CSCI-311 **Project 1**

The Guardians of the Galaxy requested decipher support. They got hold of an important enciphered message that would allow them to track down a hidden treasure in the Galaxy. They heard about the famous Wild Cat programmers and sought their assistance.

There are two files of text: the first file serves as the key to the other file, which is enciphered text. To decipher the second text, one needs to concatenate all lines of the first text into a single long string *S*, sort the suffixes of *S* in alphabetical order and use the starting positions of the sorted suffixes as deciphering keys to unlock the secret of the second text. If one puts the words in the second text using the ordering of the suffixes in the first text, the message will be revealed.

Example of Two Input Files:

```
Abra
Cadabra
```

```
what
are
plans
happens
busy
when
making
is
you
other
life
```

When we concatenate all lines from the first text, the resulting string S is "AbraCadabra".

After sorting suffixes of *S* in alphabetical order, we will obtain ordering of the starting positions of the suffixes:

10	a
7	abra
0	abracadabra
3	acadabra
5	adabra
8	bra
1	bracadabra
4	cadabra
6	dabra
9	ra
2	racadabra

Now use this ordering to put the words from the second text in the same order as the order of the starting positions of the suffixes from *S.*

0	what
1	are
2	plans
3	happens
4	busy
5	when
6	making
7	is
8	you
9	other
10	life

10	a
7	abra
0	abracadabra
3	acadabra
5	adabra
8	bra
1	bracadabra
4	cadabra
6	dabra
9	ra
2	racadabra

10	life
7	is
0	what
3	happens
5	when
8	you
1	are
4	busy
6	making
9	other
2	plans

Now, we can read the deciphered message: "*life is what happens when you are busy making other plans*".

Example of Output File:

```
life is what happens when you are busy making other plans
```

Part 1.

Your program will be run using the following *command line arguments***:**

./main file1.txt file2.txt

1. Your program needs to read in the first file and concatenate all the lines into a single string *S*.
2. Then your program will read the second file and put all the words into a vector of strings.
3. Then you need to write a function that in constant time will convert a given character into the corresponding lower-case character (do not use C++ function for this).

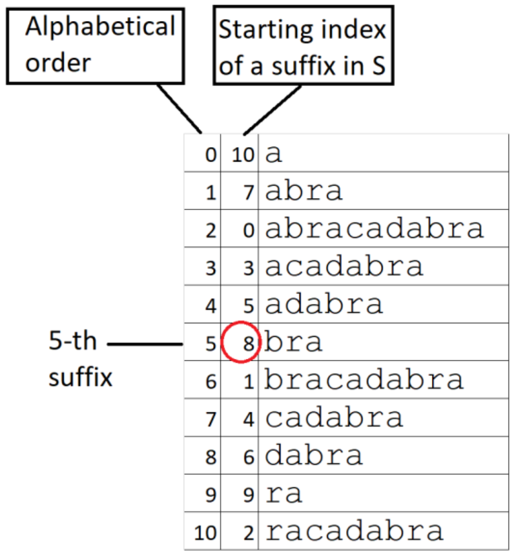
   In this part, your program will sort all the suffixes of S using *QuickSort.* Instead of sorting explicitly separate suffixes of *S*, to save space, use a vector of integers, the starting positions of suffixes, that will be passed to *QuickSort* along with the string S. So, when you need to compare two suffixes, use their starting positions together with the string *S* to compare these suffixes (compare suffixes one character at a time from the starting positions to the end of *S*). Whenever comparing two characters of two suffixes, convert the characters to lower-case before comparing to each other. Because we need to use alphabetical sort, in which characters 'A' and 'a' should be the same, and C++ uses ASCII representations of the characters for comparison, we need to convert both upper-case and lower-case characters to the same representation before comparing them.

   Then your program will use the starting positions of the sorted suffixes as the indices of the strings from the second file, to put the words from the second file in the correct order. After this, the deciphered message can be printed out using ***cout*** (place a space after each word, and ***endl*** at the end of the message).

Part 2.

1. In this part, you need to re-write the provided code for ***Selection*** algorithm to work with suffixes. Given a string ***S***, and an integer ***k***, your program must return the starting index of the suffix that is the ***k***-th suffix of ***S*** if the suffixes of ***S*** were sorted in alphabetical order. Remember that ***Selection*** algorithm does not actually sort the items in a given array.

For example, given a string S = "abracadabra" and k = 5, your program will return 8, the starting position of the suffix "bra", that is the 5-th in the alphabetical order (the count of ordering starts with 0).


Alphabetical order
Starting index of a suffix in S
5-th suffix

0	10	a
1	7	abra
2	0	abracadabra
3	3	acadabra
4	5	adabra
5	8	bra
6	1	bracadabra
7	4	cadabra
8	6	dabra
9	9	ra
10	2	racadabra

2. Next you will re-write ***Insertion*** sort algorithm to sort suffixes of a given string S. Similarly to QuickSort, your ***insertion*** function will take a string S (passed constant by reference), and a vector of integers with the starting positions of suffixes of S. The function will sort the suffixes of S by calling ***lessThan*** function (from Assignment 3) and swapping positions of the suffixes instead of suffixes.

3. **Experiment 1.** You will compare the performance of the two sorting algorithms: QuickSort and Insertion.

