ADA Assignment 1

Ankur Sharma, 2016225

January 2018

Partners: Anvit Mangal and Ishaan Bassi.

Problem 1

(a) T(n) = 4T(n/5) + 5n

$$Using\ Master\ Theorem,$$

$$Compare\ (a)\ with\ T(n)\ =\ aT(n/b)\ +\ f(n)$$

$$we\ get:\ a = 4,\ b = 5,\ f(n)\ \epsilon\ \theta(n),\ therefore\ d = 1$$

$$and,\ a < b^d\ as\ 4\ < 5$$

$$Therefore,\ T(n) = \theta(n)$$

(b) T(n) = 5T(n/4) + 4n

$$Using\ Master\ Theorem,$$

$$Compare\ (b)\ with\ T(n)\ =\ aT(n/b)\ +\ f(n)$$

$$we\ get:\ a = 5,\ b = 4,\ f(n)\ \epsilon\ \theta(n),\ therefore\ d = 1$$

$$and,\ a < b^d\ as\ 5\ < 4$$

$$Therefore,\ T(n) = \theta(n^{log_4\ 5})$$

(c) T(n) = 4T($\sqrt{n}$) + $\log^5$ n

$$Let\ n = 2^m$$

$$Substitute\ n\ in\ (c),\ we\ get$$

$$T(2^m) = 4T(2^{m/2}) + (log 2^m)^5$$

$$Let\ S(m) = T(2^m)$$

$$S(m) = 4T(m/2) + (log 2^m)^5$$

$$S(m) = 4T(m/2) + (m * log2)^5$$
$$S(m) = 4T(m/2) + m^5 * (log2)^5$$
$$Using\ Master\ Theorem,$$
$$Compare\ (c)\ with\ S(n)\ =\ aS(n/b)\ +\ f(m)$$
$$we\ get:\ a = 4,\ b = 2,\ f(n)\ \epsilon\ \theta(m^5),\ therefore\ d = 5$$
$$and,\ a < b^d\ as\ 4\ < 2^5$$
$$S(m) = \theta(m^5)$$
$$T(n) = T(2^m) = S(m) = \theta(m^5)$$
$$Therefore,\ T(n) = \theta((log_2 n)^5)$$

(d) T(n) = 2T(n/3) + 1

$$T(n) = 4 * T(n/9) + 1 + 2$$
$$T(n) = 2^k * T(n/3^k) + 2^{k-1} - 1$$
$$T(1) = 1$$
$$n/3^k = 1$$
$$n = 3^k \rightarrow k = log_3 n$$
$$Thus,\ T(n) = 2^{log_3 n} * T(1) + 2^{log_3 n - 1} - 1$$
$$T(n) = (2^{log_2 n})^{1/log_2 3} + (2^{log_2 n})^{1/log_2 3}/2 - 1$$
$$T(n) = n^{log_3 2} + (n^{log_3 2})/2 - 1$$
$$T(n) = O(n) \qquad because\ log_3 2 = 0.63$$

(e) T(n) = 2T(n-1) + n (For this n<1 T(1)=1)

$$T(n) = 4 * T(n-2) + 2(n-1) + n$$
$$T(n) = 2^k * T(n-k) + n + 2(n-1) + ... + 2^{k-1}(n-k+1)$$
$$n+2(n-1)+....+2^{k-1}(n-k+1) = n(1+2+4+...+2^{k-1}) - (0+2+8+....+(k-1)2^{k-1})$$
$$0 + 2 + 8 + ... + (k-1)2^{k-1} = S : This\ is\ an\ AGP$$
$$Therefore,\ S = 2^k(k-2) + 2$$
$$T(n) = 2^k * T(n-k) + n(2^k - 1) + 2^k(k-2) + 2$$
$$T(n) = 2^k * T(n-k) + n * 2^k - n + k * 2^k - 2k + 2$$
$$T(1) = 1$$
$$n - k = 1 \rightarrow k = n - 1$$
$$T(n) = 2^{n-1} * T(1) + n * 2^{n-1} - n + (n-1) * 2^{n-1} - 2n + 2 + 2$$
$$T(n) = 2^{n-1} + n * 2^{n-1} + n * 2^{n-1} - 2^{n-1} - 3n + 4$$
$$T(n) = n * 2^n - 3n + 4$$
$$T(n) = O(n * 2^n)$$

Problem 2

```
Integer array A of size n;
print transition(A, 0, n-1)
function Integer transition(A, l, r):
    if(r<l) return -1;
    mid = (l + r)/2
    if(A[mid]==0):
        if(A[mid+1]==1): return mid+2;
        else return transition(A, mid+1, r);
    else if(A[mid]==1):
        if(A[mid-1]==0) return mid;
        else return transition(A, l, mid-1);
```

Proof of Correctness:

We are here trying to find(search) first occurrence of '1'.
So, the approach is as follows:
1) We have a window of size r-l+1 in which we are looking for first occurrence of '1' i.e. just after '0'. Initially, window is of size n.
2) We also have a middle element in window. Now, when we check A[mid], it could be '0' or '1'
3) Case I: if it is '0', this implies index of first occurrence of '1' is after mid. Since, array is sorted. Now, we check if A[mid+1] is '1' then, we simply return the index(mid) else we decrease our window size by setting left most element of window to mid + 1. Therefore, window size gets half and required index stays in the window.
4) Case II: if it is '1', this implies index of first occurrence of '1' is less than or equal to mid. Now, we check if A[mid-1] is '0' then we simply return the index(mid) else we decrease our window size by setting right most element of window to mid - 1. Therefore, window size gets half and required index stays in the window.
5) In case, '1' is not present, code gets terminated when r <l. Since, size of window decrease at each call.

