1. Write a python function `passList(l)` which takes a list `l` containing pairs of *names* and *marks*. A student whose name appears in the list has passed if his mark is above 50 in each of the entries with his name. Otherwise he has failed. The function returns a list containing the names of those who have passed. Here are some examples:

```
passList([("Michael",51),("Madan",49),("Michael",59),("Rajan",56),
            ("Madan",53)] = ["Michael","Rajan"]
passList([("Michail",51),("Michael",49)] = []
```

2. Given a sequence of integers $a_1, a_2, \ldots, a_N$, the shortest distance of this sequence is defined as

$$Minimum(\{|a_i - a_j| \mid 1 \leq i < j \leq N\}$$

   Write a python function `shortestDist(l)` which takes a list of integers as input and outputs the shortest distance in the sequence defined by the list. You may assume that the list has at least 2 elements. Here are some examples:

```
shortDist([3,11,8,5,14,9,15]) = 1
shortDist([11,3,8]) = 3
```

   What is the complexity of your solution. Why?

3. Use dictionaries to count the number of occurrences of each word in a given string which begins with a capital letter. Here are some examples:

```
mycount("This is a Long Sentence. Length of this Long Sentence is not
     actually very Long") = {"This":1,"Long":3,"Sentence":2,"Length":1}
mycount("this is a short sentence") = {}
```

4. Write a Python function `word_count` which takes a string as input and outputs the words that occur in the string in the increasing order of their frequency of occurrence. If multiple words have the same frequency then they must appear in lexicographically increasing order. Here are some examples:

```
word_count('abcd abbcd abbcd ab bacd   cabd ab  abc ')
= ['ab', 'abbcd', 'abc', 'abcd', 'bacd', 'cabd']
word_count('''abcd abc abc
              abcd efg abc ''') = ['abc','abcd','efg']
```

5. Write a Python function `word_lengths` which works almost exactly as `word_count` except that it outputs the words in increasing order of lengths and words of the same length are output in lexicographically increasing order. So,

```
word_lengths('abcd abbcd abbcd ba bacd   cabd ba  abc')
= ['ba','ba','abc','abcd','bacd','cabd','abbcd','abbcd']
```

6. Write a Python function `detab` that takes the tab-length and a string and transforms it into one without tab characters as indicated above.

```
detab(5,"ab\tdef\t\th" ) =  "ab␣␣␣def␣␣␣␣␣␣␣h"
detab(5,"ab\td␣f\ng\th" ) =  "ab␣␣␣d␣f\ng␣␣␣␣h"
```

7. Let l be a list of strings over the letters $A-Za-z$. Write a Python function `bucket(l,i)` which returns a list, say ls constructed as follows: First transfer the elements of l that do not have position i (i.e. whose length is $< i+1$) to ls. Then transfer all the elements that have letter A at position i to the list ls. Then transfer those with the letter B at position i and so on till letter z. Here are some examples to illustrate how this function works:

```
bucket(["Abc","dEf","Baec","dA"],0) = ["Abc","Baec","dEF","dA"]
bucket(["Abc","dEf","Baec","dA"],2) = ["dA", "dEF","Baec","Abc"]
bucket(["Abc","dEf","Baec","dA"],3) = ["dA", "dEF","Abc", "Baec"]
bucket(["Abc","dEf","Baec","dA"],7) = ["Abc","dEf","Baec","dA"]
```

[You have to look up the Python documentation to see how to iterate over the list of characters $'A'...'z'$.]

Write a sorting function `BucketSort(l)` that sorts lists of strings over the alphabet $A-Za-z$ by repeatedly calling the function `bucket`.

What is the complexity of your function? Why ? (Add this as a comment at the end of your submission)