Daee kang
Cs 302-1004
Ass 2

1. Big Oh: $T(n)$ is in the set $O(f(n))$ if there exists two positive constants, c and n_0, such that $T(n) \le c f(n)$ for all $n > n_0$.
   Big Omega: For a non-negatively valued function $T(n)$, $T(n)$ is in set $\Omega(g(n))$ if there exists two positives constants, c and n_0, such that $T(n) \ge c\, g(n)$ for all $n > n_0$.
   Big Theta: $T(n) = \Theta(h(n))$ if and only if $T(n) = O(h(n))$ when $T(n) = \Omega(g(n))$
2. Big oh – upper bound for growth rate, worst case
   Big omega – lower bound for growth rate, best case
   Big theta – upper bound and lower bound are equal
3. Least to greatest: 730, 2/n, sqrt n, 2n, log n, n log n, n^2 log n, 4n^2, n^3, 5n^3, 2^(n/2), 2^n
4. O(log n)
   Binary search is done by repeatedly halving the length of the list through every iteration. This can also be seen as log 2 n. For a sequential search, the search runs by searching through every element, and in the worst case, it will search through every one at least one time making the complexity O(n).
5. First you would have to sort the given list and if using merge sort for example, it would be O(n log n). Then you would have to add the value in which you would have to search down the whole list or O(n). Simplified, it is O(n log n).
6. If the list is already sorted, it would just be a matter of searching down the list and in the worst case it would be O(n).
7. The best currently known algorithm runs in O((logn)^6) – AKS primality test.
8. T1 = O(1) + O(n^2)*O(1) + O(n^2)*O(1)
   T1 = O(n^2)
9. T1 = O(1) + O(log n^2)*O(1) + O(log n^2)*O(1)
   T1 = O(log n^2)
10. T1 = O(1) + O(n^2)*O(1)*O(1) + O(1) + O(1) + O(n log n)*O(1) + O(1) + O(1) + O(n)*O(1) + O(1) + O(1) + O(n^2)*O(1) + O(1) + O(1) + O(n)*O(1)*O(n)*O(1)+O(1) + O(1)+O(n^3)*O(1) + O(1) + O(n^4) + O(1) +O(1) + O(n^4) + O(1) + O(1) + O(n^2) + O(n) + O(1)
    T1 = O(n^2) + O(nlogn) + O(n) + O(n^2) + O(n) + O(n^2) + O(n) + O(n^3) + O(n^4) + O(n^4) + O(n^2)
    T1 = O(n^4)

11. T1 = O(n)
    T2 = O(log(n))
12. A) T1 = O(n^2)
        T2 = O(n)
    B) S1 = O(n)
        S2 = O(1)
13. A) T1 = O(n)
        T2 = O(n)
    B) S1 = O(n)
        T2 = O(1)
14. Space time trade off principle : the idea that algorithm will trade using more space for a more decreased runtime and vise-versa. The most recent example we used was assignment 1 where we were given a recursive function and a dynamic function. For example, Merge sort has a time complexity of O(n log n) while its space complexity is O(n). A majority of the sorts that are slower in run time have a space complexity of just O(1).
15. Recursive – O(2^n)
    Dynamic – O(n^2)