Recursion

Please read turn-in checklist at the end of this document before you start doing exercises.

Section 1: Pen-and-paper Exercises

1. Analyze the following code and provide a "Big-O" estimate of its running time in terms of n. Explain your analysis.

```
function fun(int n)
{
    if (n == 0)
            return 0;
    else
            return n + fun(n-1);
}
```

Note: Credit will not be given only for answers - show all your work:
(3 points) steps you took to get your answer.
(2 points) your answer.

```
for i = 1 to n
result = result + i
```

T(0) : 0 //O(1)
T(n) = T(n-1)+1
T(n-1) = T(n-2)+2
T(n-2) = T(n-3)+3
T(k) = T(n-k) +k

T(n)= T(n-k) + k

T(n) = 1 +n

O(1)+T(n-1) ——> O(n)

2. We are given an array A[] of n numbers in an arbitrary order. Design an algorithm to find the minimum and second minimum element in A[] using at most 3/2n -2 comparisons.
   (i) describe the idea behind your algorithm in English (3 points);

   (ii) provide pseudocode (5 points);

   (iii) analyze the number of comparisons used in your algorithm (2 points).

Note: Full credit (10 points) will be awarded for an algorithm that uses at most 3/2n -2 comparisons. Algorithms that make more comparisons will be scored out of 3 points.

I) The idea behind the code is to use the divide and conquer method ignorer to find the two smallest elements of an array. This was achieved by creating an object of Min2ndMin thus allowing multiple returns to the main function. The algorithm side of this code looks at the array and determines its size. If the size is 1 then the min and 2ndmin are the same thing or if the size is 2 then it determines which of the two are smaller and names those to the variables or if the size is more than 1 it splits it in to two parts. It then gets the two minimums from each side and compares those to and out the minimum of the original array and then compares the leftover minimum with the 2nd minimum on the array that just had the minimum value to determine the second minimum of the input array.

II) Pseudocode

```
dcfindmin2ndmin(A[], I , J)

Dmin pair

if( j - I = 0)

    pair.minn  = I

    pair.min2nd = J;

End if

Else if(J - I = 1)

    if(A[I] > A[j])

            pair.minn  = J

            pair.min2nd = I;

    end if

    else

            pair.minn  = J

            pair.min2nd = I;

    end else

end else if

Else

    mid = (i + j) / 2

    startLeft = i

    endLeft = mid

    startRight = mid + 1

    endRight = j

    Min2ndMin pairLeft

    Min2ndMin pairRight

    pairLeft = dcfindmin2ndmin(A, startLeft, endLeft);

    pairRight = dcfindmin2ndmin(A, startRight, endRight);

    if (A[pairLeft.min] > A[pairRight.min])
```

```
            pair.min = pairRight.min;
            pair.min2nd = min(A, pairLeft.min, pairRight.min2nd);
    end if
    else
            pair.min = pairLeft.min;
            pair.min2nd = min(A, pairRight.min, pairLeft.min2nd);
    end else
End else
Return pair



Min(A[], aIndex, bIndex)
if(A[aIndex] > A[bIndex])
    return bIndex
End if
Else
    return aIndex
End else
```

III) Analyze Time

$C(n) = C(n/2) + C(n/2) + 2$

Reduction

$C(n/2) = 2((n/4) +2$

$C(n/4) = 2((n/8) +2$

$C(n/8) = 2((n/16) +2$

Replacement

$C(n) = 2((n/2)+2$

$2^2((n/4)+2^2+2$

$2^k((n/2^k)+2^{k-1}+\ldots+2^1$

$C(n/2^k) = C(2)$

$n/2 = 2^k$

$(2^{k+1}-1/(2-1))-1 = 2^{k+1}-2$

$C(n) = n/2\ C(2) + 2^{k+1}-2$

3/2n -2 comparisons

3. Using the master theorem discussed in class, find a tight bound for the solution of the following recurrence equation (3 points each).

   a. $T(n) = 2T(n/2) + n^3$
      $a = 2\ \ b = 2\ d = 3$
      $d > \log_2 2$ —-> $O(n^d) = O(n^3)$
   b. $T(n) = T(9n/10) + n$
      $a = 1\ b = 10\ d = 1$
      $\log_{10} 1 = 0 < 3$ $O(n^d) = O(n^1)$
   c. $T(n) = 16T(n/4) + n^2$
      $a = 16\ b = 4\ d = 2$
      $\log_4 16 = 2 = d$ —>$O(n^{d*}\log(n))$ —> $O(n^{2*}\log(n))$

   d. $T(n) = 7T(n/3) + n^2$
      $a = 7\ b= 3\ d = 2$
      $\log_3 7 = 1.7 < d$ —> $O(n^d)$ —> $O(n^2)$
   e. $T(n) = 2T(n/4) + \mathrm{sqrt}(n)$
      $a = 2\ b = 4\ d = .5$
      $\log_4 2 = .5 = d$ —>$O((n^d)(\log(n)))$ —> $O((n^{.5})(\log(n)))$

Section 2: Java Implementation

4. Implement problem 2 in Java (30 points).
   Note:

   Find a file called Problem2.java in assignment 4 folder.

   Complete the method of dcfindmin2ndmin ().

   Test your method in the main method provided following the comments.

   Full credit (30 points) will be awarded for an algorithm that uses at most 3/2n -2 comparisons. Programs that make more comparisons will be scored out of 5 points.

TURN-IN CHECKLIST:

1. **Answers to Section 1 (.doc/.txt), and to Section 2 (all your source Code (.java files)). Remember to include your name, the date, and the course number in comments near the beginning of your code/report.**

2. **Create a folder and name it 'FirstName_LastName_assignment_4'. In the newly created folder copy and paste your files (.doc/.txt/.java files). Then compress the folder, and submit to iLearn.**