Midterm #1

Adrian Darian

1) a) $n \log n = O(n^2)$ **True**

b) $\log \log n = O(\log n)$ **True**

c) $\log^{50} n = O(n^{0.1})$ **True**

d) $4^n = O(2^n)$ **False**

e) $100^{100} = \Theta(1)$ **True**

f) if $f = O(g)$, then $g = \Omega(f)$ **True**

g) if $n^3 = \Omega(n^2)$ **True**

h) if worst case is $O(n^2)$, then $\Theta(n^2)$ for all n **False**

i) $100 + 200n + 300n^2 = \Theta(n^2)$ **True**

j) $\sum_{i=1}^{n} \log i = \Theta(n \log n)$ **True**

2) $A[1 \ldots 8] = \langle 7,4,2,9,4,3,1,6 \rangle$

```
Insertion sort (A)
    for j=2 to A.length
        key = A[j]
        i = j-1
        while i>0 and A[i] >key
            A[i+1] ⇐ A[i]
            i = i-1
        A[i+1] = key
```

7,4,2,9,4,3,1,6

4,7,2,9,4,3,1,6

4,2,7,9,4,3,1,6

2,4,7,9,4,3,1,6

3) a) 9,8,7,6,5,4

b) 4,5,6,7,8,9

4) Square-Matrix-Multiply-Recursive(A,B)

```
n = A.Rows
let C be a new nxn matrix
if n==1
    c11 = a11 · b11
else partition A,B, and C as in equations (4.9)
    C11 = SMMR(A11, B11) + SMMR(A12, B21)
    C12 = SMMR(A11, B12) + SMMR(A12, B22)
    C21 = SMMR(A21, B11) + SMMR(A21, B12)
    C22 = SMMR(A21, B12) + SMMR(A22, B22)
return C
```

$T(n) = 8T(\frac{n}{2}) + \Theta(n^2)$

$T(n) = \Theta(n^3)$

5) Merge-Sort(A,p,r)

if p<r

q = ⌊(p+r)/2⌋

Merge-Sort(A,p,q)

Merge-Sort(A,q+1,r)

Merge(A,p,q,r)

a) Line 3 $T(\frac{n}{2})$

b) Line 4 $T(\frac{n}{2})$

c) Line 5 $O(n)$

6) $50n+15 = O(n^2)$

$50n+15 \leq 50n^2+15n^2$ for $n \geq 1$

$50n+15 \leq 65n^2$ for $n \geq 1$

7) Heapsort(A)

max-heap(A)

for i = A.length downto 2

exchange A[1] with A[i]

A.heapsize = A.heapsize - 1

MAX-HEAPIFY(A,1)

8) invariant → when the first loop starts again the subarray A[0...i] will always be sorted

initialization → will always start with first element in array

maintenance → at every next iteration we expand the subarray. once we insert the element we assume everything to the left is smaller than that element

termination → the insertion sort algorithm terminates when i reaches the last element

9) Loop invariant → each iteration starts at the outer loop with subarray A[1...j-1] where A[1...n] where n=j-1 is sorted

10) base case → size of array = 0 or 1 return array

hypothesis → MS(A,p,q) → A[p,..q] and MS(A,q+1,r) → A[q+1...r]

step → recursize call merging new subarray over and over till A[1...size(A)]