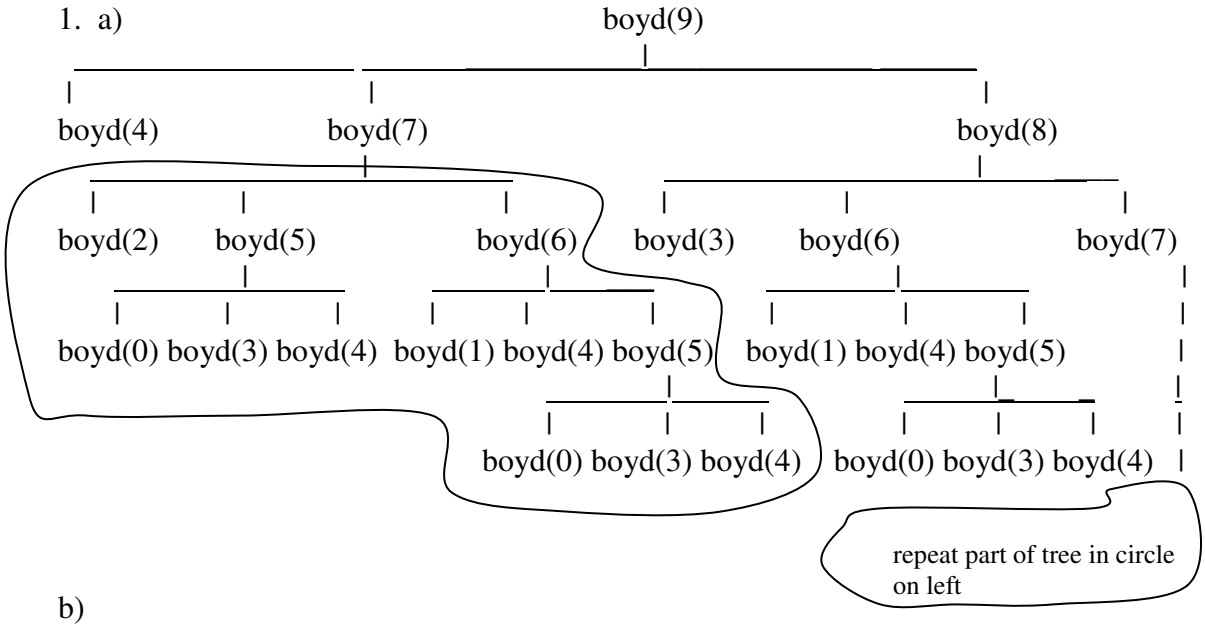
CSI 3105 Assignment 1 -- 2011
Solutions

1. a)



b)

n	0	1	2	3	4	5	6	7	8	9
T(n)	1	1	1	1	1	4	7	13	22	37

c)

Claim: $T(n) > 3^{n/5}$ for $n \geq 5$

Proof of Claim: By induction.

Base Case:

(Note: We need to verify the claim for the first 5 values of n, i.e. for n = 5, 6, 7, 8 and 9, since we need to use the Induction Hypothesis for value n-5 later in the Induction Step).

For n = 5, $T(n) = 4 > 3^{5/5}$? Yes! The claim is true for n = 5.
For n = 6, $T(n) = 7 > 3^{6/5}$? Yes! The claim is true for n = 6.
For n = 7, $T(n) = 13 > 3^{7/5}$? Yes! The claim is true for n = 7.
For n = 8, $T(n) = 22 > 3^{8/5}$? Yes! The claim is true for n = 8.
For n = 9, $T(n) = 37 > 3^{9/5}$? Yes! The claim is true for n = 9.

Induction Hypothesis (I.H.)

The claim is true for all m such that $5 \leq m < n$, i.e. $T(m) > 3^{m/5}$ for all m such that $5 \leq m < n$.

Induction Step (I.S)

Prove the claim is true for n, where $n \geq 10$ (note that we have already shown it is true for n = 5, 6, 7, 8 and 9).

Consider the call tree which corresponds to the algorithm. The number of calls for T(n) is 1 (for the initial call) + T(n-5) (for the left subtree under the initial call) + T(n-2) (for the middle subtree under the initial call) + T(n-1) (for the right subtree under the initial call).

