

Computer Science 1FC3

Lab 7 – Algorithm Analysis

Author – Paul Vrbik

The purpose of this lab is to implement basic procedures into Maple in order to analyze their complexity.

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 than 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 $(2 \text{ minutes}) * f(n)$ to run the procedure.

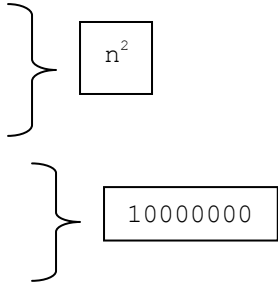
In complexity we say that `proc1` is $O(n)$, (big-oh of n), or that the running time is governed by a linear relation.

DETERMING COMPLEXITY OF MORE COMPLICATED PROGRAMS

The following examples will further demonstrate an algorithms complexity.

Example 1:

```
]>procedure2:=proc2(n) {
  local i,j;
  for i from 1 to n
    for j from 1 to n
      command();
    end do;
  end do;
  for i from 1 to 5000000
    command();
    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();
    for j from 1 to n
      command();
    end do;
  end do;
  procedure2(n);
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 tutorial before, what is its complexity?

We'll 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 correspond to $O(y)$.

When there are many possible scenarios to consider we will always pick the worse case. The worst case is the largest number of operations needed to solve the given problem using a particular algorithm with an input of specified size. This guarantees that the big-oh bound we choose will always be sufficient.

COMPLEXITY OF BUBBLE SORT

When the i th 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: $O\left(\frac{(n - 1) (n)}{2}\right) = O(n^2)$.

PROBLEMS

Question 1:

What are the complexities of the following loops.

(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;
```

(e)

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

(f)

```
i:=0;
j:=2;
while (i<j) do
  i:=i+1;
  while (j>i) do
    j:=j+1;
    command();
  end do;
end do;
```

Question 2:

Give the complexity of the algorithm below. As a point of interest, this algorithm is called “The Linear Search Algorithm”, why do you think this is?

- (1) set x to 1
- (2) look at the x^{th} element of the list, if it's 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)

EXTRA PROBLEMS (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 procedure's complexity.