Ayman Al Zaatari
11 December 2014
CMPS 211 - Assignment 10 Solution

Assignment 10 Solution

Ex1:

procedure mult (n: positive integer, x: integer)
if n=0 then return 0
else return x + mult (n-1, x)
{output is nx}
O(n)

Ex2:

procedure summation (n: positive integer)
if n=1 then return 1
else return n + summation (n-1)
{output is the sum of the first n positive integers}

Basis step: for n=1, summation = 1
Inductive step: hypothesis: suppose that till n=k summation (n) = 1+2+3+4… +k
RTP it is true for k+1
summation (k+1) = summation (k) + k+1 = 1+2+3+4… +k+ (k+1)

Ex3:

procedure max ($a_1, a_2, a_3, \ldots, a_n$: integers)
if n=1 then return a_1

subSetmax := max ($a_1, a_2, a_3, \ldots, a_{n-1}$)
if $a_n \geq$ **subSetmax** then
    return a_n
else
    return **subSetmax**

{output is maximum of the set}

O(n)

Ex4:

procedure mode(a_1, a_2, ...a_n:integers, i: integer, n: integer, mode_location: integer)

```
    if i = 0
        return  a[mode_location]
    if countMode(a1, a2, ....an, i, n, 0) >= countMode(a1, a2, ....an, mode_location, n, 0)
        return mode (a, i-1, n, i)
    else
        return mode(a, i-1, n, mode_location)
```

procedure countMode (a1, a2, ...an:integers, i: integer, n: integer, count: integer)

```
    if n = 0
        return  count
    if an-1 = ai
        count := count + 1
    return  countMode(a1, a2, ....an, n-1, count)
```

Note:

- i is used traverse the index of the array a (index starts at 0)
- n is the length of the array a
- mode_location is used to specify the location of the mode
- "count" is used to count the occurrences of the elements in the array a

→ Example: **mode** ({1, 2, 10, 3, 3, 3, 1, 4, 5, 5}, 9, 10, 0)→ 3

Ex 5:

```
procedure multiply(x, y: nonnegative integers)
    if y = 0 then
        return 0
    else if y is even then
        return 2*multiply (x, y/2)
    else
        return 2*multiply (x, (y−1)/2) + x
```

it is proved by strong induction

basis step: multiply (x, 0) = 0 = 0x and multiply (x, 1) = multiply (x, 0) +x = x = 1x

inductive step: suppose that for $0 \leq y \leq k$ xy=multiply (x, y)

-if k is odd, k+1 is even

multiply (x, k+1) = 2*multiply(x, (k+1)/2) = 2 x(k+1)/2 = x(k+1) [since $0 \leq (k+1)/2 \leq k$]

-if k is even, k+1 is odd

multiply (x, k+1) = 2*multiply(x, (k)/2) = 2 xk/2 + x = xk+x = x(k+1) [since $0 \leq k/2 \leq k$]

then in both cases multiply(x, k+1) gives x*(k+1), then its correct

Ex6:

procedure power (a: real number, n: positive integer)

if n=0
    return a
else
    pow:= power(a, n-1)
    return pow * pow
{output a^{2^n}} O(n)

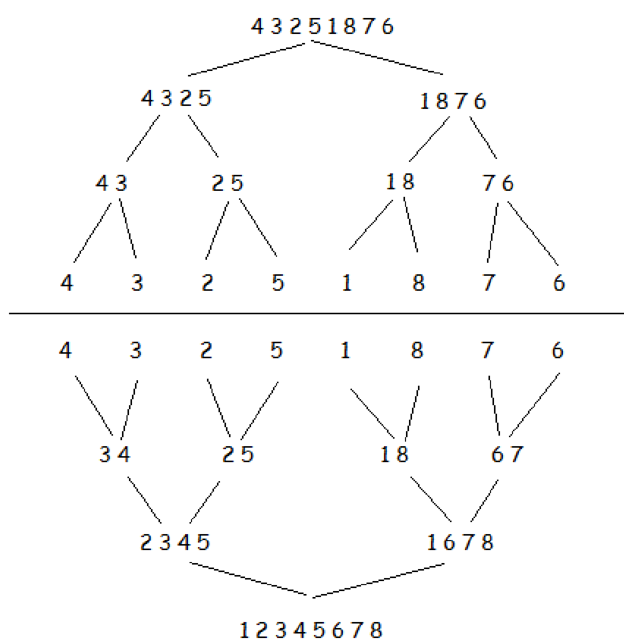
Ex7:

procedure a (n: non negative integer)

if n=0,
    return 1
else if n=1
    return 2
else if n=2
    return 3
else
    return a(n-1) + a(n-2) + a(n-3)

Ex 8:

the solution is given in the binary tree
in the upper part, we are just dividing the numbers into 2 equal lists where the difference between the number of values in these 2 lists does not exceed 1
in the lower part we sort the values by merging 2 lists at a time by removing smaller of first elemts of the 2 lists and putting them in the right of the new list until we have an empty list, we add all the values of the second list to the left of the values in the new list
since the steps are similar in the lower part, I'm going to show the last merge done



First List	Second List	Merged List	Comparison
2 3 4 5	1 6 7 8		1 < 2
2 3 4 5	6 7 8	1	2 < 6
3 4 5	6 7 8	1 2	3 < 6
4 5	6 7 8	1 2 3	4 < 6
5	6 7 8	1 2 3 4	5 < 6
	6 7 8	1 2 3 4 5	
		1 2 3 4 5 6 7 8	

Ex9:

We use strong induction on n, showing that the algorithm works correctly if n = 1, and that if it works correctly for n = 1 through n = k, then it also works correctly for n = k + 1. If n = 1, then the algorithm does nothing, which is correct, since a list with one element is already sorted. If n = k + 1, then the list is split into two lists, L1 and L2 . By the inductive hypothesis, mergesort correctly sorts L1. Now assume that L2 was split into two sublists, the first containing the elements until k and the second contains the (k+1)th element. We know the first sublist would also be correctly sorted using our algorithm given the induction hypothesis, and we know the second sublist which contains only the (k+1)th element is also sorted by definition. So it remains to only show that merge correctly merges two sorted lists into one. This is clear, since with each comparison, the smallest element in L1 ∪ L2 not yet put into L is put there