

CS 310  
Assignment 0904

Bill Jin

**Problem 1** — Use the definitions to prove or disprove the statement  $2n^2 + 3 \in O(n^3)$ , and illustrate this graphically.

*Answer:* The definition requires us to find  $c$  and  $n_0$  so that  $2n^2 + 3 \leq cn^3$  when  $n \geq n_0$ . Based on the definition  $n$  and  $c$  are both non-negative constants. Since we have:

$$2n^2 + 3 \leq cn^3$$

$$\frac{2n^2 + 3}{n^3} \leq c$$

And raise the degree of all other terms as the highest one including constants with  $n$ .

$$\frac{2n^2 + 3}{n^3} \leq \frac{2n^3 + 3n^3}{n^3} \text{ (for all } n_0 \geq 1)$$

Obviously, the right-hand will be definitely higher or equal to the left-side for all  $n_0 \geq 1$ . Then, cancel all the terms and end with a constant:

$$\frac{2n^2 + 3}{n^3} \leq 5 \text{ (for all } n_0 \geq 1)$$

$$c = 5 \text{ (for all } n_0 \geq 1)$$

Therefore, we conclude that  $2n^2 + 3 \in O(n^3)$  is true since we have  $2n^2 + 3 \leq 5n^3$ .

This is graphically illustrated by the following plot that shows  $2n^2 + 3$  along with the standard function  $n^3$  scaled by the constant coefficient  $c = 5$ , for all  $n \geq 1$ .

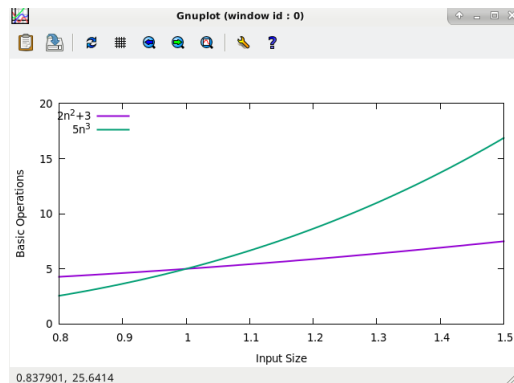


Figure 1: The graph shows  $y = 2n^2 + 3$  and  $y = 5n^3$

**Problem 2** — Either prove the following assertion using the definitions or disprove it with a specific counterexample:

$$\text{if } T(n) \in O(S(n)) \text{ then } S(n) \in \Omega(T(n))$$

*Answer:* If  $T(n) \in O(S(n))$ , then there is a constant  $c$  which is great and equal to zero ( $c \geq 0$ ), and a constant  $n_0$  which that  $n$  is great or equal to  $n_0$  ( $n \geq n_0$ ), where we can conclude:

$$T(n) \leq c \times S(n)$$

We can divide both sides of the expression by the constant  $c$ , then we have:

$$\frac{1}{c} \times T(n) \leq S(n) \text{ (for } \frac{1}{c} > 0 \text{ and } n \geq n_0.)$$

Therefore, there is a constant  $\frac{1}{c}$ . which can be replaced by constant  $k$ . Then we can conclude:

$$S(n) \geq k \times T(n)$$

for  $k > 0$  and  $n \geq n_0$ , which is exact definition of  $S(n) \in \Omega(T(n))$ .

**Problem 3** — For the following algorithm, explain what it computes, state what the input size for analysis is, state what basic operations should be counted for analyzing it, state exactly how many operations are executed as a function of the input size, and state the efficiency class to which it belongs.

```

1 void foo(vector<unsigned>& array)
2 {
3     size_t n = array.size();
4     for (size_t pass_idx = 0; pass_idx < n - 1; pass_idx++)
5     {
6         size_t min_position = pass_idx;
7         for (size_t compare_idx = pass_idx + 1; compare_idx < n;
8             compare_idx++)
9         {
10            if (array.at(compare_idx) < array.at(min_position))
11            {
12                min_position = compare_idx;
13            }
14        }
15
16        if (min_position != pass_idx)
17        {
18            swap(array.at(pass_idx), array.at(min_position));
19        }
20    }
21 }
```

*Answer:* This algorithm is used to sort the elements in the vector from minimum to maximum. Consider line 3, which defines  $n$ . This allows us to see that the input size is the size of the vector.

For the worst cases (Big-O), the operations that are counted are:

- the assignment on line 3, we count as 1 operation,
- the for loop condition on line 4, we count as  $2n$  operations,

- the `min_position` assignment on line 5, we count as  $(n-1)$  operations,
- the second for loop condition on line 7 and 8 will be counted as  $(n)+(n-1)+(n-2)..+2$  number of times, which is  $\frac{(n+2)(n-1)}{2}$ , and have two operations each time, so we count  $(n+2)(n-1)$  number of operation,
- the if condition on line which is comparison on line 10, we count as  $(n-1)+(n-2)+...1$ , which is  $\frac{n^2-n}{2}$ ,
- the assignment of `min_position` inside the if statement on line 12, we count as  $(n-1)+(n-2)+...1$ , which is also  $\frac{n^2-n}{2}$ ,
- the `min_position` comparison in if condition on line 16, we count as  $(n-1)$  operations,
- and the swap function on line 18, we count as  $2(n-1)$ ,

After sum up all, the number of times of all operations that are executed is:

$$2n^2 + 6n - 5$$

For the best cases (Big- $\Omega$ ), the operations that are counted are:

- the assignment on line 3, we count as 1 operation,
- the for loop condition on line 4, we count as  $2n$  operations,
- the `min_position` assignment on line 5, we count as  $(n-1)$  operations,
- the second for loop condition on line 7 and 8 will be counted as  $(n)+(n-1)+(n-2)..+2$  number of times, which is  $\frac{(n+2)(n-1)}{2}$ , and have two operations each time, so we count  $(n+2)(n-1)$  number of operation
- the if condition on line which is comparison on line 10, we count as  $(n-1)+(n-2) +...1$ , which is  $\frac{n^2-n}{2}$ ,
- the assignment of `min_position` inside the if statement on line 12, we count as 0 based on best case.
- the `min_position` comparison in if condition on line 16, we count as  $(n-1)$  operations,
- and the swap function on line 18, we count as 0 times, since this is best case.

After sum up all, the number of times of all operations that are executed is:

$$n^2 + 5n + \frac{n^2 - n}{2} - 1$$

Since there are two if conditions which could cause the best and the worst cases. Therefore, we can find out that this algorithm belongs to Big-Oh and Big-Omega efficiency class.

$$T(n) \in O(n^2) \text{ for } c = 4, \text{ all } n \geq 0$$

$$T(n) \in \Omega(n^2) \text{ for } c = 1, \text{ all } n \geq 0$$

or equivalently,

$$T(n) \in \Theta(n^2)$$

**Problem 4** — Write a C++ program that implements the algorithm in problem 3, counts the number of basic operations, and outputs the input size and the count of basic operations to the cerr stream.

*Answer:* See the submitted file HM.cpp

**Problem 5** — Run your program from problem 4 many times with many different inputs and capture the results. Using the output of the, create a plot of input size vs. basic operations, along with one or more standard functions properly scaled, to illustrate the analysis you obtained in Problem 3.

*Answer:* When the program is run with the command

```
for n in `seq 10 10 1000`
do
    ./program $n > /dev/null
    ./program $n > /dev/null
done 2> results.dat
```

and the resulting data file is plotted, we get the following. Also plotted on the same axes are the scaled standard functions  $3n^2$  and  $n^2$  which illustrate that the algorithm is big-theta.

