/b,n/b} \end{pmatrix} \tag{3} $$

2. For each $i = 1 \cdots n/b$ transpose the block $B_{i,i}$ in place.

3. For each $i = 1 \cdots n/b$ for each $j = i + 1 \cdots n/b$ exchange and transpose the blocks $B_{i,j}$ and $B_{j,i}$.

1. Modify the program of Exercise 1 so as to implement this blocking strategy.

2. Determine what is the best base-case b for your machine.

3. Using the UNIX `time` command, measure the running time of this program when $n = 2^k$ for $k = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14$.

4. In principle, this new program should perform better than the one of Exercise 1. Explain why.