CS 401 Algorithms 04/12/2019
Assignment1
Due Date: 04/22/2019

Part I: Algorithm Analysis

1. Give the order of growth (as a function of n) of the running times of each of the following code fragments. Please show your work.

a.
```
while (n > 0) {
    for (int k = 0; k < n; k++)
        printReport(); // runs in O(N/2) time
    n = n / 2;
}
```

b.
```
for (int i = 0; i < n; i++)
    for (int j = i+2; j > i; j--)
        for (int k = n; k > j; k--)
            System.out.println(i+j+k);
```

c.
```
void aMethod(int n){
    if (n <= 1)
        return;
    anotherMethod(); //runs in O(1) time
    aMethod(n/2);
    aMethod(n/2);
}
```

d.
```
public static int mystery(int n) {
    int i = 1;
    int j = 0;
    while (i < n) {
        j++;
        if (j == i) {
            i++;
            j = 0;
        }
    }
    return j;
}
```

e.
```
public static int compute(int n) {
    int total = 0;
    for (int i = 1; i < n; i++) {
        int k = 1;
        while (k < i) { k = k + k;          }
        while (k > 1) { k /= 2;    total++; }
    }
    return total;
}
```

f.
```
//n, the length of arr
public static int removeDuplicates(char[] arr) {
    int len = arr.length;
    int i = 0;          // index of current item to find
    while (i < len) {
        int j;          // will be index of duplicate of arr[i]
        for (j = i + 1; j < len; j++) {
            if (arr[i] == arr[j]) break;
        }
        if (j == len) {   // no duplicate of arr[i] found; go to next i
            i++;
        } else {         // duplicate found; shift array over arr[j]
            for (int k = j + 1; k < len; k++) {
                arr[k - 1] = arr[k];
            }
            len--;
            arr[len] = 0;
        }
    }
    return len;
}
```

a)

```
while ( n > 0 ) // log n
{
  for( int k = 0 ;  k < n ; k++) //n
    {
        printReport(); // runs in O(n/2) time ] hot spot
    }
  n = n/2 ;
}
```

n	# of operations
n	n
n/2	n
n/4	n
⋮	
1	n
	+

K

$T(n) = k \cdot n$

$T(n) = n \log n \cdot \frac{n}{2}$ ← from printReport()

$T(n) = n \cdot \frac{1}{2} \cdot n \log n$

$T(n) = n^2 \log n$

$\left(n \cdot \frac{1}{2}\right)^k = 1$

$n \cdot \frac{1^k}{2^k} = 1$

$n \cdot 1^k = 2^k$

$n \cdot \log = k \log 2$

$k = \log n$

---

b)

```
for (int i = 0 ;  i < n ; i++) // O(n)
```

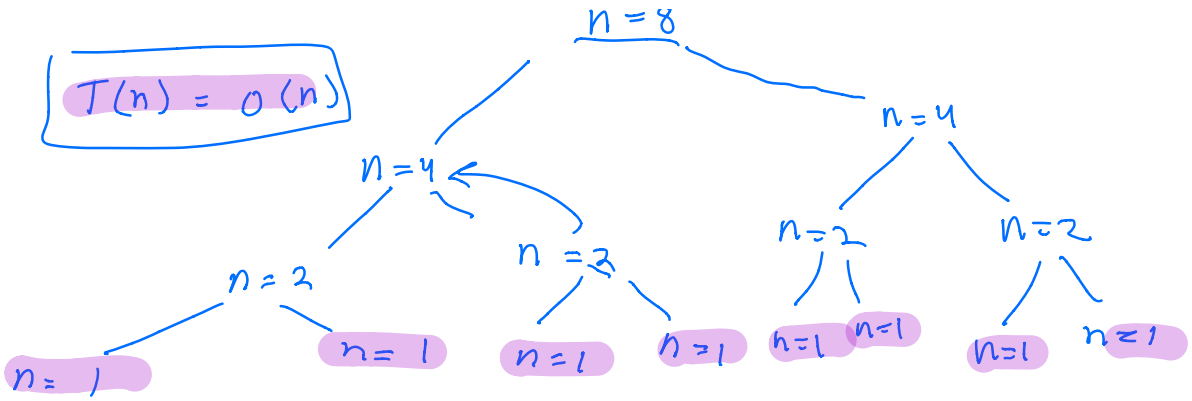
```
{
    for (int j = i + 2 ;  j > i ;  j--)   // O(1)   will always run 2 times
    {
        for (int  k = n ;  k > j ;  k--)   // O(n)
        {
            System.out.println ( i + j + k );   2
        }
    }
}
```

$T(n) = O(n^2)$

$$\sum_{i=0}^{n-1} \sum_{j=1}^{2} \sum_{k=0}^{n-i} 2 \Rightarrow \sum_{i=0}^{n-1} \sum_{j=1}^{2} 2n \Rightarrow \sum_{i=0}^{n-1} 4n \Rightarrow 4n \cdot n = 4n^2 = O(n^2)$$

---

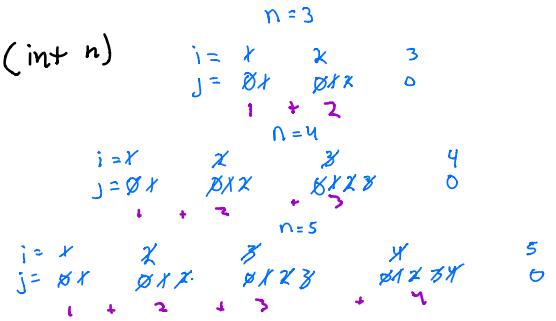
c)
```
void  aMethod ( int n)
{
    if (n <= 1)
    {
        return;
    }
    Another Method;   // runs in O(1) time
    a Method (n/2);
    a Method (n/2);
}
```



---

d)

```
Public static int mystery (int n)
{
   int i = 1;
   int j = 0;
   while (i < n)
   {
     j++;
     if (j == i)
     {
       i++;
       j=0;
     }
   }
   return j;
}
```



J runs $\sum_{j=1}^{n} 1 \Rightarrow \frac{n^2}{2} = n^2 \cdot \frac{1}{2}$

$O(n^2)$

---

e)

```
Public static int compute (int n)
{
    int total = 0;
    for (int i = 1;  i < n; i++) // n
    {
        int k = 1;
        while (k < i)
        {
            k = k + k; // 2, 4, 8 ... doubles      log n
        }
```

```
        while ( k > 1 )
        {
            k /= 2 ;   // 8, 4, 2 ... reduced by half   log n
            total ++ ;
        }
    }
    return total ;
}
```

$$\sum_{j=1}^{n-1} \sum_{k=1} 2^k \qquad 2^1 + 2^2 + 2^3$$

$T(n) = n \log n$

---

f)

```
f.  //n, the length of arr
    public static int removeDuplicates(char[] arr) {
        int len = arr.length;
        int i = 0;           // index of current item to find
        while (i < len) {
            int j;           // will be index of duplicate of arr[i]
            for (j = i + 1; j < len; j++) {
                if (arr[i] == arr[j]) break;
            }
            if (j == len) {   // no duplicate of arr[i] found; go to next i
                i++;
            } else {         // duplicate found; shift array over arr[j]
                for (int k = j + 1; k < len; k++) {
                    arr[k - 1] = arr[k];
                }
                len--;
                arr[len] = 0;
            }
        }
        return len;
    }
```

The while loop will run n times.

The first for loop will run from the start of the array until a duplicate is hit. The second for loop will run from that duplicate until the end of the array. These two for loops run in some capacity for every iteration of the while loop.

$O(n^2)$