Ali Mirabzadeh
CS180
HW1
Chapter 2
4.
Here is all the functions ordered in an ascending order:

$$n^{4/3} < 2^{\sqrt{\log n}} < 2^n < n^{\log n} < n(\log n)^3 < 2^{n^2} < 2^{2^n}$$

6.

a) This algorithm has an $O(n^3)$. The outer-most loop will run n times, the inner loop will run n times as well at its worst case. The storage operation that stores the result is one operation as well that will take n+1 time. Hence

$$f(n) = n * n * n+1 = n^3 + n^2 \text{ which result in } O(n^3)$$

b) Let's say we process this algorithm with n/4 elements. With these many elements the outer loop will run n/16 and inner one will run with the storage operation $n^2/16 + n/16$; therefore, the algorithm has a big-omega of n^3 which is the same as its big-O

c) I wrote the following algorithm that is more efficient, $O(n^2)$ and as oppose to the previous algorithm doesn't use unnecessary sources of inefficiency.

```
for i=1 , 2, …., n
        sum = i
        for j=i+1, i+2, …., n
                add A[j] into the sum
                Store the result in B[i, j]
        endfor
endfor
```

The two for loops will run each n times and the operations inside the inner loop would run at a constant time; therefore, both big-O and big-Omega is n^2

7.

I assume, n words needs l lines in which l = 1,2,…, l . Each line, l, has its own text bounded by c .
I write the following algorithm to sing the song where is the line number

```
For i=1,2,…,l
        For j= 1,2,..,i
                Sing the line j through 1
        endfor
endfor
```

Since we have l lines, n should be at least
1 +2 + … + l = l(l-1)/2 that means each line has word; hence,

$$l(l-1)/2 <= n$$
$$l <= 1 + (2n)^{(1/2)}$$

Therefore, $f(n) = O(n^{1/2})$

8.

a) Since k=2 that means we need to experiment once to see where the jar would break. If we are given n jars, I will take a square root of n then round it to the closest integer. I call it interval, interval = n^1/2, then starting from the first rung I'll drop the jar. Next rung will be current rung + interval. I keep dropping the jar in this interval till break one. When one got broken, I will go back to the previous rung in that interval, prevRung= currentRungThatJarBroke – interval. Then from that point I'll linearly move upward on each rung and drop the jar.
Hence, f(n) = n^1/2
$\lim_{n\to\infty} f(n)/n = 0$

b) I use the same algorithm, instead that I implement recursion for this part. In details when a jar is broken call the function again, get its square root and round it up. When get to the last jar, search linearly within the interval. This method follows the requirement as well.
$\lim_{n\to\infty} fk(n)/fk-1(n) = 0$