APS
Assignment 2

Q1:Implementation of suffix array

Implement a Suffix Array that is capable of performing following operations on Strings in a most efficient way .

1. Given a string S print its minimum lexicographic rotation. ***O(nlogn)***
2. Given an integer K , print the length of the longest substring that appears in the text at least K times.If no such substring exist, print -1. ***O(nlogn)***
3. Given a strings S determine its longest substring that is also a palindrome. In case of multiple solutions, print the lexicographically smallest palindrome. ***O(nlogn)***

Evaluation Criteria :- You will be given a large string S (length <= 100000) , and one of the above case. Print the corresponding output.
String consist of either Lower/Upper Case alphabet and Numeric digits.

Note : For each sub part implement a different code. Submit it as Q1a_rollnumber.cpp, Q1b_rollnumber.cpp, Q1c_rollnumber.cpp

Example :

S = “dcabca”

1. All possible rotation are “dcabca” , “cabcad” , “abcadc” , “bcadca” , “cadcab” , “adcabc” . Among all lexicographically minimum is “abcadc” .
2. If K=2 than since “a” is only substring that appears twice and it’s length is 1, so answer is 1.
3. Since only length 1 substring are palindromic, and among them “a” is lexicographically smallest, hence answer is “a”.

Q2:Implement a B-Tree for integers

B-Trees are self balancing search trees like AVL, Red-Black trees. But unlike those, in B-Trees we can store more than 1 value in a node and each node can have more than 2 children.
Since each node stores relatively large number of keys, B-Trees becomes well suited for secondary storage where we read data in large chunks. Similarly this makes B-Trees more cache efficient than BSTs.

Implementation :

Your task is to make a in memory B-Tree (We do not expect a disk based implementation). Maximum number of keys per node should be kept configurable (atleast something which can be changed before compilation).
And you have to implement following operations for it -
Insert : worst case O(log n)
Delete : worst case O(log n)
Search : worst case O(log n)
Input format :
Q queries of following format -
1 x - insert x
2 x - search x
3 x - delete x

Resources :
https://en.wikipedia.org/wiki/B-tree
http://btechsmartclass.com/DS/U5_T3.html
https://www.cs.usfca.edu/~galles/visualization/BTree.html (visualization for B Trees)

Q3:Nth no. of a unsorted array

Task: To find the nth smallest element in a given sequence. The operation should take an amortized cost of O(n).

Aim: To learn how to use randomization in algorithms.

Hint: Think of the partition function of randomized quick-sort.

Testing: Time your function using time.h and compare it with the Standard Library Function nth_element().

Bonus Read: Read about how std::partial_sort() works and when how and when it used.

References:
https://en.cppreference.com/w/cpp/algorithm/partial_sort
https://en.cppreference.com/w/cpp/algorithm/nth_element

Q4:Hashing(Unordered_map)

Task: Implement a hash table as data structure which insert, delete, find element in amortized constant time.

Aim: To learn how Hashing works and importance of Hash Functions. Also look how Universal Hashing is implemented.

Paramter to Judge: Time and space complexity

Note:If the map is not generic then it should atleast work for string and integers.

References:
https://en.wikipedia.org/wiki/Hash_function
https://en.wikipedia.org/wiki/Universal_hashing

Q5:Implement Java StringBuilder

- Java StringBuilder class is mutable sequence of characters. StringBuilder Class can be comparable to String however the StringBuilder class provides more versatility because of its modification features.

- You are required to implement a library which supports following functionalities:
    1. Initialize a string   O(1)
    2. Append two string   O(1)
    3. Find substring in string   O(n)

- What is expected?
  Implement a library which provides following interface.

```
int main(){
  stringBuilder s1 = stringInitialize("hello");
  stringBuilder s2 = stringInitialize("world");
  int index1 = findSubstring(s2,"or");
  // index1 will have value 1. Starting index of substring

  int index2 = findSubstring(s2,"hell");
  // index2 will have value -1. No substring found

  stringBuilder s3 = stringAppend(s1,s2);
   // s3 will become "helloworld". Append string in second      argument to string
in first argument.
}
```

NOTE: You are not allowed to use STLs or any other inbuilt libraries.