Student: Leila Erbay
ID: 260672158
Assignment #6

Q1

a)
```
Algo: GreedyChoice(int C[0…k-1], int U[0…k-1], int k, int N){
        Input:
             An array C of costs for the k objects
             An array U of utilities for the k objects
             A total Budget N
        Output: An array Q[0…k-1] of quantities to purchase

        /* initialize a total cost, total utility, budgetLeft, and a quantity
        * array that will hold the quantity for each good to be selected */
        int totalC = 0;
        int totalU = 0;
        int budgetLeft = 0;
        qty = 1;
        int Q[] = new int [k];

        int i = k-1;


        while(totalC < N) {
             if( i >=0 ){
                  /* add to total utility, total cost, and quantity array as
                  * long as the total cost is less than the initial budget */
                  totalU += U[i];
                  totalC += C[i];
                  Q[i]+= qty;
                  if (totalC > N) {
                  /* if total cost is greater than initial budget, decrease
                  * total utility, total cost, quantity for that item and
                  * decrement i
                      totalU-=U[i];
                      totalC-=C[i];
                      Q[i]=Q[i]-1;
                      if (i ==0){
                      // if i = zero than we will not be able to buy any more
                           break;
                      }
                      i--;
                  }
             /*update budget with the initial budget minus the current total
              cost */
             budgetLeft = N - totalC;
             }
        }
```

b) For the C, N, U provided in the second example where N = 26, my greedy algorithm fails to produce the optimal choice. Instead of providing an optimal solution where C = 26 and U = 25, my algorithm gives a solution where C = 26 and U = 23.

c) Algo: DynProgChoice(int C[0…k-1], int U[0…k-1], int k, int N){

```
input:
     An array C of costs for the k objects
     An array U of utilities for the k objects
     a Total budget N
output:   a max utility for the given budget

int maxUtil = -1;
int [] optUtil = new int [N+1];


/* base case: if no. of objects is zero or the budget is zero, you
cannot buy any good thus the maxUtil you can receive is 0. */
if (k == 0 || N ==0){
     maxUtil = 0;
     optUtil[0] = maxUtil;
     return optUtil[0];
}

for ( int i = 0; i <= N; i++){
     for (int qty = 0; qty < k; q++){
          /* compare every budget value until max budget to the
          current cost. Determine the max utility by selecting the
          greater utility between the current optimal utility and the
          previous*/
          if (i >= C[i]){
               maxUtil = Math.max(optUtil[i-C[i]] + U[c], maxUtil);
          }
     }
optUtil[i]= maxUtil;
return optUtil[i]]
}
```

d) When the Budget (N) = 38, the max utility you will is 37.

2.

