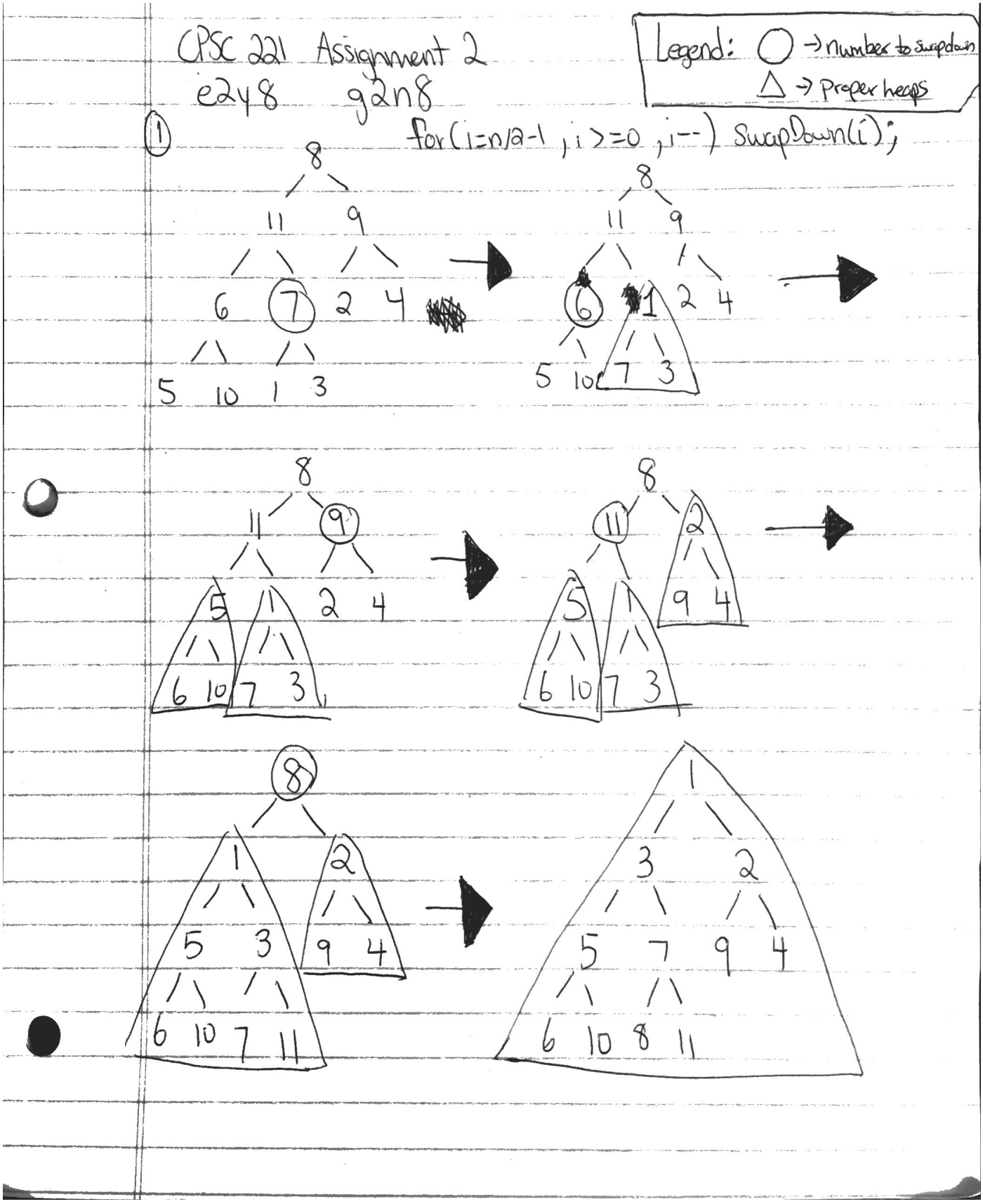
1.


CPSC 221 Assignment 2
e248 g2n8
Legend: ○ → number to swapdown
△ → Proper heaps
for(i=n/2-1 ; i>=0 ; i--) SwapDown(i);
1
8
11 9
6 7 2 4
5 10 1 3
8
11 9
6 1 2 4
5 10 7 3
8
11 9
5 2 4
6 10 7 3
8
11 2
5 9 4
6 10 7 3
8
1 2
5 3 9 4
6 10 7 11
1
3 2
5 7 9 4
6 10 8 11

2.

```
int findPar(string s, i=0){
   if (s[i] == NULL){
      return;
   }
   if (s[i] == ')'){
      return 1 + findPar(s, i++);
   }
   if (s[i] == '('){
      return -1 + findPar(s, i++);
   }
   else{
      return findPar(s, i++);
   }
}
```

3.

