WarWithArray:

```
List 2k;
quicksort(s);

for each i in s
  for each j in s
    if(i != j)
      2k.add(i+j);

for each i in 2k
  for j = k+1 to i.length()
    temp = i.substring(j-k, j);
    if(!s.binarySearch(temp))
      2k.remove(i);
return 2k;
```

this starts with quick sort which has a time complexity of $n \log n$, it then constructs the list of all possible 2k length strings which takes n^2 time. Then it loops through the list of 2k strings which is $n(n-1)$ long, and for each of those it does $k-1$ binary searchs which takes $k(\log n)$ time since it takes k time to compare the strings. So the total time complexity is $n \log n + n^2 + n(n-1)(k-1)(k \log n)$

WarWithBST:

```
List 2k;
BST bs;

for each i in s
  bs.add(i);
  for each j in s
    if(i != j)
      2k.add(i+j);

for each i in 2k
  for j = k+1 to i.length()
    temp = i.substring(j-k, j);
    if(!bs.search(temp))
      2k.remove(i);
return 2k;
```

this starts with inserting the k length strings into a bst, the insertion takes up to $\log n$ time and there are n strings to insert, it also must create the list of $2k$ strings which takes n^2 time. then like before it loops through the $n(n-1)$ 2k strings and does up to $k-1$ searches through the BST which each take $k \log n$ time. The total time complexity ends up being $n \log n + n^2 + n(n-1)(k-1)(k \log n)$

WarWithHash:

```
List 2k;
HashTable hs;

for each i in s
  hs.add(i);
  for each j in s
    if(i != j)
      2k.add(i+j);

for each i in 2k
  for j = k+1 to i.length()
    temp = i.substring(j-k, j);
    if(!hs.search(temp))
      2k.remove(i);
return 2k;
```

This starts with inserting the k length strings into the hashtable, the insertion takes k time because the string needs to be hashed and there are n strings, and the list of $2k$ length strings is also made taking n^2 time. Again it loops through the $n(n-1)$ strings and does up to $k-1$ searches through the Hash Table each taking $2k$ time because the strings need to be hashed and java calls the equals method after the element is accessed to confirm its a match. The total time complexity ends up being $kn + n^2 + n(n-1)(k-1)(2k)$

WarWithRollHash:

```
List 2k;
HashTable hs;

for each i in s
  hs.add(hash(i));
  for each j in s
    if(i != j)
      2k.add(i+j);

for each i in 2k
  hashVal = hash(i, k);
  for j = k+1 to i.length()
```

```
        hashVal.increment();
        if(!hs.search(temp))
          2k.remove(i);
    return 2k;
```

This starts with inserting the k length strings into the hashtable, the insertion takes k time because the string needs to be hashed and there are n strings, and the list of $2k$ length strings is also made taking n^2 time. Again it loops through the $n(n-1)$ strings it does an initial k hash then for the $k-1$ searches it increments the hash taking constant time the accessing the hash still takes k time for each $k-1$ search because of the equals check. after the element is accessed to confirm its a match. The total time complexity ends up being

$$\begin{aligned} kn + n^2 + n(n-1)(k-1)(k) + n(n-1)k \\ kn + n^2 + n(n-1)k((k-1)+1) \\ kn + n^2 + n(n-1)2k \end{aligned}$$