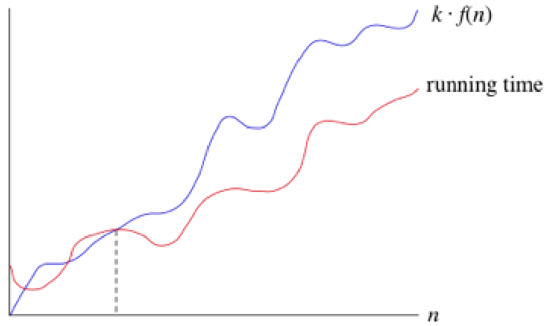
Algorithm Efficiency

A – Theory part

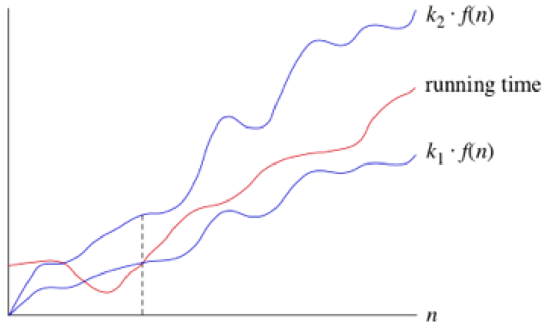
A.1. Which asymptotic notation is demonstrated in each of the following figures?

Note that the red line denotes the actual running time of the algorithm being considered, while the blue line shows some asymptotic bound.

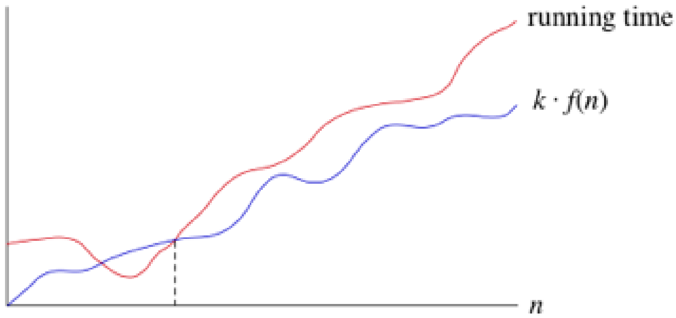
a.



b.



c.



A.2. Using Big O notation, indicate the time requirement of each of the following tasks in the worst case. Describe any assumptions that you make.

a. After arriving at a party, you shake hands with each person there.
b. Each person in a room shakes hands with everyone else in the room.
c. You climb a flight of stairs.
d. After entering an elevator, you press a button to choose a floor.
e. You ride the escalator from the ground floor up to the nth floor.
f. You read a book twice.

A.3. List the following growth-rate functions in order of growth.

- $\log_2(n)$
- $n\log_2(n)$
- n
- n^2
- 2^n
- n^3
- 3^n
- $n!$
- 1

A.4. Describe the running time of the following pseudocode in Big-O notation in terms of the variable n. Assume all variables used have been declared.

<table>
<tr><td></td><td><pre><code>int foo(int k) {
   int cost;
   for (int i = 0; i < k; ++i)
      cost = cost + (i * k);
   return cost;
}</code></pre></td></tr>
<tr><td>a.</td><td><pre><code>answ = foo(n);</code></pre></td></tr>
<tr><td>b.</td><td><pre><code>int sum;
for (int i = 0; i < n; ++i) {
   if (n < 1000)
      sum++
   else
      sum += foo(n);
}</code></pre></td></tr>
<tr><td>c.</td><td><pre><code>for (int i = 0; i < n + 100; ++i) {
   for (int j = 0; j < i * n ; ++j)
      sum = sum + j;
   for (int k = 0; k < n + n + n; ++k)
      c[k] = c[k] + sum;
}</code></pre></td></tr>
<tr><td>d.</td><td><pre><code>for (int j = 4; j < n; j = j + 2) {
   val = 0;
   for (int i = 0; i < j; ++i) {
      val = val + i * j;
      for (int k = 0; k < n; ++k)
         val++;
   }
}</code></pre></td></tr>
<tr><td>e.</td><td><pre><code>for (int i = 0; i < n * 1000; ++i) {
   sum = (sum * sum)/(n * i);
   for (int j = 0; j < i; ++j)
      sum += j * i;
}</code></pre></td></tr>
</table>

A.5. Assume that each of the following expressions has a running time of T(n) and the input size is n. Specify the highest-order operand in the expression and the corresponding Big-O.

a. $5 + 0.001n^3 + 0.025n$
b. $500n + 100n^{1.5} + 50n\log_{10}n$
c. $100n + 0.01n^2$
d. $2n + n^{0.5} + 0.5n^{1.25}$
e. $0.3n + 5n^{1.5} + 2.5n^{1.75}$
f. $0.01n + 100n^2$
g. $n\log_3 n + n\log_2 n$
h. $0.01n^2\log_2 n + n(\log_2 n)^2$
i. $2\log_2 n + 2\log_5 n$

