HW#2

Due date: Problems 1~6: 4/2; Problem 7: 4/9

1 Do Ex. 4.3-6 (10%)

Hint: $T(n) = 2T\left(\left\lfloor\frac{n}{2}\right\rfloor + 17\right) + n \leq 2T\left(\frac{n}{2} + 17\right) + n$

Find a constant α such that $S(n) = T(n+\alpha)$ and $S(n) \leq 2S(n/2) + O(n)$. This technique is called *domain transformation*, i.e. transform the domain $n+\alpha$ to n.

2 Do Ex. 4.3-7 (10%)

3 Do Problem 4-1 a) ~ f). Use the master theorem. (25%)

4 Do Problem 4-1 for the following two recurrences.

a) $T(n) = T\left(n/2 + \sqrt{n}\right) + n$ (10%)

Hint: Find recurrences $S(n)$ and $U(n)$ such that $S(n) \leq T(n) \leq U(n)$. Apply the master theorem.

b) $T(n) = T(n/2) + T\left(\sqrt{n}\right) + n$ (10%)

Hint: Compare the two terms $T(n/2)$ and $T\left(\sqrt{n}\right)$ for sufficiently large n. Guess a solution and prove it by structural induction.

5 Let $A[1..n]$ be an array of distinct signed integers in increasing order. Find an index $i, 1 \leq i \leq n$, such that $A[i] = i$. If there is no such an index, return 0; if there are more than one index, return any one.

a) Design a divide-and-conquer algorithm to solve this problem. Write down the pseudocode of your algorithm. (10%)

b) Let $T(n)$ be the worst-case running time of your algorithm.
Write down the "exact" recurrence for $T(n)$, i.e. don't ignore the floor and ceiling functions as well as the boundary condition. (5%)

c) Find the tight bound of $T(n)$. (5%)

6 [Past exam question]

Consider a variant of mergesort that divides an array of n elements into $\sqrt{n}$ subarrays, each having $\sqrt{n}$ elements. The $\sqrt{n}$ sorted subarrays are then merged simultaneously with the help of a min-priority queue.

MERGESORT$(A[1..n])$

1 **for** $i = 1$ to $\sqrt{n}$ do // $\sqrt{n}$ subarrays

MERGESORT$\left(A\left[(i-1)\sqrt{n}+1..i\sqrt{n}\right]\right)$ // sort the ith subarray

2 MERGE$(A[1..n])$

MERGE$(A[1..n])$ // $A[1..n]$ contains $\sqrt{n}$ sorted subarrays

1 Let $B[1..n]$ be a new array

2 Build a min-priority queue Q on the $\sqrt{n}$ smallest elements, one from each sorted subarray

3 **for** $k = 1$ to n do

$B[k] =$ EXTRACT-MIN(Q)

// Suppose the element just extracted comes from the ith subarray

if the ith subarray is not empty **then**

INSERT$(Q,$ the next element of the ith subarray$)$

4 Copy $B[1..n]$ back to $A[1..n]$

a) Show that the running time of MERGE$(A[1..n])$ is $O\left(n \lg \sqrt{n}\right)$. (5%)

b) Let $T(n)$ be the running time of MERGESORT$(A[1..n])$, then

$T(n) = \sqrt{n}T\left(\sqrt{n}\right) + O\left(n \lg \sqrt{n}\right)$

Give an asymptotic upper bound for $T(n)$. (10%)

Hint

First, let $S(n) = T(n)/n$.

This technique is called *range transformation*, i.e. transform the range $T(n)$ to $T(n)/n$.

Next, change of variable.

7 [Programming exercise] 100%

Implement the remarkable Strassen's algorithm and the classic $O(n^3)$ matrix multiplication algorithm. The classic algorithm is primarily used to check the correctness of your implementation of Strassen's algorithm.

Requirements

1 [Strassen's algorithm]

In case $n > 1$ is odd, you shall enlarge the $n \times n$ square matrices to be of size $(n+1) \times (n+1)$, filling the extra row and column with 0's, so that the half-size submatrices remain square. The extra row and column shall then be stripped off from the matrix obtained by multiplying two enlarged matrices.

