James Stallkamp

CS 325 HW 2

1)

a) $T(n) = T(n-2) + n$

$= n + (n-2) + T(n-4)$

$= n + (n-2) + (n-4) + T(n-6)$

$= n + (n-2) + (n-4) + (n-6) + T(n-8)$

$= \frac{1}{4(n(n+2))}$

$= \theta(n^2)$

b) $T(n) = T(n-1) + 3$

$= T(n-2) + 3 + 3$

$= T(n-2) + 3 + 3 + 3$

$= T(n-2) + 3 * 3$

$= T(n-k) + 3 * k$

$= 3n$

$\theta(n)$

c) $T(n) = 2T\left(\frac{n}{4}\right) + n$

using the master method $T(n) = \alpha T\left(\frac{n}{b}\right) + F(n)$

$a = 2, b = 4, f(n) = n$

$n^{\log_b a} = n^{\log_4 2} = n^{1/2}$

$n^{\log_4 2 + \varepsilon} = f(n), where\ \varepsilon = 1/2$

Case 3 then if the regularity condition holds for f(n)

$af\left(\frac{n}{b}\right) = 2\left(\frac{n}{4}\right) = \left(\frac{2}{4}\right)n \leq cf(n)\ for\ c = 1$

Therefore according to the master theorem case #3, $T(n) = \theta(n)$

d)$T(n) = 4T\left(\frac{n}{2}\right) + n^2\sqrt{n}$

$f(n) = n^2\sqrt{n} = n^{5/2} and\ n^{\log_b a} = n^{\log_2 4} = n^2$

Since
$n^{\frac{5}{2}} = \Omega(n^{2+\varepsilon}) for\ \varepsilon =$
$\frac{1}{2}$ $we\ look\ at\ the\ regularity\ condition\ in\ case\ 3\ of\ the\ master\ theorem.$

We have $af\left(\frac{n}{b}\right) = 4\left(\frac{n^2}{2}\right)\sqrt{\frac{n}{2}} = \frac{n^{\frac{5}{2}}}{\sqrt{2}} \leq \frac{cn^{\frac{5}{2}} for 1}{\sqrt{2}} \leq c < 1 \quad case\ 3\ applies\ and\ T(n) = \theta(n^2\sqrt{n})$

2

a) The input is first sorted so that if the list is of length two then the list can be immediately returned.
b) No. when N 4 the list would be divided into unequal subparts and then the sort wouldn't work.
c) $T(n) = 2T\left(\frac{n}{3}\right) + 2$
d) $\theta\ (\log n)$

3

PseudoCode:

```
Function quaternarySearch(A, key, start, end)

        If(start > end)

                Return false

        Piece1=start+(end-start)/4

        Piece2=start+ 2 *(end-start)/4

        Piece3=start+ 3 *(end-start)/4

        If(A[piece1] == key)

                Return(true)

        Else if(A[piece2] == key)

                Return(true)
```

```
Else if(A[piece3] == key)
    Return(true)
Else If(key < A[piece1])
    Return quaternarySearch(A, key, start, piece1 -1)
Else if(A[piece2] <key & A[piece3] >key)
    Return quaternarySearch(A, key, piece1+1,piece2 -1)
Else if(A[piece3] <key)
    Return quaternarySearch(A, key, piece2+1,piece3 -1)
Else
    Return quaternarySearch(A, key, piece3+1,end)
```

Recurrence

$$T(n) = t\left(\frac{n}{4}\right) + 2 \; or \; T(n) = T\left(\frac{n}{4}\right) + 2k \; or \; T(n) = T\left(\frac{n}{4}\right) + k \; or \; T(n) = T\left(\frac{n}{4}\right) + \theta(1)$$

Using the master method. A =1 b=4 $\log_4 1 = 0 \; n^0 = 1$

Comparing F(n) so case 2 meaning T(n) =$\theta(\log n)$

Compare

Both Binary and quaternary search are $\theta\ (\log n)$

4

Min_Max(a)

If |A| =1 then return min=max=A[0]

Divide A into two equal subsets A1 and A2

$(min_1, max_1) = Min\_Max(A_1)$

$(min_2, max_2) = Min\_Max(A_2)$

$if\ min_1 \leq\ min_2\ then\ \min =\ min_1\ else\ \min =\ min_2$

$if\ max_1 \geq\ max_2\ then\ \max =\ max_1\ else\ \max =\ max_2$

Return(min,max)

Recurrence: $T(n) = 2T\left(\frac{n}{2}\right) + 2$

Solution: $\theta(n)$

5

Solution:

The array is split into two equal pieces and then the function is called twice recursively to find the majority elements of both sub pieces. This is repeated until the sublists are one element long and then they are counted back up.

getMajorityElement(A[0….n])

n = |A|

if n = 1, return a[0]

k = n/2

leftSub = getMajorityElement(a[0…k])

rightSub = getMajorityElement(a[k…n])

if leftSub = rightSub

    return leftSub

lCount = getFrequency(a[0…n], leftSub)

rCount = getFrequency(a[0…n], rightSub)

if lCount > k+1

    return leftSub

else if rCount > k+1

    return rightSub

else

    return NO MAJORITY ELEMENT

recurrence is $T(n) = 2T\left(\frac{n}{2}\right) + \theta(n)$

Solution is $\theta(n)$