# Computer Science 1FC3

Lab 5 – Algorithm Analysis

Author – Paul Vrbik – vrbikp@mcmaster.ca

The purpose of this lab is to implement basic procedures into maple in order to analyze their complexity. It is broken into two parts, ALGORITHMS, and COMPLEXITY.

---

**ALGORITHMS**

An *algorithm* is finite sequence of unambiguous instructions that terminates. For example, long division is a type of algorithm.

```
LONG DIVISION

      To calculate x ÷ y = A remainder B do the following:

      (1)   Set A and B to 0
      (2)   While x<=y, increase A by 1 and decrease y by x
      (3)   If x>y then set B to y
      (4)   finish
```

To implement long division into maple we may do the following:

```
(01)        ]>long_division := proc(x,y)
(02)              local A,B,Y;
(03)              A:=0;
(04)              B:=0;
(05)              Y:=y;
(06)              while (x<=Y) do
(07)                    A:=A+1;
(08)                    Y:=Y-x;
(09)              end do;
(10)              B:=Y;
(11)              print(A,remainder,B);
(12)        end proc:

            [>long_division(7,15);
                                    3, remainder, 0
```

Lets look at each line carefully:

| | |
|---|---|
| `(01)` | Creates a procedure called `long_division` which accepts two inputs `x` and `y`. |
| `(02)` | Reserves the variables `A`, `B`, and `C` for use in the procedure. |
| `(03)-(04)` | Gives an initial value to `A` and `B`. |
| `(05)` | Since maple does not allow us to modify the value of `y` we create a temporary variable `Y` which we can manipulate. |
| `(06)` | Starts a *loop*, that is, everything that is contained within the loop will be repeated until `(x<=y)` |
| `(07)-(08)` | Incrementally increases `A` by 1 and decreases `B` by `x`. |
| `(09)` | Marks the end of the while loop. |

(10) Sets `B` to `Y`.
(11) Prints the results
(12) Marks the end of the procedure.

Now let us try to program a more difficult function that sorts a given list of integers. The description of bubble sort is given on page 126 of your textbook (Rosen).

**Bubble Sort**

```
bubble_sort:=proc(B,n)
local i,j,A,temp;
A:=B;
for i from 1 to (n-1) do
     for j from 1 to (n-i) do
          if (A[j]>A[j+1]) then
               temp:=A[j+1];
               A[j+1]:=A[j];
               A[j]:=temp;
          end if;
     end do;
end do;
print(A);
end proc:



]>bubble_sort([1,3,4,1,3],5);
```

*[1,1,3,3,4]*

We first note that we are using a `for` loop instead of a `while` loop as we did with division. The main difference is that a `for` loop will repeat something a set amount of times. For instance,

```
for i from 1 to 10 do
    command();
end do;
```

will execute `command();` ten times.

**PROBLEM SET 1**

Question 1:

How many times will `command();` execute in the following code?

(a)

```
for i from 1 to 10 do
    for j from 1 to 20 do
          command();
    end do;
end do;
```

(b)

```
for i from 1 to 10 do
    for j from 1 to i do
        command();
    end do;
end do;
```

(c)

```
i:=0;
j:=0;
while (i<11) do
    i:=i+1;
    while (j<21) do
        j:=j+1;
        command();
    end do;
end do;
```

(d)

```
for i from 1 to x do
    for j from 1 to y do
        command();
        for j from 1 to i;
            command();
        end do;
    end do;
    command();
end do;
```

Question 2:

Write an equivalent statement in Maple using the while command.

(a)

```
for x from 1 to 100 do
    command();
end do;
```

(b)

```
for x from 1 to 100 do
    for y from 1 to x do
        command();
    end do;
end do;
```

Question 3:

Here is an algorithm to find the number `N` in a given list of integers, say A.

```
(1)          set x to 1
(2)          look at the xth element of the list, if its N then print x
             and stop looking
(3)          if x is the end of the list then print 0 and stop
(4)          increase x by 1 and go to step (2)
```

Implement this algorithm into maple and test it.

## COMPLEXITY

Since computers have different processing capabilities, it is more meaningful to represent the speed of an algorithm by the number of times a command is executed rather then the time it takes to complete the algorithm. This representation is called complexity. The complexity of an algorithm is a function that relates the number of executions in a procedure to the loops that govern these executions.

Consider the code:

```
]>procedure1:=proc(n)
      local i;
      for i from 1 to n do
            command();
      end do;
end proc;
```

The number of times `command` is executed is directly related to the size of `n`. A function modeling this relation would be `f(n) = n`, where `f(n)` represents the number of times `command` is evoked. If a machine took two minutes to execute `command` it would take `(2 minutes)*f(n)` to run the procedure.

In complexity we say that `proc1` is `O(n)`, (big-oh of n), or that the running time is governed by a linear relation.

## DETERMING COMPLEXITY OF MORE COMPLICATED PROGRAMS

The following examples will further demonstrate an algorithms complexity.

<u>Example 1:</u>

```
]>procedure2:=proc2(n){
      local i,j;
      for i from 1 to n
            for j from 1 to n
                  command();                n^2
            end do;
      end do;

      for i from 1 to 10000000
            command();                      10000000
            command();
      end do;
end proc;
```

`f(n)=n^2+10000000` that corresponds to `O(n^2)`.

Example 2:

```
procedure3:=proc(n)
      local i,j;
      for i from 1 to n
            command();
            command();                 2n
            for j from 1 to n                          2n + n^2
                  command();           n^2
            end do;
      end do;

      procedure2(n);         n^2
end proc;
```

`f(n)=2n+2n^2` that corresponds to `O(n^2)`.

**WORST CASE SCENARIO**

Realistically we do not have `command();` laid out in plain sight for us. Let us consider the long division algorithm from the section before, what is its complexity?

Well first let us fix `y`, the number that we are dividing into, what is the worst-case scenario, or the scenario where we will have to do the most amount of computation? The answer to this is when `x` is equal to one, if this is the case we will have to loop `y` times. From this we can conclude that at worst we have to carry out `y` computations which corresponds to `O(y)`.

When there are many possible scenarios to consider we will always pick the worse case. This guarantees that the big-oh bound we choose will always be sufficient.

**COMPLEXITY OF BUBBLE SORT**

When the `ith` pass begins, the `(i-1)` largest elements are guaranteed to be in the correct positions. During this pass, `(n-i)` comparisons are used. Consequently, the total number of comparisons used by the bubble sort to order a list of `n` elements is:

$$(n-1)+(n-2)+\dots+2+1=\sum_{n=1}^{n-1} n=\frac{(n-1)(n)}{2}$$

So we conclude that the complexity of bubble sort is: $O\left(\frac{(n-1)(n)}{2}\right)=O\left(n^2\right)$.

**PROBLEM SET 2**

Question 1:

What are the complexities of the loops given in Question 1 from problem set 1.

Question 2:

Give the complexity of the algorithm outlines in Question 3 from problem set 1. As a point of interest this algorithm is called “The Linear Search Algorithm”, why do you think this is.

**PROBLEM SET 3 (STUDY)**

The following is a pseudo-code description of the Binary Search Algorithm.

```
procedure binary search (x : list of integers in increasing order)
      i=1
      j=n
      while i<j
            m=(i+j)/2 rounded down to the closer integer
            if x > a[m] then i=m+1
            else j=m
            end if
      end while
      if x=a[i] then location=i
      else location = 0
      end if
end procedure binary search
```

Question 1:

Implement this algorithm into Maple.

Question 2:

Determine how the algorithm works by printing out the list at various places in the procedure.

Question 3:

Determine this procedures complexity.