Design and Analysis of Algorithms — Practical Python – Review

R. S. Milton, Department of CSE, SSN College of Engineering

12 December 2019

1 Functions

1. Write a function `fibR(n)` to compute f_n, the Fibonacci number. Use the recursive definition:

$$f_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ f_{n-1} + f_{n-2} & \text{otherwise} \end{cases}$$

2. Write an iterative function `fibI(n)` to compute f_n, the nth Fibonacci number.

3. Write a memoized function `fibM(n)`. Every Fibonacci number it calculates, it stores in a global table.

2 Selection sort

1. Write a function `print_array(a, low, high)` to print the items of a subarray `a[low:high]` on the screen. Test it.

2. Write a function `read_array(a)` to read a list of integers from the keyboard into the subarray `a[0:n]` and return `n`, the number of integers read. Since the caller of `ready_array(a)` does not know in advance how many numbers would be read by `read_array(a)`, the caller should pass a sufficiently large `a`.

3. Write a function `minimum(a)` that finds the smallest number in an array `a`. Input is an array `a[0:n]` of `n` comparable items. Output should be the index of the smallest item `a[0:n]`. Test

the function interactively and from a `main()` function. Test it for several lists of numbers where each test should read a list of numbers from the keyboard.

4. Modify `minimum(a)` to `minimum(a, low, high)`. You are given an array `a[0:n]` of `n` comparable items. Define the function `minimum(a, low, high)` that returns the index of the smallest item in the subarray `a[low:high]`.

5. **Selection sort**: Selection sort is an algorithm for sorting an array of items, say `a[0:n]`. The idea of the algorithm is expressed below:

   ```
   bring the smallest item to 0
   bring the next smallest item to 1
   bring the next smallest item to 2
   ...
   bring the largest item to n-1
   ```

   which translates to

   ```
   swap minimum(a,0,n) with 0
   swap minimum(a,1,n) with 1
   swap minimum(a,2,n) with 2
   ...
   swap minimum(a,n-2,n) with n-2
   ```

   which uses `minimum(a, i, n)` to find the minimum of a subarray `a[i:n]`. Implement selection sort, using `minimum()` function. Note: remember that when a function changes the items of an array parameter, the changes are effected in the items of the actual array argument also.

   Test the function from a `main()` for several lists of numbers. Each test should read a list of numbers from stdin.

3 Insertion sort

1. The input to function `ordered_insert(a)` is an array `a[0:n]` of `n` integers. The precondition is: the items of the array are in non-decreasing (ascending) order except the last item (that at `n-1`) which may be out of order. The postcondition is that `a[0:n]` is sorted, that is, the out-of-order item has been brought to its right position. Use this iterative step: keep swapping the out-of-order item until it comes to the "right" position, that is, ordered with its neighbors.

2. Modify `ordered_insert(a)` to `ordered_insert(a, n)` that inserts `a[n-1]` in the right place in `a[0:n-1]`.

3. Solve problem 2; but use recursion instead of iteration.

4. Solve problem 2, but use a slighly different iterative step: back up `a[n-1]` in a variable. Then keep shifting the items of the array to the right until the correct position for the out-of-order item is found; then, in that position, insert the out-of-order item from its backup variable.

5. You are given as input an array `a[0:n]` of `n` numbers. The output should be sorted `a[0:n]`. Break the problem into `a[0:n-1]` and `a[n-1]`. Sort `a[0:n-1]` recursively. Then, insert `a[n-1]` in the right place in `a[0:n-1]` using `ordered_insert(a, n)`.

6. The input is an array of `N` numbers. The array is sorted in ascending order except for the first number (that at 0), which may be out of order. The output should be a sorted array of `N` original numbers which include the out-of-order number.

7. The input is an array of whole numbers. The numbers are stored only at positions 1, 2, 4, 8, 16, ...(powers of 2). We do not store anything in between. The elements of the array are sorted except the element at position 1. Rearrange the numbers so that the elements of the array are sorted.

4 Polish National Flag (PNF)

1. You are given an array of items `a[low:high]`, each of which is either positive or negative. Define a function `PNF(a, low, high)` that partitions the array into two subarrays `a[low:i]` and `a[i:high]` such that all the negative items of the array form `[low:i]`, and all the positive items form `[i:n]`. Test the function from `main()`. Use several lists of numbers for testing. (Note: We will use this algorithm for implementing `quicksort()`.)

2. Input is an array of items `a[low:high]`. Modify the function `PNF(a, low, high)` that selects `a[low]` as a pivot and partitions the array into two subarrays `a[low:i]` and `a[i:high]` such that all the items smaller than the pivot form `[low:i]`, and all the other items form `[i:n]`.