Tutorial 2

Recursion

Lecturer: Dr Andrew Hines *TA: Esri Ni*

Question 1. Give a definition for *recursion* including explanations of the terms *base case* and *stop condition*.

Recursion is a way of decomposing problems into smaller, simpler sub-tasks that are similar to the original.

- Thus, each sub-task can be solved by applying a similar technique.
- The whole problem is solved by combining the solutions to the smaller problems.
- Requires a **base case** (a case *simple enough to solve without recursion*).
- Requires a **stop condition** to end the recursion.

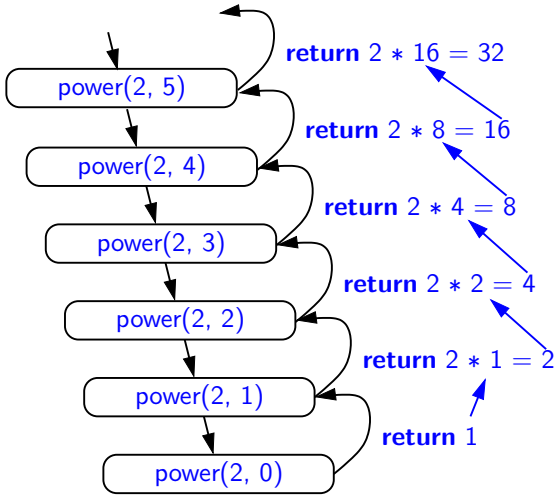
Question 2. Describe a recursive algorithm for finding the maximum element in a sequence, S, of n elements. What is your running time?

If the sequence has 1 element, that is the maximum. Otherwise, consider the bigger of the first element or the maximum of the other $n-1$ elements. The running time is $\mathcal{O}(n)$.

Question 3. Draw the recursion trace for the computation of power(2,5), using the traditional function below:

```
def power(x, n):
    '''Compute the value x     n for integer n.'''
    if n == 0:
        return 1
    else:
        return x * power(x,  n 1 )
```

Hint: This is probably the first power algorithm you were taught



Question 4.
Write a short recursive function that finds the maximum value in an array *without using any loops*. There are multiple ways of doing it, but a good one (often used when scanning an array-like structure) consists in keeping track of the current element inspected (hence here the use of $index$). Each recursive call collects the maximum value from a subset of the array (from $index + 1$ till the end) and then the algorithm just needs to get the maximum between the current ($my\_array[index]$) value and the recursive call (which returns the maximum of the rest of the array) – and to return the maximum value in the array from $index$ on.

Hint: Python provides inbuilt function called `max` that will compare all input numbers and return the highest number (and other languages do too). It doesn't take lists as inputs. You could write pseudo code that makes use of an inbuilt max function or just code the comparisons yourself. Or try both ways.

Algorithm 1 $maximum(my\_array, index)$

Input: an array my_array (of integers) of size $n > 1$ and $index$ the current element inspected
Output: the maximum value in my_array

```
if ? then
    return ?
else
    return ?
end if
```

Below are two slightly different versions of the same algorithm.
In each case, we have a base case and a recursive case. Our base case stops the algorithm from running on and on. In this case, we stop when we reach the last index of the array (because there is nowhere else to go after this index!). We then return the element at this index of the array because, if this element is the last element, there are no elements after it to compare to, therefore this index must be the maximum at that point in time. For all indexes larger than that, we have to do a little more work. At each recursive call, we are looking to find the max of two elements:

1. The element at $index$ (which denotes the element currently being worked on).
2. The maximum element of the array from $index + 1$ onwards.

Lets take a simple example....
We will use Algorithm 2 for simplicity sake. Say we have an array with 3 elements $array = [2, 6, -1]$ and a starting index of 0. We call the function $maximum(array, 0)$. The index is not equal to the length of the array - 1 so we move to the $else$ statement which is the recursive case. We now need to find $max(2, maximum(array, 0 + 1))$. So before we compare these two, we enter the function from the beginning again because we have just called the function again. The index is now 1, this still doesn't meet our base case so we enter the recursive case again. Now we need the $max(6, maximum(array, 1 + 1))$. This means we have to enter the function again! This time we can see that we do meet the base case ($n - 1 = 2$) so we return the element at this index, which is -1. We can now work backwards. We now know $max(6, maximum(array, 1 + 1) = -1)$ which will evaluate to 6. We can then pass this back to the function call before it so we now know $max(2, maximum(array, 0 + 1) = 6)$ which will evaluate to 6.
So after all that now we know the maximum is 6!

Algorithm 2 $maximum(my\_array, index)$

Input: an array my_array (of integers) of size $n > 1$ and $index$ the current element inspected
Output: the maximum value in my_array

```
if index = n − 1 then
    return array[index]
else
    return max(array[index], maximum(array, index + 1))
end if
```

In Algorithm 3, we are performing the comparison ourselves by assigning the maximum element returned recursively by the function to a variable and then comparing it against the element whose index we are currently working on. In Algorithm 2 we are simplifying things a little by assuming that a built-in function $max()$ will find the maximum of these two elements.

Algorithm 3 $max(my\_array, index)$

Input: an array my_array (of integers) of size $n > 1$ and $index$ the current element inspected
Output: the maximum value in my_array

```
if index = n − 1 then
    return array[index]
else
    max ← maximum(array, index + 1)
    if array[index] > max then
        return array[index]
    else
        return max
    end if
end if
```

Question 5.
We want to use recursion to reverse all the elements in an array. Basically each recursive call will work on a single element, at index $index$, and will swap it with the element at $size\ of\ the\ array - index$. Like the previous exercise, it is good to keep track of the current index (and hence have it in the parameters of the function). The base case consists in checking whether we've reached the middle of the array, $\frac{size\ of\ the\ array}{2}$. Note that contrary to the previous exercise there is no `return` here, just recursive calls!

Algorithm 4 $reverse(my\_array, index)$

Input: an array my_array and $index$ the current element inspected
Output: The elements in my_array are reversed
  if ? **then**
    ?
  else
    ?
  end if

Here, we are swapping the first element in the array with the last element at each recursive step. At each step, we are moving forward an index toward the end ($index+1$) and moving one index toward the beginning ($last\_index - index$: The last index is the size of the array $n-1$). We use a temporary variable to store the original value of the first index so this index can now hold the value of the last element of the array. We then pass the value of the temporary variable to the last index. Note that we only need to go up as far as half the array as, at that point, all of the swaps will have been made. Note also that there is no need for a return as we are simply changing the state of the array object. The base case here is also not explicit in this case (i.e. no if $base\_case$, else $recursive\_step$) as the recursion will stop once the index is less than or equal to the size of the array, divided by 2.

Algorithm 5 $reverse(my\_array, index)$

Input: an array my_array and $index$ the current element inspected (n denotes the length of the array).
Output: The elements in my_array are reversed
  if $index \leq (n-1)/2$ **then**
    $temp \leftarrow my\_array[index]$
    $my\_array[index] \leftarrow my\_array[(n-1) - index]$
    $my\_array[(n-1) - index] \leftarrow temp$
    $reverse(my\_array, index + 1)$
  end if