ICCS200: Assignment homework-number-1

Poonnarat Nakartit poonpptn@gmail.com
Recitation: Your recitation section
The date

Exercise 2: : Hello, Definition (4 points)

---

(1) Show,using either definition,that $f(n) = n$ is $O(nlogn)$.

$$\begin{aligned}\lim_{n\to\infty} \frac{n}{nlogn} &= \lim_{n\to\infty} \frac{1}{logn} \\ &= \frac{1}{\infty} \\ &= 0\end{aligned}$$

(2) Inclass,we saw that $Big-O$ multiplies naturally. You will explore this more formally here. Prove the following statement mathematically (i.e. using proof techniques learned in Discrete Math): Proposition: If $d(n)$ is $O(f(n))$ and $e(n)$ is $O(g(n))$, then the product $d(n)e(n)$ is $O(f(n)g(n))$.

$d(n)$ is $O(f(n))$ and e(n) is $O(g(n))$. We can write it in terms of matamatical formulars,and since each of them are less than infinity, therefore we can assum that ecah of them can be equal to some constant M and N, where M and N are less than infinity.

$$\lim_{n\to\infty} \frac{d(n)}{f(n)} < \infty$$
$$\lim_{n\to\infty} \frac{d(n)}{f(n)} = M$$

$$\lim_{n\to\infty} \frac{e(n)}{g(n)} < \infty$$
$$\lim_{n\to\infty} \frac{e(n)}{g(n)} = N$$

So, if we multiply these to together. we will get

$$\lim_{n\to\infty} \frac{d(n)}{f(n)} \times \frac{e(n)}{g(n)} = M \times N < \infty$$

Since $M \times N$ is$<\infty$,d(n)e(n)isO(f (n)g(n)).

(3) There is a function `void fnE(int i, int num)` that runs in $1000i$ steps,regard less of what num is. Consider the following code snippet:

```
void fnA(int S[]) {
     int n = S.length;
     for (int i=0;i<n;i++) { //  this is O(n)
         fnE(i, S[i]);            // Basically you do O(i) times
     }
}
```

Whats the running time in Big-O of fnA as a function of n, which is the length of the array S. You should assume that it takes constant time to determine the length of an array.

We see that the `fnE(i, S[i])` function runs $(1000)(i)$,we can run say that it is $O(i)$, and it is inside the for-loop, which we also know that it is $O(n)$. For n = 0,1,2,3,4...n `fnE(i, S[i])` runs$(1000)(0+1+2+3+4+...+n)$ times, we can write this in terms of $(1000)\frac{n^2+n}{2}$. Therefore, `fnA(i, S[i])` is $O(n^2)$.

Exercise3:: HowLongDoesThisTake? (2points)

For each of the following functions, determine the running time in terms of in the variable n. Show your work. Were more interested in the thought process than the final answer.

```
void programA(int n) {
     long prod = 1; // this takes constance timesO(1).
     for (int c=n;c>0;c=c/2) // O(logn).
          prod = prod * c; // this takes constance timesO(1).
}
```

Since we decrese each interation by the factor of $\frac{1}{2}$. We know that the loop will stop when $n \leq 1$, let say t is the number of times that we do the loop, so we could say that

$$\begin{aligned} \frac{n}{2^t} &= 1 \\ t &= \log_2 n \end{aligned}$$

So, we can say the total running time is Constance$(C) \times \log n$ Therefore this program is $\Theta(\log n)$.

```
void programB(int n) {
     long prod = 1; \\ takes O(1)
     for (int c=1;c<n;c=c*3) \\ takes O(logn)
          prod = prod * c;
}
```

Same way as the previous code, the funtion increase the interation by the factor of 3. The loop will stop when $c \geq n$. Lets t is the number of interation that this loob will perform.

$$\begin{aligned} 3^t &= n \\ t &= \log_3 n \end{aligned}$$

So, the loop performs $\log_3 n$ times. Therefore, this function is $\Theta(\log n)$.

Exercise 4: : Halving Sum (6 points)

Part II: (2 points)

```
public static double hsum(double[] X) {
     while (X.length>1){
          double[]Y = new double[X.length/2]; ----> k1*n/2
          for (int i = 0; i < X.length/2; i++) {
               Y[i] = X[2*i] + X[2*i+1]; ----> k2*n/2
          }
          X=Y; ---->k2
     }
return X[0];
}
```

At `double[]Y = new double[X.length/2];`, the work done at this line for is depending on how big is `double[] X` is ,so the work done is $k1 \times \frac{1}{2}$. Also the same way as The for-loop. we get $k2 \times \frac{1}{2}$, and lastly `X=Y` performs work $k2$. For the first interation.

$$k1(\frac{1}{2}) + k2(\frac{1}{2}) + k2 - -- > 1^{st}$$

Part III: (2 points)
We've already seen that if the code go by $\frac{1}{2}$, so this function is going to runlog n times. For many n interations. It will look like this. The assumpsion is $n = 2^k$,so

$$\begin{aligned} (k1 + k2)(n)(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + ... + \frac{1}{n}) + k2 \times \log_2 n \\ (k1 + k2)(n)(1 - \frac{1}{n}) + k2 \times \log_2 n \\ (k1 + k2)(n - 1) + k2 \times \log_2 n \end{aligned}$$

All in all, we can say that $k1 + k2)(n - 1) + k2 \times \log_2 n$ is the O(n).

Exercise 5: : Random Permutations (2 points)

1. Analyze the running time of `mkPerm` in $\Theta()$as a function of n.

```
public static int[] mkPerm(int n) {
     int[] perm = new int[n]; // O(n)
         Random rng = new Random();// O(1) = C

     for (int i=0;i<n;i++)//O(n)
          perm[i] = i+1;//c

     for (int i=0;i<n;i++) {// O(n)
     int t = rng.nextInt(i+1);//O(1) = C
           if (i!=t) {
                swp(perm, t, i); // C
                }
          }
return perm;
}
```

Therefore, this code is $\Theta(n)$.

2. Write a program to execute this algorithm. For each value of n, run it $T = 10$ or more times to obtain a good average. Complete the following table and make a plot (n vs. running time):

n	running time (average of T trials)
$100,000$	10.7535145
$200,000$	6.7806776
$400,000$	10.1913599
$800,000$	28.901040000000002
$1,600,000$	75.7735128
$3,600,000$	109.0172607
$6,400,000$	266.1274668

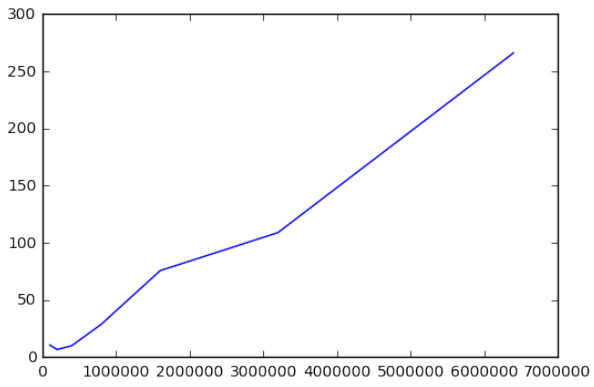

Figure 1: n vs. running time $h \to 0$.

3. Does the graph/data support the analysis you did earlier ? Discuss your findings.

It does not support my analysis, since I analysed this function as is $O(n)$, which mean that theoratically this function should have a linear graph. There is something going on more than what I get from doing the code. I assumed that the thing that makes this graph does not look linear is the java garbage collection.