

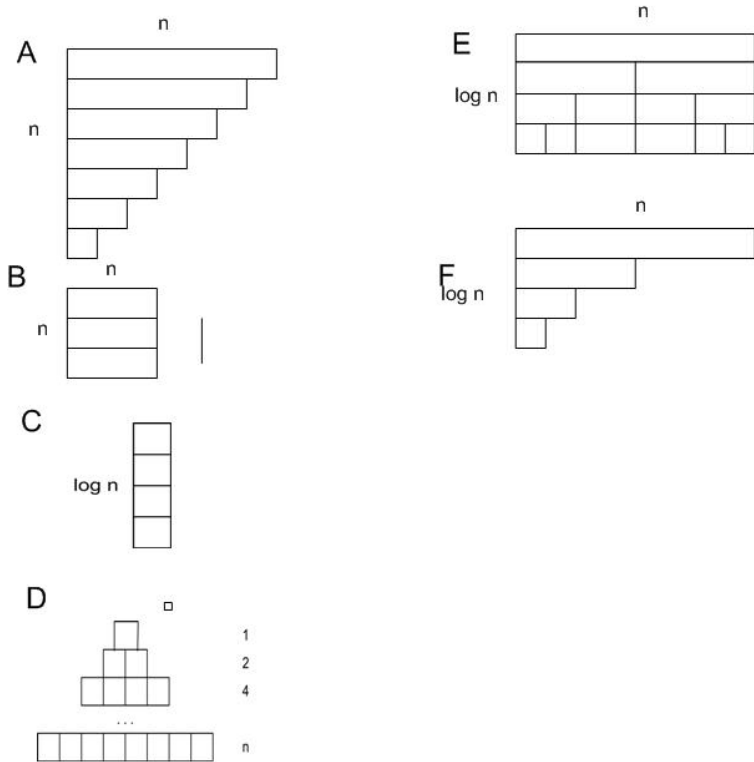
# CS5050 Homework 1

Due September 12, 2008, (due at class time) 10 points

Written homework provides an excellent framework for achieving the goals of obtaining a working knowledge of data structures, perfecting programming skills, and developing critical thinking strategies to aid the design and evaluation of algorithms. Since programming has a high overhead in terms of program entry and debugging, all important topics in this course cannot be covered via programming projects. Written homework exercises allow students to learn important material without a high time investment. Although the point value is low, the benefits are great. You can perfect your design skills without spending hours at the computer and can get feedback on your thinking skills from your study partners. Students who consistently do quality homework have far superior test scores.

Because assignments are done as a group and any questions are discussed in class or during office hours, written solutions to the homework will not be provided.

1. The terminology  $f(n)$  is  $O(n^2)$  is equivalent to saying  $f(n)$  is of order  $n^2$  or saying  $f(n)$  is of complexity  $n^2$ . For each problem, (1) Find the complexity (2) select an appropriate picture from the list below (A-F) (or draw one of your own) to justify your answer.



- a. 

```
for (int k = 0; k<n;k++)
    for (int j = 0; j < k; j++) // Notice j ends at k (not n)
        x = x++;
```
- b. 

```
void doit(int n)
{ if (n <=0) return;
  doit(n/2);
  doit(n/2);
  for (int i=0;i<n;i++)
    cout << i;
}
```
- c. 

```
void doit(int n)
{ if (n <=0) return;
  doit(n/2);
  doit(n/2);
  x++;
}
```
- d. 

```
int doit(int look,int beg, int end)
{ if (beg > end) return -1;
  mid = (beg + end)/2;
  if (A[mid] == look)
    return mid;
  if (look < A[mid])
    return doit(look, beg,mid-1);
  return doit(look, mid+1,end);
}
```

2. Consider the following sample data. For each set of timing information, indicate the complexity of the algorithm.

a.

$n$	$T(n)$
2	5
4	5
8	5
16	5
32	5

b.

$n$	$T(n)$
2	5
4	10
8	20
16	40
32	80

c.

$n$	$T(n)$
2	5
4	10

8	15
16	20
32	25

d.

$n$	$T(n)$
2	4
4	16
8	256
16	65,536
32	4,294,967,296

e.

$n$	$T(n)$
2	10
4	16
8	48
16	128
32	320

f.