Therefore,

$T(n) = 1 + T(n-5) + T(n-2) + T(n-1)$ (from the above explanation)

$> 1 + 3^{(n-5)/5} + 3^{(n-2)/5} + 3^{(n-1)/5}$ (since $n \geq 10$ for the I.S., we know each of n-5, n-2, and n-1 are $<n$ and ≥ 5 and thus we can apply the I.H. for T(n-5), T(n-2) and T(n-1))

$> 3^{(n-5)/5} + 3^{(n-2)/5} + 3^{(n-1)/5}$ (since $1 > 0$)

$> 3^{(n-5)/5} + 3^{(n-5)/5} + 3^{(n-5)/5}$ (since $3^{(n-2)/5} > 3^{(n-5)/5}$, $3^{(n-1)/5} > 3^{(n-5)/5}$)

$= 3 * (3^{(n-5)/5})$ (grouping the terms)

$= 3^{(n-5+5)/5}$ (adding the exponents)

$= 3^{n/5}$

Thus $T(n) > 3^{n/5}$ for $n \geq 5$, as required, and the claim is proved.

2. (We assume that the input n is a non-negative integer)

```
int boydNum(int n)
{
 index i;
 int b[0..n];
 while ((i <= n) && (i <= 4))  {
   b[i] = i;
   i++
 }
 for (i = 5; i <= n; i ++)
      b[i] = b[i-1] + b[i-2] + b[i-5] + 2;
 return b[n];
}
```

Worst Case Analysis:

Fixed input size: n, the index of the term of the Boyd sequence we wish to find.
Basic operation being counted: Additions and subtractions (but not loop counter additions, as specified in the question)
Input that gives worst case: All the same.
Analysis:
All the additions/subtractions occur inside the for loop (3 additions and 3 subtractions per loop). There are n-4 iterations of the for loops, so
$W(n) = 6 * (n-4) = 6n - 24$.

3.

a) Iteration 1: low = 1, high = 8, mid = 4
Iteration 2: low = 1, high = 3, mid = 2
Iteration 3: low = 1, high = 1, mid = 1
After this: low = 1, high = 0 and we do not enter the loop

Total number of comparisons is 2 per iteration for 3 iterations = 6
From class: W(n) = 2lgn + 2, so W(8) = 2 (lg 8) + 2 = 8
So in this case we get fewer comparisons than worst case.

b) Iteration 1: low = 1, high = 16, mid = 8
Iteration 2: low = 9, high = 16, mid = 12
Iteration 3: low = 9, high = 11, mid = 10
Iteration 4: low = 9, high = 9, mid = 9
After this: low = 9, high = 8, and we do not enter the loop.

Total number of comparisons is 2 per iteration for 4 loop iterations = 8.
From class: W(n) = 2lgn + 2, so W(16) = 2 (lg16) + 2 = 10
So in this case we get fewer comparisons than worst case.

4. a)

Fixed input size: n
Basic operation being counted: multiplications
Input that gives worst case: all the same
Analysis:
First consider the number of multiplications that occur when i = 1. When i = 1, we have j= 1, 2, 3, ...,n. For each of these values of j, we have the k loop goes from 1 to j, with two multiplications happening per loop. Thus for each value of j we have 2j multiplications. Thus summing 2j over j, j from 1 to n gives the number of multiplications for i = 1, namely

$$\sum_{j=1}^{n} 2j = 2\sum_{j=1}^{n} j = 2\frac{n(n+1)}{2}$$ (using the formula for the sum of the first n numbers)

$$= n(n+1)$$ (simplifying)

Now consider other values for i. It is easy to see that the number of multiplications is n(n+1) for the other values of i as well. Since in the outer loop, i goes from 1 to n, the total number of multiplications will be

$$n*(n(n+1)) = n^2(n+1) = n^3 + n^2.$$

So $W(n) = n^3 + n^2$.

b)

Fixed input size: n

Basic operation being counted: multiplications

Input that gives worst case: all the same

Analysis:

In general, for each value of i in the outer loop, the inner k loop executes 3^i times, with one multiplication per loop. So for each value of i there are 3^i multiplications. Since the outer i loop goes from 0 to n-1, this gives the total number of multiplications to be

$$= \sum_{i=0}^{n-1} 3^i$$

In Appendix A, page 439, is the formula

$$= \sum_{i=0}^{n} r^i = \frac{r^{n+1} - 1}{r - 1}$$

We can use this formula to calculate our total number of multiplications by substituting 3 for r, and summing to n-1 instead of n. This gives

$$W(n) = \frac{3^n - 1}{2}$$

5. a)

