CSC 120 Algorithms and Data Structures **Spring Semester, 2002**

Midterm Review

The Exam is on **Thursday, 3/21 in class**!
We will have review on Wednesday, common hour in Room 200 Adams.

The exam is closed book and notes, but you are allowed to bring ONE page 8x11 with relevant definitions, and formulae. You should NOT have specific solutions of recurrences, or solutions to problems. No calculators!

In case of sickness or other emergencies which may prevent you of taking the exam, please notify Prof. Kamberova as soon as possible, prior to the exam, `cscglk@husun3.hofstra.edu`, tel (516) 463-5775. For medical excuses present doctor's note.

The problem set in this review sheet will give you an idea about the level of difficulty of the exam.

Scope of the exam

Material we have covered in lectures, assigned textbook readings, homeworks, and quizzes, the material covered up to 3/14 class.

Go over this problem set, over the problems from homeworks 1 to 4 (without extra credit problems), and Quizz 1.

The review will be useful only if you have tried the problems yourself first!

Practice problems

In all problems whenever you solve recurrences you should check your answer, unless otherwise specified, you may use the Master method when appropriate.

1. For the following program fragment compute the worst-case asymptotic time complexity (as a function of n). Whenever it says "loop body" you can assume that a constant number of statements are there. Show your work. your answer.

```
for (i=0; i<=n-1; i++){
   for (j=0; j< 5; j++) {
       for (k=0; k<n; k++) {
      loop body
}
for (i=0; i<=n-1; i++){
   for (j=0; j<=i; j++){
      loop body
   }
}
```

2. Suppose you have three algorithms to solve a problem (with an input size of n). You are given that The first has asymptotic time complexity $\Theta(n^2)$, the second has asymptotic time complexity $O(n\sqrt{n})$, and the third has asymptotic time complexity $\Omega(n \log, n)$. Which algorithm would you choose? Briefly explain.

3. Answer each of the following questions AND justify your answer.

   (a) Assume that f and g take only positive values. Is $f(n)+g(n) = O(\max\{f(n), g(n)\})$? Justify your answer.

   (b) If $f(n) = O(g(n))$ then does it follow that $g(n) = O(f(n))$?

4. What is the worst-case asymptotic time complexity of the following divide-and-conquer algorithm. You may assume that n is a power of 2. (NOTE: It doesn't matter what this algorithm does.)

```
foo(n,A){
  let B be an array of size n
  if (n==1){
    B[0] = 1;
    return B;
  }

  let AL be an array of size n/2
  let AR be an array of size n/2
  for (i=0; i <= (n/2)-1; i++)
    AL[i] = A[i];
  for (i=n/2; i <= n-1; i++)
    AR[i- (n/2)] = A[i];

  BL = foo(n/2,AL);
  BR = foo(n/2,AR);

  for (i=0; i<= n-1 ; i++)
     B[i] = 1;
  for (i=n/2; i <= n-1; i++)       //for each element on right half
     for (j=0; j <= (n/2)-1; j++)  //  for each element on the left half
        if (A[i-n/2] > A[j])
           B[i] = max(B[i],BR[i]+BL[j]);
  return B;
}
```

5. Design a divide and conquer algorithm, `NewMinAlgo(A,n)` that takes an array `A` of size `n` and finds and returns the minimal element.

   (a) Give clearly the algorithm in C++.

(b) Evaluate the time complexity of your algorithm (justify your answer, show your work).

(c) Compare the complexity of your algorithm with the complexity of the "brute force" algorithm, which is

```
int BruteMinAlgo(int A[],n) {
  min=A[0];
  for (i=0; i<n ; i++)
     if (A[i] < min)
        min=A[i];
  return min;
}
```

(d) Of the two algorithms which one would you choose? Explain why.

6. For the following procedures:

- Give the **exact** running times as a function of n. Write recurrences when appropriate, and solve them. You cannot use the Master method since it does not give exact run times.
- Now, give the asymptotic time complexity of each of the running times you obtained in the first part.
- And, last, for the run time expression which accept the master method, give the complexity using the Master method.

(a)
```
Goo(positive integer n)
  if (n==1)
     return 1
  a = Goo(n-1) + Goo(n-1)
  return a
```

(b)
```
Pan(positive integer n)
  result=0
  for (i=2; i<=n; i++)
    result = result + Foo(i)  //Foo(i) requires Theta(i) time,i.e. c*i
  return result
```

(c)
```
Merge(positive integer n)
  if ( n==1)
     return 1
  result = 0
  temp = Merge(n/2) + Merge(n/2)
  for (i=1; i <= n; i++)
        temp = temp + i;
  Homer(n)                 //Homer(n) requires constant time
  return temp;
```

7. Give tight asymptotic bounds for $T(n)$ in each of the following recurrences. Assume that $T(1) = 1$. Use the master method.)

   (a) $T(n) = 16T\left(\frac{n}{2}\right) + \Theta(1)$

   (b) $T(n) = T\left(\frac{3n}{4}\right) + \Theta(\sqrt{n})$

   (c) $T(n) = 9T(n/9) + \Theta(n \log^2 n)$

8. Consider the following array

   ```
   120, 3, 17, 11, 101, 15, 8, 10
   ```

   The way we did tracing in class: trace merge sort , and then quick sort on that array. That is, "going down", build a tree at each node showing the subarray on which a recursive call is made, and "coming back up", under each node, in parenthesis, give the result of that call.

   I have started for you the merge sort case. You have to continue building the tree. Do then similar the quick sort. You do not have to trace Partition().

```
  120, 3, 17, 11, 101, 15, 8, 10
 (                              )
             /     \
            /       \


 120, 3, 17, 11       101, 15, 8, 10
(3,  11, 17, 120)     (8, 10, 15, 101)
```