A.6. Prove that if $f_1(n)$ is $O(g_1(n))$ and $f_2(n)$ is $O(g_2(n))$ then

a. $f_1(n) + f_1(n)$ is $O(\max(g_1(n), g_2(n)))$
b. $f_1(n) \times f_2(n)$ is $O(g_1(n) \times g_2(n))$

A.7. Show that

a. $f(n) = 2n^2 - n + 30$ is $O(n^2)$

b. $f(n) = (3n + 2) \log_2(n^2 + 5)$ is $O(n\log_2 n)$

c. $f(n) = (n^2 + 4\log_2 n) / (n + 1)$ is $O(n)$

A.8. Answer the following questions regarding algorithm efficiency.

a. Why is it essential to study algorithms and evaluate the algorithm efficiency?

b. In real life applications, what is more important than performance? Explain your answer.

A.9. Explain why the following statement, *"The running time of Algorithm A is at least $O(n^2)$."*, is meaningless.

A.10. Consider the following function:

```
int mystery(int n) {
   int answer;
   if (n > 0) {
      answer = ( mystery(n - 2) + 3 * mystery(n / 2) + 5 );
      return answer;
   }
   else
      return 1;
}
```

Write down the complete recurrence relation, T(n), for the running time of mystery(n). Be sure you include a base case T(0). You do not have to solve this relation, just write it down.

A.11. For each of the following statements, identify whether it is correct or not.

a. $f(n) = 2^{n+1}$ is $O(2^n)$

b. $f(n) = 2^{2n}$ is $O(2^n)$

A.12. Give one problem that can be solved using recursion (not the ones that were discussed in class). Write your recursive solution for that problem. Furthermore, rewrite the solution using iteration.

B – Coding part

B.1. Write a program to receive a positive integer N and calculate the N^{th} partial sum of the following series: $2 + 2^2 + 2^3 + \ldots + 2^N$

You also need to write the following functions for the program

a. POWER to calculate 2^N
b. SUMPOWER to calculate the given series by using the function POWER
c. SUMPOWER2 to calculate the given series by NOT using the function POWER

Test Data:

Input a positive integer: 3

Expected Output:

The sum of series is 14

Which one runs faster, SUMPOWER or SUMPOWER2? Explain your choice.

B.2. Write a program to receive positive integer N ($N \geq 3$) and prints out N first elements of the Fibonacci sequence, in which the first two elements F_0 and F_1 are always 1.

Test Data:

Input a number: 8

Expected Output:

The Fibonacci sequence is 1 1 2 3 5 8 13 21

Implement the above problem either recursively or non-recursively

B.3. Write a program to receive a positive integer N ($N \geq 3$) and prints out the factorial of N.

Test Data:

Input a number: 5

Expected Output:

The factorial is 120

Implement the above problem either recursively or non-recursively

B.4. Write a program to receive an array of N positive integers and find the subsequence with largest sum, using either of the following approaches

a. Brute force with complexity $O(N^3)$
b. An improved approach with complexity $O(N^2)$
c. Dynamic programming with complexity $O(N)$

Test Data:

Input the number of elements in the array: 6

Input an array: -2, 11, -4, 13, -5, -2

Expected Output: The subsequence with largest sum is 11, -4, 13

For each of the following problems, you will need an implementation of the ADT bag whose method **`remove`** removes a random entry instead of a specific one.

B.5. Suppose that you have several numbered billiard balls on a pool table. At each step you remove a billiard ball from the table. If the ball removed is numbered N, you replace it with N balls whose number is N / 2, where the division is truncated to an integer. For example, if you remove the 5 ball, you replace it with five 2 balls.

a. Using Big O notation, predict the time requirement for this algorithm when initially the pool table contains only the N ball.
b. Write a program that simulates this process. Use a bag of positive integers to represent the balls on the pool table. Time the actual execution of the program for various values of N and plot its performance as a function of N. Compare your results with your predicted time requirements.
c. Repeat the previous project, but instead replace the N ball with N balls randomly numbered less than N.

B.6. In mythology, the Hydra was a monster with many heads. Every time the hero chopped off a head, two smaller heads would grow in its place. Fortunately for the hero, if the head were small enough, he could chop it off without two more growing in its place. To kill the Hydra, all our hero needed to do was to chop off all the heads.

Write a program that simulates the Hydra. Instead of heads, we will use strings. A bag of strings, then, represents the Hydra. Every time you remove a string from the bag, delete the first letter of the string and put two copies of the remaining string back into the bag. For example, if you remove HYDRA, you add two copies of YDRA to the bag. If you remove a one-letter word, you add nothing to the bag. To begin, read one word from the keyboard and place it into an empty bag. The Hydra dies when the bag becomes empty.

Using Big O notation, predict the time requirement for this algorithm in terms of the number N of characters in the initial string. Then time the actual execution of the program for various values of n and plot its performance as a function of N.