Input: A sorted array A[1…] containing n numbers and a number num.

```
index Search(int A[] , int n, int num)
{
  index i;  /i will be the minimum index such that num should be inserted into A[i]
  index k;  /k will be an index used when scanning through A
  i = n+1;    /this will be the final value for i  if num>A[n]
  k = 1;
  while (i == n+1) && (k<=n){
     if (num <= A[k])
         i=k;
     else
         k++;
   }
   return i;
}
```

Worst case analysis
Fixed input size: n, the number of numbers in A.
Basic operation being counted: Comparisons of num with elements in A.
Input that gives worst case: When num is bigger than all the numbers in A, or when num = A[n].
Analysis:
For the worst case input, the algorithm will execute the while loop n times. There is one comparison for each loop, so W(n) = n.

b)
Input: A sorted array A[1…] containing n numbers, a number num to be inserted and an index i which is where num is to be inserted.

```
void MakeRoom(int A [], int n, index i)
{
  index k;
  for (k=0; k <= n-i ; k++)
     A[n+1 - k] = A[n-k];     /This loop moves everything from A[n] down to A[i] one
                               position to the right , in that order)
 }
```

Worst case analysis
Fixed input size: n, the number of numbers in A.
Basic operation being counted: The number of times an array element is moved.
Input that gives worst case: When i=1.
Analysis
In the worst case, every number in A must be moved over 1 position. This is a total of n moves. So W(n) = n.

c)
Worst case analysis for Insert
Fixed input size: n, the number of numbers in A.
Basic operation being counted: The number of comparisons of num with elements in A, and the number of times an array element is moved.
Input that gives worst case:
In general, algorithm Search does i comparisons if the correct insertion position is i. Given position i, algorithm MakeRoom moves a total of n-i+1 numbers in A. So if the correct insert position is i for num, the combined work of the two algorithms will be i + (n-i+1) = n+1. So when the algorithms are combined, the total work is n+1 no matter what the input. So all cases are the same.

Analysis: W(n) = n+1, as explained above.

6) a)

(NOTE: As discussed in class, you can assume the numbers in A are integer, but you cannot assume they are non-negative or distinct.)

Idea of Algorithm:
First check to see if all of the numbers in the array A are in the correct range, i.e. that they all lie between 1 and n. If not, stop. If the numbers are all in the correct range, create a new array Count[1..n] where Count[j] will be used to represent the number of times the number j appears in A. Note that since all the numbers in A are in the correct range and integer, they each represent an index of the array Count. Initialize all components of Count to 0. Then for i=1, 2, .., n let Count[A[i]]= Count[A[i]] + 1. Then A is a permutation of the numbers 1,...,n if and only if no entry of Count is >= 2.

Pseudocode Algorithm:

```
boolean CheckPermutation(int A[1..n], int n)
{
index i,j;
int Count[1..n];
boolean Answer, InRange;
i=1;
InRange = true;
while (InRange==true) && (i<=n){
      if (A[i] > n) or (A[i] <1)
           InRange = false;
      i++;
}
if (InRange == false)
   Answer = false;
else {
   for (j = 1; j <= n; j ++)
      Count[j] = 0;
   i = 1;
   while (Answer = true) && (i<=n){
      j=A[i];
      Count[j] = Count[j] + 1;
      if (Count[j] >1)
           Answer = false;
      i++;
    }
 }
return Answer;
}
```

b)
Worst case analysis
Fixed input size: n, the number of numbers in A.
Basic operation being counted: Comparisons of elements in A with anything, and writing

a number to an array.

Input that gives worst case: When A is a permutation of the numbers 1,..., n.

Analysis:
The first while loop checks if each number is in the correct range (which they will be for our worst case input) and there are two comparisons involving numbers in A for each iteration of the loop for a total of **2n** comparisons.

The for loop initializes each component of Count to 0 for a total of **n** writing operations to an array.

The second while loop writes to Count for each entry of A, for a total of **n** writing operations to an array.

Hence $W(n) = 2n + n + n = 4n$, which makes this a linear-time algorithm.