2 You shall use C++ to write a template class

```
template<typename T> matrix;
```

The exact implementation of this class is up to you, as long as the following sample test is runnable.

```
int main()
{
    const int n=9;
    matrix<int> A(n),B(n);                // 1
    for (int i=0;i<n;i++)
        for (int j=0;j<n;j++) {
            A[i][j]=rand()%5;             // 2
            B[i][j]=rand()%5;
    }
    cout << "Matrix A\n" << A;            // 3
    cout << "Matrix B\n" << B;
    cout << "Matrix AxB\n" << A*B;        // 4
    cout << "Matrix A^B\n" << (A^B);      // 5
}
```

You need at least the following functions.

Line 1: Implement a constructor to construct $n \times n$ matrices

`matrix<int> A(n),B(n);` // all elements initialized to 0

`matrix<int> C(n,2);` // all elements initialized to 2

Line 2: Overload `operator[]` to return a reference to some row of a matrix, e.g. `A[i]` references to the `i`th row of matrix `A`.

Line 3: Overload `operator<<` to output the contents of a matrix.

Line 4: Overload `operator*` to implement Strassen's algorithm. To this end, overload `operator+` and `operator-` to add and subtract two matrices, respectively.

Line 5: Overload `operator^` to implement the classic algorithm.

The output of this program depends on the implementation of `rand()`. The sample output given in the next page is for illustration only. Any consistent outputs will be accepted.

3 You shall turn in a program that contains the preceding sample test.

Bonus

The author of the fast implementation of Strassen's algorithm will gain one final grade point. For this bonus point, you shall turn in another program that contains the following speed test.

```
#include <ctime>
int main()
{
    matrix<int> A(200,1),B(200,2);
    clock_t start=clock();
    matrix<int> C(A*B);
    clock_t finish=clock();
    cout << (double)(finish-start)/CLOCKS_PER_SEC;
    cout << " seconds\n";
}
```

The running time will be measured under g++48 on bsd2:
bsd2> g++48 –std=c++11 strassen.cpp

Suggestion: Use move semantics to improve the speed of your program.

Sample output

```
Matrix A
   2   3   0   4   3   0   2   2   2
   0   3   4   0   4   1   2   2   0
   4   4   2   3   0   3   4   1   3
   3   4   0   2   4   1   2   3   3
   0   0   3   0   0   4   2   0   1
   3   0   2   1   3   1   1   2   2
   4   4   0   0   2   3   0   4   0
   3   1   2   2   2   0   1   0   3
   1   0   0   2   0   4   4   0   3
```

Matrix B

4	3	2	3	4	0	2	3	4
2	4	2	3	3	0	1	2	3
4	1	2	1	0	1	2	3	4
0	1	3	1	4	3	3	3	1
3	4	1	4	4	1	0	4	3
3	2	1	2	4	2	3	3	2
4	1	0	3	4	2	2	3	3
0	2	0	2	1	2	1	1	2
0	4	1	3	4	3	0	1	1

Matrix AxB

31	48	27	47	63	29	25	46	42
45	40	19	41	39	18	20	45	49
57	57	35	58	81	36	43	60	62
43	65	28	62	75	31	26	54	55
32	17	11	20	28	18	22	28	27
36	39	19	39	46	22	20	40	41
39	50	21	46	52	16	25	41	48
32	38	23	36	47	21	19	37	37
32	29	15	34	56	31	28	36	29

Matrix A^B

31	48	27	47	63	29	25	46	42
45	40	19	41	39	18	20	45	49
57	57	35	58	81	36	43	60	62
43	65	28	62	75	31	26	54	55
32	17	11	20	28	18	22	28	27
36	39	19	39	46	22	20	40	41
39	50	21	46	52	16	25	41	48
32	38	23	36	47	21	19	37	37
32	29	15	34	56	31	28	36	29