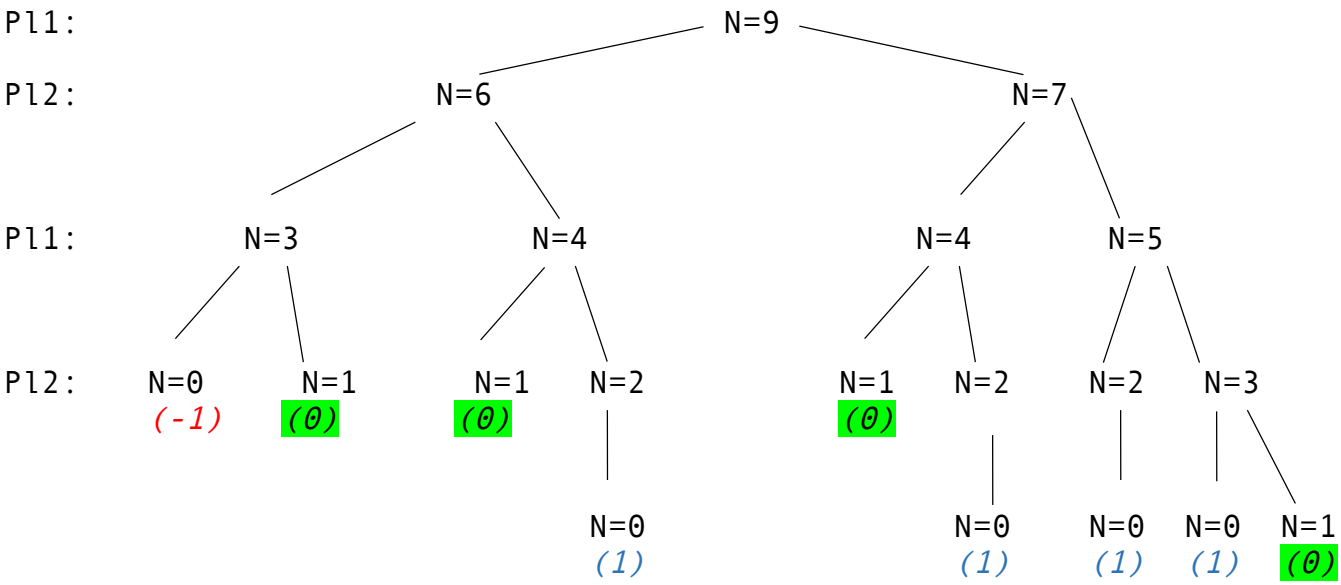
a)

```
sort (int [] A, int n){
     //create an array where each index holds a linked list
     LinkedList <Integer> [] b = new LinkedList[n];
     //this sorted array will contain the final sorted array
     int [] sorted = new int[n];

     /*loop through input array and put it into the correct index of b
     (ie  the correct bucket) */
     for (int i = 0; i < N; i++) {
          int index = (A[i]/2n)*n;
          b[index].addLast(A[i]);
     }

     //sort each Linked List in at index of b
     for (int i =0; i< b[j].length; i ++){
          insertionSort(b[i]);
     }
```

```
        //fill sorted array with each element of b
        //looping through sorted
        for (int k = 0; k < N; ){
             //looping through linked list in each index of b
             for (int q = 0; q < b[q].length; q++){
                  sorted[k]= remove(b[q]);
                  k++;
             }
        }
```

b) You need to know the possible upper and lower bounds of the values so that you can create a correct proportion of buckets to hold the linked lists. It's helpful to know the number of integers of the array to be sorted, but you can always use array length to create an array of the same size.

3.
a)
Player1 Wins = 1 Player2 wins = -1 tie = 0



There will be a tie.

b)

N	1	2	3	4	5	6	7	8	9	10	11	12	13
P1		1	1				1	1	0			1	1
P2					-1					-1			
Tie	0			0		0					0		

- If N is a multiple of 5, player 2 wins
- If N divided by 5 has a remainder of 2 or 3, player 1 wins
- If N divided by 5 has a remainder of 1 or 4, there is a tie

4.

a)

```
Algo: eccentricity(vertex u)
Input: vertex u from the graph
Output: the eccentricity of u

Idea:
1. start at u
2. enqueue u into a queue
3. visit the neighbors of u and change their status to visited
4. every time you visit a neighbor, you put them into the queue and
   increment a counter for eccentricity
5. by the end, the eccentricity counter will have value which represents
   the longest shortest path from u to another vertex

q ← new Queue()
setVisited(u,TRUE)
q.enqueue(u)

distance=getDistance(u)
while(!q.empty())
     vertex r ← S Dequeue()
     distance=getDistance(r);
     for each vertex v in r.getNeighbors()
          if vertex getVisited(v) == false
               setVisited(v,TRUE)
               if (getDistance(v) == 0) setDistance(v)=r+1;
               q.enqueue(v)
     distance=getDistance(v);
return distance
```

b)

```
Algo: is2Colorable(vertex u)
Input: graph vertex u
Output: true if graph of u is colorable, false otherwise

Idea:
1. start with the first node
2. set the color of the first node
3. for each neighbors of u, check if they are visited
4. if they have been visited, compare the color current vertex to the color
   of the previous vertex
5. then recursively call is2Colorable on the neighbors of the neighbors

setVisited(u,TRUE)
c=0;
setColor(u, c)

for each v.getNeighbors(u)
     if(getVisited(v) == TRUE)
          if (getColor(v) == getColor(u)) return false
          else if (is2Colorable(v) == FALSE) return false
     if (getVisited(v) == FALSE)
          setVisited(v, TRUE)
          if (getColor(v) == getColor(u)) return false
          else if (is2Colorable(v) == FALSE) return false
     return truei
```