STUDENT OUTLINE

Lesson 24 – Order of Algorithms

INTRODUCTION: The two criteria used for selecting a data structure and algorithm are the amount of memory required and the speed of execution. The analysis of the speed of an algorithm leads to a summary statement called the order of an algorithm.

The key topics for this lesson are:

A. Order of Algorithms
B. Constant Algorithms, O(1)
C. $\log_2 N$ Algorithms, O($\log_2 N$)
D. Linear Algorithms, O(N)
E. N * $\log_2 N$ Algorithms, O(N * $\log_2 N$)
F. Quadratic Algorithms, (N^2)
G. Other Orders
H. Comparison of Orders of Algorithms

VOCABULARY:

ORDER OF ALGORITHM	CONSTANT
LOG_2 N	LINEAR
N LOG_2 N	QUADRATIC
CUBIC	BIG O NOTATION

DISCUSSION:

A. Order of Algorithms

1. The order of an algorithm is based on the number of steps that it takes to complete a task. Time is not a valid measuring stick because computers have different processing speeds. We want a method of comparing algorithms that is independent of computing environment and microprocessor speeds.

2. Most algorithms solve problems involving an amount of data, N. The order of algorithms will be expressed as a function of N, the size of the data set.

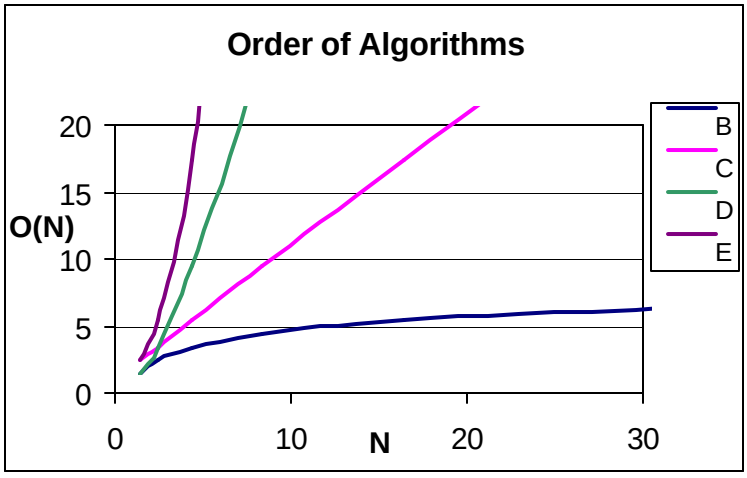
3. The following chart summarizes the numerical relationships of common functions of N.

A N	B $O(\log_2 N)$	C O(N)	D $O(N^* \log_2 N)$	E $O(N^2)$
1	0	1	0	1
2	1	2	2	4
4	2	4	8	16
8	3	8	24	64
16	4	16	64	256
32	5	32	160	1024
64	6	64	384	4096
128	7	128	896	16384
256	8	256	2048	65536
512	9	512	4608	262144
1024	10	1024	10240	1048576

a. The first column, N, is the number of items in a data set.

b. The other four columns are mathematical functions based on the size of N. In computer science, we write this with a capital O (order) instead of the traditional F (function) of mathematics. This type of notation is the order of an algorithm, or *Big O notation*.

c. The graph below gives a clearer sense of the relationships among the columns of numbers. Since the vertical axis represents the theoretical number of steps required by an algorithm to sort a list of N items, lines B and C represent more efficient algorithms than D and E. Today's data sets can grow to enormous sizes, so algorithm designers are always looking for ways to reduce the number of steps, even on the fastest supercomputers.



d. You have already seen column E in an experimental sense when you counted the number of steps in the quadratic sorting algorithms. The relationship between columns A and E is quadratic - as the value of N increases, the other column increases as a function of N^2. The graph of column E is a portion of a parabola.

B. Constant Algorithms, O(1)

1. This relationship was not included in the chart. Here, the size of the data set does not affect the number of steps this type of algorithm takes. For example:

```
int howBig (int[] list)
{
   return list.length;
}
```

2. The number of data items in the array could vary from 0…4000, but this does not affect the `howBig` algorithm. It will take one step regardless of how big the data set is.

3. A constant time algorithm could have more than just one step, as long as the number of steps is independent of the size (N) of the data set.