(a)

```
// Sort an array A of 0's and 1's so the 0's come before the 1's
// using swapping. The size of the array is n.
int bitsort(int *A, int n) {
int i=0;
int j=n-1;
while( i != j) {
        if( A[i] == 0 ) {
                i++;
        } else if( A[j] == 1) {
                j--;
        } else {
                std::swap( A[i], A[j] );
                }
        }
return j;
}
```

(b)

Invariant: 0 <= i+j <= (n-1)+(n-1)
Proof: The iteration of the loop will always fall between the value of the size of
the (array - 1) all the way to the size of the ((array - 1) * 2). Given that
i starts at 0 and j starts at n-1, and that i increments when it finds a 0 and that j decrements when it finds a 1.

The largest case will be where the array is full of zeros. The iteration of i and j

will start with i = 0 and j = n-1 (loop invariant held since
i + j >= 0) both i and j will get to the point where i = n-1 and j = n-1 (loop invariant held since i + j <= 2(n-1)).

The smallest case is when you have a full array of 1s. The values for i and j will
start at 0 and n-1 respectively (where the loop invariant is held since n-1 is greater than 0 and smaller than 2(n-1)), and iterate down until i is still at 0 and j will be equal to 0. Thus the loop invariant is upheld since i+j = 0
as required.

c)

The correctness of our code follows from the loop invariant and the termination condition of the loop as the loop invariant is upheld during the while loop that states that i is not equal to j and as long as that is held true with the loop invariant being held, the code will return when i=j, thereby verifying the code correctness.

d)

In all cases, except for the case where there are no 1s in the array (where the code will return number of zeros minus one 0s), the code will return the number of 0s in the input array.

4.

If n mod 3 = 0 //div by 3
If n mod 3 = 1
If n mod 3 = 2

3 cases, if n mod 3 = 0,1,2
if n mod 3 != 0 then alice can pick 1 or 2 respectively and be left with 0 remaining pieces for bob to pick up
if n mod 3 = 0 however, then alice cannot pickup all 3 pieces, and is required to leave 1 or 2 pieces, meaning bob can take the remaining

base case n = 1, alice takes 1 and wins the game.
base case n = 2, alice takes 2 and wins the game.
base case n = 3, If alice takes 1, bob takes 2; If alice takes 2, bob takes 1

alice must always make the number of remaining links not divisible by 3

for a case where we have (3n) pieces, alice must keep the number of pieces at the end of each round as anything but 3(n-1) otherwise she conceeds the chance to win to bob.

If there are (3n + 1) or 3(n + 2) pieces then alice must follow the same strategy of trying to reduce the number of pieces to either be 3(n - k + 1) or +2 for k rounds,
or have it reduce to only having only 1 or 2 pieces left on the last round.
Either way all three cases will resolve to one of the different base cases from above and she must play accordingly.

5.

a)

m = 2^16-1 = 0b1111 1111 1111 1111
h(s) = (s0 + s1*256 + s2*256^2 + ...) mod m

Given 3 characters, and 1 byte per character (8 bits), we get 256*256*256 different possibliities, or 16 777 216.
Because we're using staic hashing with m = 2^16-1 we have a set numbr of possible hashes, or 65 535.
By the pidgeon hole principal of there being over 256 times as many input strings as there are hashes,
there must exist at least 257 hash collisions.

b)

Similar to the previous answer, anagrams of strings ocasionally will return different numbres, which are different multiples of 256 but have have the same remainder, that is, (a + (b*256) + (c*256^2) + (d*256^3)) is different numbers when a = 65, b = 66, c = 67, and d = 68 and a = 67, b = 68, c = 65, d = 66 but when you take the mod 255 of them they return the same number, 11.
The basic idea is to find different orderings of the numbers where the lowest denomination is the most important when changing the result of the mod.
For example, if we take out the A term, and only have b,c,d and we use the numbers 2,3,4 we'll get the same result no matter the positioning of the numbers.
We'll therefore want to be careful of what we choose our least significant bit to be when we're trying to calculate permutations of strings which has to the same value.
IE) ABCD, ACDB, ACBD, ADBC, ADCD, ABDC all are collisions
as are BCDA, BCAD, BDAC, BDCA, BADC, BACD, BCAD, etc.

Used wolfram alpha to help compute different hashes according to the formula, also consulted the notes/wiki page on hashing