Efficient Algorithms: Suffix Array Construction

BruteForce:

```
import re


def getAllWords():
    all_words_cleaned=[]

    with open('./TomSawyerCompleteGutenberg.txt', encoding='utf-8-sig') as
f:
        for line in f:

            # split on space and line endings
            splittet_line=line.split()

            wordIndex = 0

            replacedSpecialChars = []
            for wordIndex in range(len(splittet_line)):
                # find all special chars and replace them with nothign
                newWord = re.sub("[^A-Za-z0-9]+", "",
splittet_line[wordIndex])
                # make all words lowercase
                newWord = newWord.lower()
                replacedSpecialChars.append(newWord)

                wordIndex+=1
            if(len(replacedSpecialChars)>0):
                for word in replacedSpecialChars:
                    all_words_cleaned.append(word)
    return all_words_cleaned

## generate suffix array
def createSuffixArr(all_words_cleaned):
    suffix_arr = []
    singleString = " ".join(all_words_cleaned)
    #limit string
    singleString = singleString[0:int(len(singleString)/4)]

    #generate suffix arr
    charIndex = 0
    for charIndex in range(len(singleString)):
            if(singleString[charIndex]== " "):
                continue
            suffix = singleString[charIndex:]
            suffix +="$"
            suffix_arr.append(suffix)
```

```
    return suffix_arr
##sort suffix array
import time

i = 0

runs = 5


allTimesTogehter = 0
while i < runs:
    start_time = time.time()


    allWords = getAllWords()

    suffix_array = createSuffixArr(allWords)
    suffix_array.sort()
    currentTime = time.time()-start_time
    allTimesTogehter += currentTime
    print("--- %s seconds ----" % (currentTime))
    i +=1

print("--- %s All Runs ----" % (allTimesTogehter/5))
```

Durchschnittliche Laufzeit bei 5 Durchläufen bei Brute Force(Bei Verwendung des ersten Viertel des Buches):

Ich konnte leider nicht den gesamten Text des Buches verwenden da dies zu einer out of memory Exception führt. Deswegen habe ich nur das erste viertel verwendet und bekomme dabei diese Zeit nach 5 Versuchen gemittelt heraus:

```
--- 39.259 All Runs ----
```

Eigene Implementierung

```
import re


def getAllWords():
    all_words_cleaned=[]

    with open('./TomSawyerCompleteGutenberg.txt', encoding='utf-8-sig') as
f:
        for line in f:
```

```
            # split on space and line endings
            splittet_line=line.split()


            wordIndex = 0

            replacedSpecialChars = []
            for wordIndex in range(len(splittet_line)):
                # find all special chars and replace them with nothign
                newWord = re.sub("[-]+", "", splittet_line[wordIndex])
                # make all words lowercase
                newWord = newWord.lower()
                replacedSpecialChars.append(newWord)

                wordIndex+=1
            if(len(replacedSpecialChars)>0):
                for word in replacedSpecialChars:
                    all_words_cleaned.append(word)
    return all_words_cleaned

## generate suffix array

def createSuffixArr(all_words_cleaned):
    suffix_arr = []



    singleString = " ".join(all_words_cleaned)


    #limit string
    singleString = singleString[0:int(len(singleString)/4)]

    #generate suffix arr
    charIndex = 0
    for charIndex in range(len(singleString)):
            if(singleString[charIndex]== " "):
                continue
            suffix = singleString[charIndex:]
            suffix +="$"
            suffix_arr.append(suffix)



    return suffix_arr
```

```
def quick_sort(s):
    if len(s) == 1 or len(s) == 0:
       return s
```

```
    else:
        pivot = s[0]
        i = 0
        for j in range(len(s)-1):
            if s[j+1] < pivot:
               s[j+1],s[i+1] = s[i+1],s[j+1]
               i += 1
        s[0],s[i] = s[i],s[0]
        first_part = quick_sort(s[:i])
        second_part = quick_sort(s[i+1:])
        first_part.append(s[i])
        return first_part + second_part

##sort suffix array
import time

i = 0
runs = 5
                            allTimesTogehter = 0
while i < runs:
    start_time = time.time()


    allWords = getAllWords()

    suffix_array = createSuffixArr(allWords)
    # suffix_array.sort()
    quick_sort(suffix_array)
    currentTime = time.time()-start_time
    allTimesTogehter += currentTime
    print("--- %s seconds ----" % (currentTime))
    i +=1

print("--- %s All Runs ----" % (allTimesTogehter/5))
```

Beschreibung der Optimierungs Idee:

Da zuvor list.sort() verwendet wurde, habe ich mit etwas eingelesen und herausgefunden dass Python Timsort verwendet. Deshalb habe ich versucht einen anderen sorting algo zu verwenden. Die Wahl ist dann einfach auf Quicksort gefallen und habe diesen ausprobiert.

Durchschnittliche Laufzeit(Bei Verwendung des ersten Viertel des Buches), 5 Versuche:

```
--- 28.204 All Runs ----
```

Argumentation warum Ihr Algorithmus praktische schneller/langsamer ist, als Brute-Force.

Timsort liefert eine bessere Performance bei „leicht“ vorsortieren Daten. Deshalb wird Timsort auch in vielen realen Beispielen verwendet da eine große Anzahl der Daten bereits natürlich etwas vorsortiert sind. In unserem Fall sind jedoch die Suffixe wenig vorsortiert weshalb dieser Algorithmus eine bessere Performance bei mir liefert.