C. Log_2N Algorithms, $O(log_2N)$ – line B on the graph

1. A logarithm is the exponent to which a base number must be raised to equal a given number.

2. A log_2N algorithm is one where the number of steps increases as a function of log_2N. If the number of data was 1024, the number of steps equals log_21024, or 10 steps.

3. Algorithms in this category involve splitting the data set in half repeatedly. Several examples will be encountered later in the course.

4. Algorithms that fit in this category are classed as O(log N), regardless of the numerical base used in the analysis.

D. Linear Algorithms, O(N) – line C on the graph

1. This is an algorithm where the number of steps is directly proportional to the size of the data set. As N increases, the number of steps also increases.

```
long sumData (int[] list)
//  sums all the values in the array
{
  long total = 0;

  for (int loop = 0; loop < list.length; loop++)
  {
    total += list[loop];
  }
  return total;
}
```

2. In the above example, as the size of the array increases, so the number of steps increases at the same rate.

3. A non-recursive linear algorithm, O(N), always has a loop involved.

4. Recursive algorithms, in which the looping concept is developed through recursive calls, are usually linear. For example, the recursive factorial function is a linear function.

```
long fact (int  n)
//  precondition:  n > 0
{
  if (1 == n)
    return 1;
  else
    return  n * fact(n - 1);
}
```

The number of calls of fact will be n. Inside of the function is one basic step, an **if/else**. So we are executing one statement *n* times.

E. N * $\log_2$N Algorithms, O(N * $\log_2$N) – line D on the graph

1. Algorithms of this type have a $\log_2$N procedure that must be applied N times.

2. When recursive MergeSort and Quicksort are covered, we will discover that they are O(N * $\log_2$N) algorithms.

3. As the graph shows, these algorithms are markedly more efficient than our next category, quadratics.

F. Quadratic Algorithms, (N^2) – line E on the graph

1. This is an algorithm in which the number of steps required to solve a problem increases as a function of N^2. For example, here is bubbleSort.

```
void bubbleSort (int[][] list)
{
  for (int outer = 0; outer < list.length - 1; outer++)
  {
    for (int inner = 0; inner <= list.length - outer; inner++)
    {
      if (list[inner] > list[inner + 1])
      {
        // swap list[inner] & list[inner + 1]
        int temp = list[inner];
        list[inner] = list[inner + 1]);
        list[inner + 1] = temp;
      }
    }
  }
}
```

2. The **`if`** statement is buried inside nested loops, each of which is tied to the size of the data set, N. The **`if`** statement is going to be executed approximately N times for each of N items, or N^2 times in all.

3. The efficiency of this bubble sort was slightly improved by having the inner loop decrease. But we still categorize this as a quadratic algorithm.

4. For example, the number of times the inner loop happens varies from 1 to (N-1). On average, the inner loop occurs (N/2) times.

5. The outer loop happens (N-1) times, or rounded off N times.

6. The number of times the **if** statement is executed is equal to this expression:

   # if statements = (Outer loop) * (Inner loop)

   # if statements = (N) * $(\frac{N}{2})$

   # if statements = $\frac{N^2}{2}$

7. The coefficient $\frac{1}{2}$ becomes insignificant for large values of N, so we have an algorithm that is quadratic in nature.

8. When determining the order of an algorithm, we are only concerned with its category, not a detailed analysis of the number of steps.

G. Other Orders

1. A cubic algorithm is one where the number of steps increases as a cube of N, or N^3.

2. An exponential algorithm is one where the number of steps increases as the power of a base, like 2^N.

3. Both of these categories are astronomical in the number of steps required. Such algorithms are avoided when possible, and for large values of N can run slowly on small computers.

H. Comparison of Orders of Algorithms

1. We obviously want to use the most efficient algorithm in our programs. Whenever possible, choose an algorithm that requires the fewest number of steps to process data.

See Transparency T.A.24.1, *Order vs. Efficiency in Algorithms.*

2. The transparency, T.A.24.1, *Order vs. Efficiency in Algorithms*, summarizes all the categories in this lesson. Note that both axes in this diagram are exponential in scale.

SUMMARY/ REVIEW: When designing solutions to programming problems, we are often concerned with finding the most efficient solutions regarding time and space. We will consider memory requirements at a later time. Speed issues are resolved based on the number of steps required by algorithms.

ASSIGNMENT: Worksheet W.A.24.1, *Order of Algorithms*