Homework 2

Author: Ajit Aarya Kankipati (01211068)

Problem 4-1 D, F

Problem 4 - 1(D)

$$T(n) = 7T(n/3) + n^2$$

from equation,

$a = 7, \; b = 3$

$f(n) = \Theta(n^2)$

if $n = b^i$

$T(n) = aT(n/b) + f(n)$

$$T(n) = \begin{cases} \\ \Theta(1) \text{ if } n = 1 \end{cases}$$

This is Case-3

$T(n) = \Theta(f(n))$

$T(n) = \Theta(n^2)$

Problem 4 - 1(f)

$$T(n) = 2T(n/4) + \sqrt{n}$$

From the question,

$a = 2, \; b = 4, \; f(n) = \sqrt{n}$

This is Case-2 i.e.,

$T(n) = aT(n/b) + f(n)$ if $n = b^i$

then $T(n) = \Theta(n^{\log_b a} \log n)$

$n^{\log_4 2} = n^{1/2} = \sqrt{n}$

$T(n) = \Theta(\sqrt{n} \log n)$

Problem 4-3 H

Problem 4-3 (h)

Given $T(n) = T(n-1) + \log n$

$= T(n-2) + \log n + \log(n-1)$

$= \cdots$

$= T(1) + \sum_{K=2}^{n} \log(K)$

$= T(1) + \log\left(\prod_{K=2}^{n} K\right)$

$= T(1) + \log(n!)$ $\left(\because \log(n!) = \Theta(n \log n)\right)$

$= \Theta(1) + \Theta(n \log(n))$

$= \Theta(n \log(n))$

Exercise 6.2-5

```
MIN-HEAPIFY (A, i):
    while i ≤ heap-size[A]:
        l <- LEFT(i)
        r <- RIGHT(i)
        largest <- i
        if l ≤ heap-size[A] and A[l] > A[i]:
            then largest <- l
        if r ≤ heap-size[A] and A[r] > A[largest]:
            then largest <- r
        if largest ≠ i:
            then swap(A[i], A[largest])
                i = largest
        else break
```

Exercise 6.5-9

The simplest method is to repeatedly select the smallest of the top entries in each list. This requires k-1 comparisons for each element, for a total of O(k n).

As the hint suggests to use min-heap for k-way merging, the best method is to keep the smallest member from each list in a heap. so each element is supplemented with the index of the lists from which it originates. On the heap, we may use DeleteMinimum (minimum priority queue) method to discover and delete the smallest element, then insert the next member from the related list.
From the above method, we can notice that it takes O(k) to create the heap, and it takes O(log k) to DeleteMinimum (minimum priority queue) and O(log k) to insert the next member from the same list for each element. It takes O(k + n log k) = O in total (n log k).

Extra Credit Problems

6-3
a. Given the elements {9, 16, 3, 2, 4, 8, 5, 14, 12}, the 4x4 matrix will be

2	4	9	∞
3	8	16	∞
5	14	∞	∞
12	∞	∞	∞

c.
The smallest element is A[1, 1]. We save it so that we may return to it later and replace it with infinite. This destroys the young tableau property, which must be restored using a process similar to MAX-HEAPIFY.

We compare A[i, j] to each of its neighbors and swap it for the smallest. This retains the property for A[i, j], but simplifies the issue to A[i, j + 1] or A[i + 1, j]. When A[i, j] is smaller than its neighbors, we stop.

T(p)= T(p-1) + O(1) = T(p-2) + O(1) + O(1) =… = O(p)

d.

To insert a new element into a non-full m x n young tableau in O(m+n) time is

```
i = m, j = n
Y [i, j] = key
while Y [i − 1, j] > Y [i, j] or Y [i, j − 1] > Y [i, j] do
   if Y [i − 1, j] < Y [i, j − 1] then
      Swap Y [i, j] and Y [i, j − 1]
      j − −
   else
      Swap Y [i, j] and Y [i − 1, j]
      i − −
   end if
end while
```

Insert(Y, key) We know that this program has a runtime of O(n + m) since I + j decreases at each step, starts as n + m, and is bounded by 2 below.

e.

Call the algorithm from part d on each of the n^2 components in a Young Tableau. Then, in section c, n^2 execute the procedure to acquire the numbers in increasing order. Both of these actions require at most $2n \in O(n)$ seconds and are repeated n^2 times, therefore the overall runtime is $O(n^3)$

f.

We'll begin at the lower-left corner. We compare the current element "current" to the key we're looking for and move up if current > key and right if current < key. If current = key, given number is sorted; otherwise, we terminate if we leave the tableau.