$n$	$T(n)$
2	4
4	15
8	68
16	270
32	1024

3. Suppose that  $T_1(N) = O(F(N))$  and  $T_2(N) = O(F(N))$ . *Note that if something is  $n^2$ , the run time may actually be  $10n^2 + 47n + 15$  as you throw out constants and ignore lesser order terms.*

Which of the following are true:

- $T_1(N) + T_2(N) = O(F(N))$
  - $T_1(N) - T_2(N) = O(F(N))$
  - $T_1(N)/T_2(N) = O(1)$
  - $T_1(N) = O(T_2(N))$
4. Program A and B are analyzed and are found to have worst-case running times no greater than  $O(n \log n)$  and  $O(n^2)$ , respectively. Answer the following questions if possible.
- Which program has the better guarantee on the running time for large values of  $N$  ( $N > 10,000$ )?
  - Which program has the better guarantee on the running time for small values of  $N$  ( $N < 100$ )?
  - Which program will run faster *on average* for  $N = 1000$ ?

- d. Can program B run faster than program A on *all* possible inputs?
5. What is the order of each of the following tasks in the worst case (using the best algorithm)?
- Computing the sum of the first  $n$  even integers by using a for loop.
  - Displaying all  $n$  integers in a sorted linked list.
  - Displaying the last element in a linked list.
  - Searching an array of  $n$  integers for a particular value by using sequential search.
  - Searching an array of  $n$  integers for a particular value by using a binary search.
  - Adding an item to a stack of  $n$  items.
  - Adding an item to a queue of  $n$  items.
6. An implementation of an algorithm has the following loop structure. What is the complexity of the algorithm?
- ```
for ( int pass = 1; pass <= n; ++pass )
  for ( int index = 0; index < n; ++index )
    for ( int count = 1; count < 10; ++count )
      x++;
```

### Formula Approach

The following theorem gives the mathematical computation for what we have been analyzing visually. This formula approach for computing complexity is only appropriate for recursion and only when the problem size decreases by a factor.

Let:

- $T(n)$  – amount of time to solve a problem of size  $n$ .
- $a$  – Number of recursive calls made.
- $b$  – Number  $n$  is divided by in the recursive calls.
- $k$  – The **exponent on  $n$**  which represents the amount of work in a single call.

Theorem:

- $T(n) = aT(n/b) + O(n^k)$ .
- If  $a > b^k$  then  $T(n)$  is  $O(n^{\log_b a})$ .
- If  $a = b^k$  then  $T(n)$  is  $O(n^k \log n)$ .
- If  $a < b^k$  then  $T(n)$  is  $O(n^k)$ .

7. According to the previous theorem, what is the complexity of the following piece of code?

```
void doit (n){
  if (n <=1) return;
  doit(n/2);
  doit(n/2);
  doit(n/2);
  for(i=0;i<n;i++) x++;
```

}

8. According to the previous theorem, what is the complexity of the following piece of code?

```
void doit (n){
  if (n <=1) return;
  for(i=0;i<n;i++)
    for (j=0; j < n; j++) x++;
  doit(n/2);
}
```

9. A problem with complexity  $O(n)$  doubles in size, what would you expect would happen to the execution time?
- Remains unchanged
  - Increases by a constant
  - Doubles
  - More than doubles
  - Quadruples (increases by four times)
  - Is squared
10. A problem with complexity  $O(1)$  doubles in size, what would you expect would happen to the execution time?
- Remains unchanged
  - Increases by a constant
  - Doubles
  - More than doubles
  - Quadruples (increases by four times)
  - Is squared
11. A problem with complexity  $O(n^2)$  doubles in size, what would you expect would happen to the execution time?
- Remains unchanged
  - Increases by a constant
  - Doubles
  - More than doubles
  - Quadruples (increases by four times)
  - Is squared
12. Find a counterexample to the following claim:

$f(n) = O(s(n))$  and  $g(n) = O(r(n))$  imply that  $f(n)-g(n) = O(s(n)-r(n))$ .

Be sure to use regular big Oh rules, such as  $O(n^2+n) = O(n^2)$  as you throw out lower order terms.

13. With Big Oh notation, we remove constants. Thus, if one algorithm is twice as fast as another, they still have the same complexity. How do we justify this sloppy accounting?
14. For what values of  $n$  is  $100 * \log_2 n < n^2$ ?
15. If one algorithm is  $O(\log_2 n)$  and another algorithm (for solving the same problem) is  $O(n^2)$ , is the first always better than the second in terms of time? Be sure to consider various values of  $n$  and various constants.
16. It is often helpful to think of the general form for a divide and conquer problem. Given that  $T(n) = a * T(n/b) + c * n^k$  where  $cn^k$  is work of combining,  $a$  is number of pieces examined, and  $n/b$  is the size of a piece.

For the following examples, identify  $a, b, k$ . Verify that the formulas of the theorem produce the correct complexity.

1. Finding the largest element in a binary tree by comparing the largest elements of the two subtrees.
2. Locating an item in a binary search tree.
3. Performing a mergesort.