- Read a test file, concatenate all lines into a single string S;
- Then make a separate copy of a vector of integers with the suffix indices for each algorithm (the length of a vector equals to the length of S). In other words, you will have two vectors ***A*** and ***B***, such that A[i] = B[i] = i. You will pass A to QuickSort and B to Insertion.
- Measure time that it takes to run QuickSort;
- Measure time that is required to run Insertion.
- Output the results to the screen in the format: length of S, time for Quicksort, time for Insertion.
- Record this information for all five input files.
- Build a graph using Excel (or a similar tool) that shows the length of S on x-axis and time in seconds on y-axis.

4. **Experiment 2.** Next, you will compare the performance of *QuickSort* and *Selection* algorithms to accomplish the following task. Assume that we only need to know the starting position (index) of the k-th suffix of a given string S if all suffixes of S were sorted in alphabetical order. Selection accomplishes this task in linear time, but QuickSort also can accomplish this task by first sorting the suffixes of a given S in alphabetical order, and then return the index of the k-th suffix.

- ➢ Read a test file, concatenate all lines into a single string S;
- ➢ Then make a separate copy of a vector of integers with the suffix indices for each algorithm (the length of a vector equals to the length of S). In other words, you will have two vectors ***A*** and ***B***, such that A[i] = B[i] = i. You will pass A to QuickSort and B to Selection.
- ➢ Measure time that it takes to run QuickSort;
- ➢ Measure time that is required to run Selection.
- ➢ Output the results to the screen in the format: length of S, time for Quicksort, time for Selection.
- ➢ Record this information for all five input files.
- ➢ Build a graph using Excel (or a similar tool) that shows the length of S on x-axis and time in seconds on y-axis.

5. **Experiment 3.** Finally, you will compare the performance of ***Selection*** and ***Insertion*** to sort suffixes of a given string S in alphabetical order. To use ***Selection*** algorithm to sort suffixes, we can simply write a **for** loop that at each iteration calls *Selection* algorithm to find ***i***-th suffix, and will write down the returned suffix.

- ➢ Read a test file, concatenate all lines into a single string S;
- ➢ Then make a separate copy of a vector of integers with the suffix indices for each algorithm (the length of a vector equals to the length of S). In other words, you will have two vectors ***A*** and ***B***, such that A[i] = B[i] = i. You will pass A to Inserrtion and B to Selection.
- ➢ Measure time that it takes to run Insertion to sort all the suffixes of S;
- ➢ Measure time that is required to run Selection to sort all the suffixes of S;
- ➢ Output the results to the screen in the format: length of S, time for Insertion, time for Selection.
- ➢ Record this information for all five input files.
- ➢ Build a graph using Excel (or a similar tool) that shows the length of S on x-axis and time in seconds on y-axis.

Main.cpp must include: The file *main.cpp* must include the code for Insertion and Selection and QuickSort in it, as well as main() function that calls these algorithms.

- If ***argc*** equals to 2, then argv[1] is the input file, run **Experiment 1.**
- If **argc** is 3, then argv[1] is the input file, and argv[2] is the parameter ***k*** that you need to pass to *Selection* algorithm; run **Experiment 2.**
- If argc is 4, then argv[1] is the input file, and two other command line arguments are just dummy arguments, so you know that you need to run **Experiment 3.**

Write a one-two pages report (printed) where you will discuss your three experiments: (1) Comparison of QuickSort and Insertion to sort all the suffixes of a given string S; (2) Comparison of QuickSort and Selection to find the k-th suffix taken in alphabetical order of a given string S; and (3) Comparison of Insertion and Selection to sort all the suffixes of a given string S.

Outline of your report (for each experiment):

1) State theoretical asymptotic time analysis of each algorithm (best and worst cases);
2) Space analysis of each algorithm as a function of *n*, the length of S;
3) A graph with experimental results;
4) Conclusion: which algorithm performs better in terms of space and time.

How to measure time in seconds inside a program:
Include <ctime> header.
Assume that you need to measure how long does it take to run function *partition*, then you need to declare two variables of type ***clock_t***, place one before the function, and the other after the function:

```
clock_t t1 = clock();
partition(… parameters …);
clock_t t2 = clock();
double elapsed = double(t2 – t1)/CLOCKS_PER_SEC; //elapsed is the time in seconds
cerr << "Time to run partition is " << elapsed << " seconds." << endl;
```

Test files for Part 2: download **time_tests.tar**. For Experiment 2, use the following command line arguments:

Input File, parameter k
chrY_50.txt 1500
chrY_100.txt 5000
chrY_150.txt 8500
chrY_200.txt 100
chrY_250.txt 8000

Submission:

1. Submit your code, the file *main.cpp*, for Part 1 to *Project1* on turnin.

2. Submit your code, *main.cpp*, for Part 2 to *Project1_part2* on turnin. Note there are no test files on turnin. I just use this to collect your code and check this manually.

3. Submit your printed report in person by the deadline in class, and a copy of the report on Blackboard to Project1.

Grading:

Project 1 is worth 500pts: (1) 200pts for Part 1, (2) 200pts comes from correctly implementing Insertion and selection (Assignment 4), and (3) 100pts for the report.