Time Complexity:

As the algorithm is based on binary search,
$T(n) = T(n/2) + 1$
Therefore, $T(n) = O(\log(n))$

Problem 3

Let 'a' be a sorted array with n integers. And, exists(a, l, r, w) be our recursive function.
Idea: We will keep two pointers in our array. Initially, first, say 'l' , one would

be at 0^{th} index i.e. first element of array and second one, say 'r', would be at $(n-1)^{th}$
i.e. last element in the array.
if a[l] + a[r] = w. Then, we are done.
if a[l] + a[r] > w. We will decrement our right pointer 'r' by 1 i.e. r–.
if a[l] + a[r] < w. We will increment our left pointer 'l' by 1 i.e. l++.
Following is the pseudo-code of function exists:

```
exists(a, l, r, w):
    if(l > r) return false;
    if( a[l] + a[r] == w and l != r ) return true;
    if( a[l] + a[r] > w) return exists(a, l, --r, w);
    if( a[l] + a[r] < w) return exists(a, ++l, r, w);
    return false;
```

Proof of Correctness:

As, it can be seen from the algo that we have 4 cases:
Case I: a[l] + a[r] >w, here as sum is greater than w, we decrement r by 1. Because array is sorted (a[l] is less than or equal to a[r]).
Case II: a[l] + a[r] <w, here as sum is less than w, we increment l by 1. Because array is sorted (a[l] is less than or equal to a[r]).
Case III: a[l] + a[r] = w, we gets our answer and we simply return true as there exists two such element whose sum is w. Therefore, Termination.
Case IV: l >r.We don't have such elements. Termination.
As we can see, at each stage we are either incrementing l or decrementing r or returning true or false. Therefore, it is evident from function itself that code does not run forever.

Time Complexity:

If such elements do not exist. Then, algo runs n times. Therefore, worst time: O(n).
if a[0] + a[n-1] = w, it's the best case, i.e. O(1).

Problem 4

Analyze the complexity of stable marriage problem in all cases. Suppose, we have n men and n women.
Best case O(n): Given each men has different women on their no. 1 priority i.e. no men has same women on their no. 1 priority. This implies, when a men proposes for the first time he gets engaged. And this pair never gets broken as no other men is going to propose to that women as they all have their own different women at no. 1 priority. And this is true for all men. Therefore, O(n).
Worst Case $O(n^2)$: means maximum no. of proposals. This could happen when a man m proposes n times. This implies, m gets the women with least priority on his list say w. Now, we can say other men can propose n-1 times at max.

Because otherwise if there is some man m' who has proposed n times means he got engaged with a women, say w', with least priority on his list. As we know w is least preferred by m and w' is least preferred by m' implies (m, w') and (m', w) are more stable than (m, w) and (m', w'). Therefore, a contradiction because it's a stable marriage. Hence, T(n) = n + (n-1)(n-1) = n(n-1) + 1. i.e $O(n^2)$.

Problem 5

Given: n balls out of which one weighs 2 Kg and other 1 Kg.
Idea: Now, we add minimum no. of balls of 1 kg to n such n + x is divisible by 3. Let n + x = m. Now, we divide m into three buckets. When we do that, we get 3 buckets with same no. of balls.
Now, if first 2 buckets have same weight, then 2 kg ball is
present in 3rd bucket. And we call the function again passing only 3rd bucket. Else, we call the function again passing only bucket with heavier weight than other.

Code:

```
function(bucket):
    m = size(bucket)
    if( m % 3 != 0 ) m = m + 3 - m % 3
    bucket1 = first m/3 balls
    bucket2 = next m/3 balls
    bucket3 = remaining balls
    if( size(bucket1) == 1 and weight(bucket1)==2 ) return bucket1;
    if( size(bucket2) == 1 and weight(bucket2)==2 ) return bucket2;
    if( size(bucket3) == 1 and weight(bucket3)==2 ) return bucket3;
    if( weight(bucket1)==weight(bucket2) ) return function(bucket3);
    else if( weight(bucket1)¿weight(bucket2) ) return function(bucket1);
    else return function(bucket2);
```

Proof of Correctness and Complexity:

Since, we are dividing m balls into three parts, till we get a bucket with 1 ball and 2 Kg weight. And function gets terminated, at this point. Also, as we are adding some balls if size of bucket is not divisible by 3. But as they are added into bucket3 and we we are doing operations on b1 and b2, it does not affect us. Therefore, T(n) = T(n/3) + 1. Hence, $T(n) = O(\log_3 n)$