CS119 – Module 5: Order of Growth

Purpose: What if we have more than one algorithm to solve a problem and wish to know which one is more efficient. Computer scientists use an asymptotic approach to compare the efficiency of algorithms. This means that when comparing algorithms, we look at which algorithm is more rapidly taking longer as the problem size increases and we call this order of growth.

Knowledge: This module will help you become familiar with the following content knowledge:

- Determining the order of growth of a function

Activity: Perform the following tasks and answer the questions. You will be reporting your answers back to the class at our next class meeting.

To get a feel for this approach to determining efficiency we will try out two algorithms for sorting a deck of cards. Use index cards with numbers written on them and sort them with each of the algorithms for decks of size 4, 8, 16, and 32 cards and time your results.

Two sorting algorithms are given below - one for Selection Sort and the other is Merge Sort. Look carefully at these algorithms. Once you are convinced that you understand these two algorithms you will try them out yourself and time the results.

Use the following two sorting algorithms to complete the table below with all times measured in seconds. That means that a time of 1 minute and 13 seconds would be recorded as 73 seconds.

Card	Selection Sort	Merge Sort
4		
8		
16		
32		

Then use Excel to graph your results. You may do this by selecting the last two columns of the table and going to the Insert tab and selecting a line chart. Your chart should give you a nice view of how the time increases as the problem size gets larger.

Selection Sort
You will use three positions for stacks of cards: source stack, destination stack, and the discard stack.
Initially you should put all the cards, face down, on the source stack, with the other two positions empty. Now do the following steps repeatedly:

1. Take the top card off the source stack and put it face-up on the destination stack.
2. If that makes the source stack empty, you are done. The destination stack is in numerical order.
3. Otherwise, do the following steps repeatedly until the source stack is empty:
   (a) Take the card off the source stack and compare it with the top of the destination stack.
   (b) If the source card has a larger number,
   (c) Take the card on the top of the destination stack and put it face down on the discard stack.
   (d) Put the card you took from the source stack face up on the destination stack.
   (e) Otherwise, put the card from the source stack face down on the discard stack.
4. Slide the discard stack over into the source position, and start again with step 1.

Merge Sort
Lay out the cards face down in a long row. We will consider these to be the initial source "stacks" of cards, even though there is only one card per stack. The merge sorts works by progressively merging pairs of stacks so that there are fewer stacks but each is larger; at the end, there will be a single large stack of cards.

Repeat the following steps until there is a single stack of cards:

1. Merge the first two face-down stacks of cards using Merge algorithm given below.
2. As long as there are a least two face-down stacks, repeat the merging with the next two stacks.
3. Flip each face-up stack over.

Merging
You will have the two sorted stacks of cards to merge side by side, face down. You will be producing the result stack above the other two, face up.

Take the top card off of each source stack – one in your left hand and one in your right hand. Now do the following repeatedly, until all the cards are on the destination stack:

1. Compare the two cards you are holding.
2. Place the one with the larger number on it onto the destination stack, face-up.
3. With the hand you just emptied, pick up the next card from the corresponding source stack and go back to step 1. If there is no next card in the empty hand's stack because that stack is empty, put the other card you are holding on the destination stack face-up and continue flipping the rest of the cards over onto the destination stack.

Activity: With your group perform the following tasks and answer the questions. You will analyze the two sorting algorithms and see if the analysis matches your timed results. You will be reporting your answers back to the class in 30 minutes.

1. Take a look at the Selection sort algorithm. We will define a "pass" as once through steps 1 through 4. If we are sorting n cards, how many cards will be handled in the first pass?

2. How many cards will be handled in the second pass?

3. Why will the total number of cards handled be $n + (n-1) + (n-2) + ... + 1$?

4. Since $n + (n-1) + (n-2) + ... + 1 \leq n + n + n + ... + n$. Why is the number of cards handled no greater than n^2? We call this $O(n^2)$.

5. Take a look at the Merge sort algorithm. The first pass will be when we merge the n piles into $n/2$ piles. How many cards are handled in the first pass?

6. How many cards are handled on the second pass when we merge $n/2$ piles into $n/4$ piles?

7. Why will it take log n passes to merge the cards into one pile?

8. Why will we handle no more than $n\log n$ cards to sort with merge sort? We call this $O(n\log n)$.

9. If we sort n cards with Selection sort and it takes time t, how much time would we expect it to take if we sort $2n$ cards? Hint: The time does not double.

10. If we sort n cards with Merge sort and it takes time t, why would we expect the time to be just a little bit more than double for sorting $2n$ cards?

11. Does this analysis match your recorded times? What would we expect the graphs to look like?

Activity: With your group perform the following tasks and answer the questions. You will be reporting your answers back to the class in 15 minutes.

1. Roughly how many recursive calls will be needed to compute b^n with the following function? What is the order of growth in terms of the size of the exponent n?

```
power: Int -> Int -> Int
power b n = if n == 0
                    then 1
                    else b * (power b (n-1))
```

2. Roughly how many recursive calls will be needed to compute b^n with the following function? What is the order of growth in terms of the size of the exponent n?
Hint: Try the substitution model for 2^8 or 2^{16}.

```
power: Int -> Int -> Int
power b n = if n == 0
                    then 1
                    else if even n
                            then square(power b (n `div` 2))
                            else b * (power b (n-1))
```

3. Which algorithm is more efficient?

Complete the following assignments to be submitted for grading. Each should be done individually but you can consult with a classmate to discuss your strategies.
You will need to copy and examine the functions in the file `BigO.hs` contained in the `lab5` directory.

Assignment 1:
Take a look at the mysterious function `f1` in the file. Try it out with the following input:

```
> f1 (sent "3 6 7 10 20 23") 10

> f1 (sent "3 6 7 10 20 23") 11
```

Determine what this function does. You might find it helpful to use the substitution model to figure this out. Also, assuming that the length of the sentence is n, tell me what the order of growth would be for this function.

Criteria for Success: You have determined the purpose and order of growth for this function.

Assignment 2:
Take a look at the mysterious function `f2` in the file. Try it out with the following input:

```
> f2 (sent "3 6 7 10 20 23") 10

> f2 (sent "3 6 7 10 20 23") 11
```

Determine what this function does. You might find it helpful to use the substitution model to figure this out. Also, assuming that the length of the sentence is n, tell me what the order of growth would be for this function.

Criteria for Success: You have determined the purpose and order of growth for this function.

Assignment 3:
Take a look at the mysterious function `f3` in the file. Try it out with the following input:

```
> f3 (sent "3 6 7 10 20 23") (sent "10 6 23")

> f3 (sent "3 6 7 10 20 23") (sent "10 6 11 23")
```

This function uses `f1` (perhaps several times). What is the maximum number of times that `f1` will be called. If the two sentences are roughly the same size, n, what would be the order of growth described in terms of n of `f3` for its worst case given what we know about the order of growth of `f1`?

Criteria for Success: You have explained how you determined the order of growth of `f3`.

Assignment 4:
Take a look at the mysterious function `f4` in the file. Try it out with the following input:

```
> f4 (sent "3 6 7 10 20 23") (sent "10 6 23")

> f4 (sent "3 6 7 10 20 23") (sent "10 6 11 23")
```

This function uses `f2` (perhaps several times). What is the maximum number of times that `f2` will be called. If the two sentences are roughly the same size, n, what would be the order of growth described in terms of n of `f4` for its worst case given what we know about the order of growth of `f2`?

Criteria for Success: You have explained how you determined the order of growth of `f4`.

Submit your written answers in Canvas for grading.