Theorem (gives less information than the theorem from the book, but is easier to use):

$$T(n) = a T(n/b) + O(n^k)$$

$$\text{if } a > b^k \quad O(n^{\log_b a})$$

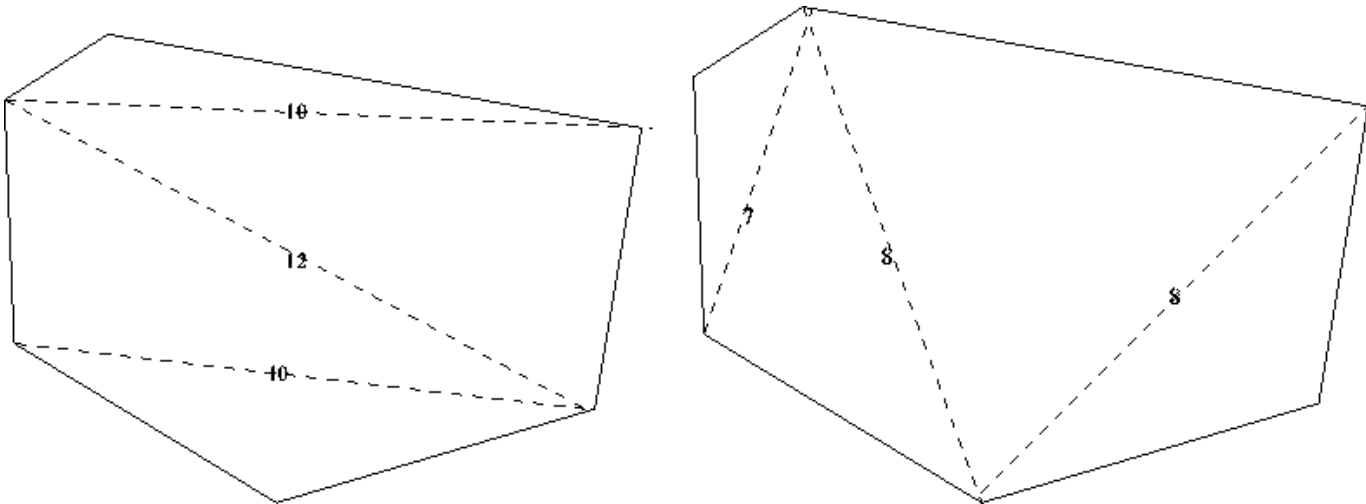
$$\text{if } a = b^k \quad O(n^k \log n)$$

$$\text{if } a < b^k \quad O(n^k)$$

17. Design a divide and conquer algorithm for finding the minimum and maximum elements of  $n$  numbers which uses no more than  $3n/2$  comparisons. Show the recurrence relation. Using an  $n$  of size 16, how many comparisons are required? Hint, be careful in choosing your base case.
18. Suppose we have a collection,  $A$ , of  $n$  positive integers that add to  $N$ . Design an efficient algorithm for determining whether there is a subset  $B$  of  $A$  such that the sum of the numbers in the subset equals the sum of the numbers of  $A$  which are not chosen for the subset. **In describing your algorithm, use your algorithm vocabulary (greedy, dynamic programming, brute force). If the algorithm happened to have a well known name, you could just refer to the name of the algorithm without going through the details. In describing your algorithms, be sure to explain any variables you use.**
19. A convex polygon is a polygon in which a line connecting any two vertices lies entirely within the polygon. A triangulation of a convex polygon,  $P$ , is an addition of diagonals connecting the vertices of  $P$  so that triangles are formed. The weight of a triangulation is the sum of the lengths of the diagonals. Give an efficient

algorithm for computing a minimum weight triangulation. In describing your algorithm, use your algorithm vocabulary (greedy, dynamic programming, brute force). If the algorithm happened to have a well known name, you could just refer to the name of the algorithm without going through the details. In describing your algorithms, be sure to explain any variables you use.

The example below shows two possible triangulations of the polygon. Assuming the labels represent weight, the right triangulation is better.



(In case you are interested, triangulation is a fundamental problem in computational geometry, because the first step in working with complicated geometric objects is to break them into simple geometric objects. The simplest geometric objects are triangles in two dimensions, and tetrahedra in three. Classical applications of triangulation include finite element analysis and computer graphics.

A particularly interesting application of triangulation is surface or function interpolation. Suppose that we have sampled the height of a mountain at a certain number of points. How can we estimate the height at any point  $q$  in the plane? If we project the points on the plane, and then triangulate them, the triangulation completely partitions the plane into regions. We can estimate the height of  $q$  by interpolating among the three points of the triangle that contains it. Further, this triangulation and the associated height values define a surface of the mountain suitable for graphics rendering.)