1. Order the following functions by growth rate. Indicate which functions grow at the same rate.

\[ n, \sqrt{n}, n^{1.5}, n^2, n \log n, n!, n \log \log n, n \log^2 n, n \log(n^2), 2/n, 2^n, 2^{n/2}, 37, n^3, n^n, n^2 \log n \]

2. An algorithm takes 0.5ms for input size 100. How long will it take for input size 500 if the running time is the following? What assumptions do you have to make in order to answer this problem?

(a) \( O(n) \)

(b) \( O(n \log n) \)

(c) \( O(n^2) \)

(d) \( O(n^3) \)
3. What is the big-Oh of \texttt{calc}, in terms of \( n \), the size of the array \( v \)?

\begin{verbatim}
int subcalc1(int[] v1)
{
    int sum = 0;
    for (int i=0; i < v1.length; i++)
        sum = sum + v1[i]*v1[i]*v1[i];
    return sum;
}

int subcalc2(int[] v2)
{
    int sum = 0;
    for (int i=0; i < v2.length; i++)
        for (int j=0; j < i; j++)
            sum = sum + v2[i]*v2[j];
    return sum;
}

int calc(int[] v)
{
    return subcalc1(v) + subcalc2(v);
}
\end{verbatim}

4. What is the big-Oh of \texttt{power2}, in terms of \( n \)?

\begin{verbatim}
int power2(int n)
{
    int prod = 1;
    while (prod < n)
        prod = prod * 2;

    return prod;
}
\end{verbatim}
5. The code in `SetThinking.java` is the basis for some of the following questions.

(a) The method `maxAndMin` returns the maximal and minimal element in parameter `list`—an array is used to return two values. How would the code change if we used `ArrayList`s instead of arrays? If the array has $N$ elements, what is the big-Oh complexity of the call `maxAndMin(list)`? It is one of $O(N)$, $O(N^2)$, $O(\log N)$ or $O(N \log N)$. Justify your answer briefly.

```java
public static String[] maxAndMin(String[] list) {
    String[] ret = new String[2];
    int maxIndex = 0;
    int minIndex = 0;
    for(int k=1; k < list.length; k++) {
        if (list[k] > list[maxIndex]) maxIndex = k;
        if (list[k] < list[minIndex]) minIndex = k;
    }
    ret[0] = list[minIndex];
    ret[1] = list[maxIndex];
    return ret;
}
```

(b) The method `counts` below returns the number of occurrences of `s` in `list`. If the array has $N$ elements, what is the big-Oh complexity of the call `counts(list, s)`? It is one of $O(N)$, $O(N^2)$, $O(\log N)$ or $O(N \log N)$. Justify your answer briefly.

```java
public static int counts(String[] list, String s) {
    int count = 0;
    for(int k=0; k < list.length; k++) {
        if (list[k].equals(s)) {
            count++;
        }
    }
    return count;
}
```
(c) The method `fastcounts` below returns the number of occurrences of `s` in `list`. Assume the `list` array is sorted so that the call to `Arrays.binarySearch` works (the complexity of `binarySearch` is $O(\log N)$ for an $N$-element array.) If the array has $N$ elements, what is the big-Oh complexity of the call `fastcounts(list,s)`? It is one of $O(N)$, $O(N^2)$, $O(\log N)$ or $O(N \log N)$. Justify your answer briefly — assume that no word occurs more than 25 times (this means that you don’t have to take the number of occurrences of a word into account when determining the complexity, just the size of the array.)

```java
public static int fastcounts(String[] list, String s){
    int count = 0;
    int index = Arrays.binarySearch(list, s);
    int first = index;
    int last = index;
    if (first < 0){
        return 0;
    }
    // s occurs at least once, find the range of occurrences
    while (0 <= first && list[first].equals(s)){
        first--;
    }
    while (last < list.length && list[last].equals(s)){
        last++;
    }
    return last - first + 1;
}
```
(d) The method `uniqueCounts` below returns an int array containing the number of occurrences of each unique element \( k \), where the \( k^{th} \) value in the returned array is the number of occurrences of the \( k^{th} \) different/unique word in `list`, where 0 is the alphabetically first word, 1 is the index of the next word alphabetically, and so on.

Assume that the process of creating a set from an array (see the code below) is \( O(N \log N) \) for an \( N \)-element array. What is the big-O complexity of the method `uniqueCounts` for an \( N \)-element array? It is one of \( O(N) \), \( O(N^2) \), \( O(\log N) \) or \( O(N \log N) \). Justify your answer briefly.

```java
public static int[] uniqueCounts(String[] list) {
    TreeSet set = new TreeSet();
    set.addAll(Arrays.asList(list));
    int[] ret = new int[set.size()];
    Iterator it = set.iterator();
    int index = 0;
    while (it.hasNext()){
        ret[index] = counts(list, (String) it.next());
        index++;
    }
    return ret;
}
```

(e) If the call to `counts` in the code above is replaced by `fastcounts` what is the big-Oh complexity of the function? It is one of \( O(N) \), \( O(N^2) \), \( O(\log N) \) or \( O(N \log N) \). Justify your answer briefly.

(f) Answer both of the previous two questions but use two values in giving the big-Oh complexity: \( N \), the number of values in the array, and \( K \), the number of different/unique values in the array. Assume the process of creating the set of unique values has complexity \( O(N \log(K)) \). Your answer should be expressed in terms of \( N \) and \( K \).