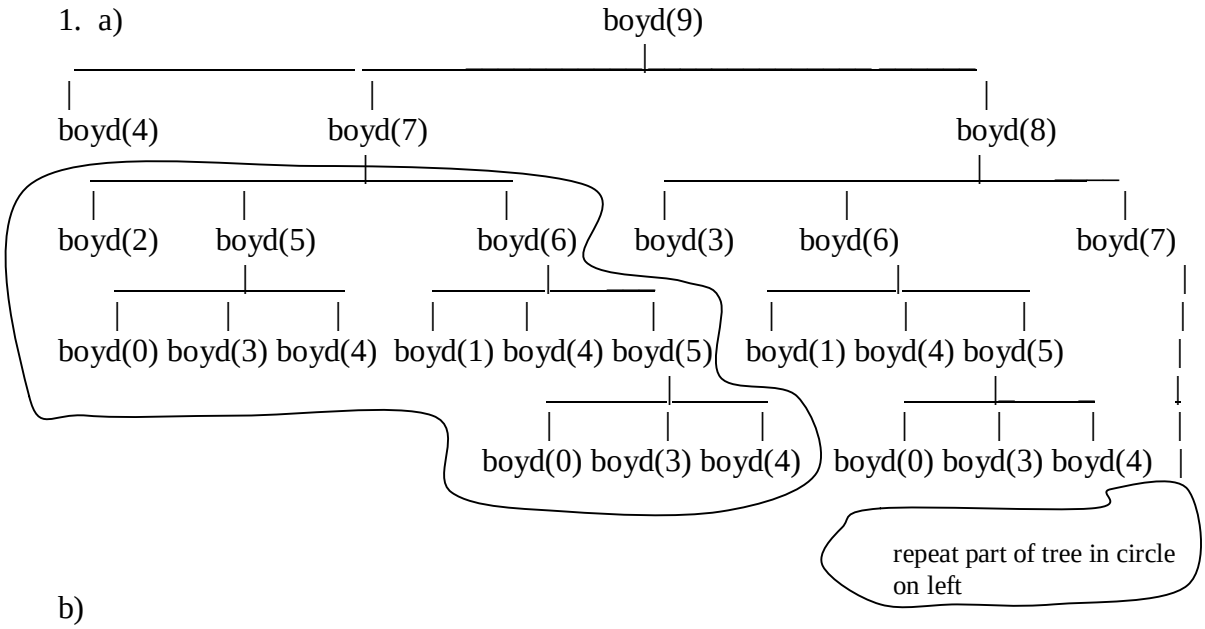
CSI 3105 Assignment 1 -- 2014
Solutions

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: $W(n) = 2\lg n + 2$, 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: $W(n) = 2\lg n + 2$, so $W(16) = 2 (\lg 16) + 2 = 10$
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:
(Note: This analysis can also be completed in chart form, as we did in class).
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 (and on the formula sheet for the course) 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) Analysis of Alg. 1:

Input Size: n, the number of keys in the list, n a power of 2.
Basic Operation Counted: Comparison of x with keys, and comparison of keys with each other.
Input that gives worst case: When none of the values of for which we are searching are in the list.

Analysis: From class, W(n) = n for sequential search. Since Alg. 1 uses sequential search R times, W(n) = R*n for Alg. 1.

Analysis of Alg. 2:

Input Size: n, the number of keys in the list, n a power of 2.
Basic Operation Counted: Comparison of x with keys, and comparison of keys with each other.
Input that gives worst case: When all of the values of x for which we are searching are bigger than all of the entries in the list.

Analysis: From class, W(n) = (n(n-1))/2 for exchange sort, and W(n) =2lgn + 2 for binary search (iterative version) for n a power of 2. So performing exchange sort once and binary serach R times gives W(n) = (n(n-1))/2 + 2R(lgn + 1) for Alg. 2.

b) We want to see when W(Alg 2)(n) < W(Alg1)(n) (i.e. for which values of R). i.e. when do we have

$(n(n-1))/2 + R(2\lg n + 2) < n*R$?

Solving for R (and of course, you would have more details here!), we end up with the result

$R > (n(n-1))/(2n-4\lg n-4)$

c) Using the formula from b), we put in n=2048 to obtain R > 1035.64. Thus, since R is integer, we conclude that we must have R >= 1036 to make Alg. 2 more efficient when n = 2048.

6) a)
(NOTE: As mentioned in the assignment, you can assume the numbers in N 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 N are in the correct range, i.e. that they all lie between 1 and k. If not, stop. If the numbers are all in the correct range, create a new array Count[1..k] where Count[j] will be used to represent the number of times the number j appears in N. Note that since all the numbers in N 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, .., k let Count[N[i]]= Count[N[i]] + 1. Then N is a permutation of the numbers 1,...,k if and only if no entry of Count is >= 2.

Pseudocode Algorithm:

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

```
if (InRange == false)
   Answer = false;
else {
   for (j = 1; j <= k; j ++)
      Count[j] = 0;
   i = 1;
   while (Answer = true) && (i<=k){
      j=N[i];
      Count[j] = Count[j] + 1;
      if (Count[j] >1)
          Answer = false;
      i++;
    }
 }
return Answer;
}
```

b)

Worst case analysis

Fixed input size: k, the number of numbers in N.

Basic operation being counted: Comparisons of elements in N with anything, and writing a number to an array.

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

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 N for each iteration of the loop for a total of **2k** comparisons.

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

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

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