Computer Science 1FC3

Lab 5 – Algorithms

Original Author – Paul Vrbik Modified by Jessica Cao and Don Vo

The purpose of this lab is to implement basic procedures into Maple.

ALGORITHMS

An *algorithm* is finite sequence of unambiguous instructions that terminates.

For example, long division is a type of algorithm.

LONG DIVISION

To calculate $y \div x = A$ 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)        ]>longdivision := 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:

            [>longdivision(5,15);
                                3, remainder, 0
```

Let's look at each line carefully:

(01)	Creates a procedure called `longdivision` 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). If you are using the 6th edition, it is page 173.

Bubble Sort

```
(01)        bubblesort:=proc(B,n)
(02)        local i,j,A,temp;
(03)        A:=B;
(04)        for i from 1 to (n-1) do
(05)              for j from 1 to (n-i) do
(06)                    if (A[j]>A[j+1]) then
(07)                          temp:=A[j+1];
(08)                          A[j+1]:=A[j];
(09)                          A[j]:=temp;
(10)                    end if;
(11)              end do;
(12)        end do;
(13)        print(A);
(14)        end proc:



            ]>bubblesort([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.

Insertion sort is another simple sorting algorithm:

Insertion Sort

```
(01)        insertionsort:=proc(B,n)
(02)        local i,j,A,minIndex,temp;
(03)        A:=B;
(04)        for i from 1 to (n-1) do
(05)              minIndex:=i;
(06)              for j from (i+1) to (n-1) do
(07)                    if (A[j]<A[minIndex]) then
(08)                          minIndex:=j;
(09)                    end if;
```

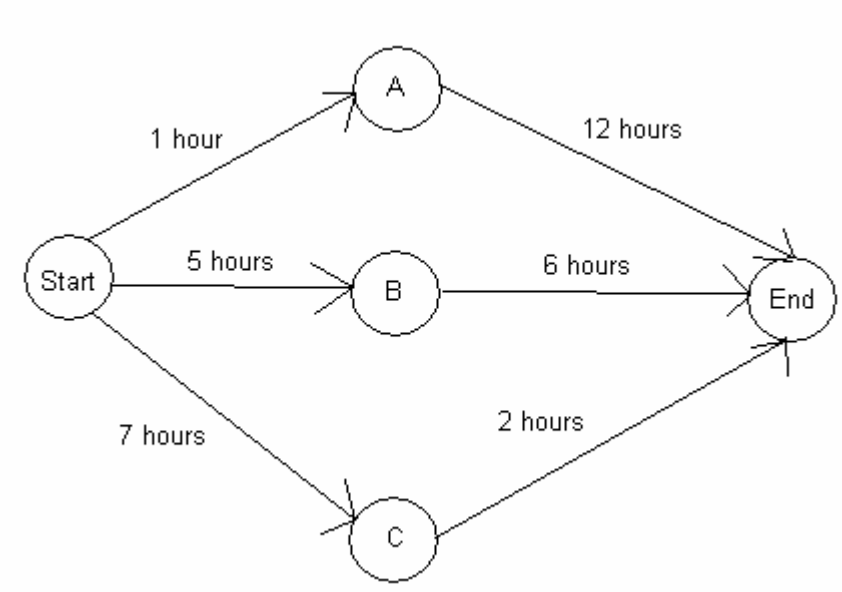
```
(10)                end do;
(11)                if (minIndex!=i) then
(12)                      temp:=A[i];
(13)                      A[i]:=A[minIndex];
(14)                      A[minIndex]:=temp;
(15)                end if;
(16)          end do;
(17)          print(A);
(18)          end proc:



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

GREEDY ALGORITHMS

Many algorithms that you may study in this course are designed to solve optimization problems. The Department of Computing and Software here at McMaster does a lot of work in this field in the Advanced Optimization Laboratory (you can check out them out here http://optlab.mcmaster.ca/).

The goal of optimization is to find a solution to a problem that either minimizes or maximizes the value of a certain parameter. For example, it may be minimizing the time of construction for a building or maximizing profit in a manufacturing business.

Often when performing an algorithm, there are opportunities to make a choice. A greedy algorithm is one that always chooses the most optimal option at that given moment (the local optimum). While this may result in an optimal solution (a global optimum), it could also not.



If our algorithm is one that chooses the shortest path from the current point to the next, it would first go from Start to A as it only takes 1 hour. From A to End, there is no choice but to take the 12 hour path. However, if one were to explore all the options, it can be seen that taking C is the best option. Greedy algorithms may not always work but when they do, they are usually the fasting way to solve the problem.

PRACTICE PROBLEMS

Question 1:

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 2:

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 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)
```

Implement this algorithm into Maple and test it.

Question 3:

What if the list A in Question 3 were sorted (e.g. numbers are listed in increasing order)? Is there a more efficient algorithm to find N? Write the psuedocode for this algorithm and implement it in Maple.

Question 4:

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

(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;
```