# Data Structures – Spring 2023
# Project 1
# Due: February 6th 2023 at 11:59 PM

## Objective:

In this project you will learn about reading data from input files and store and play with dynamic arrays with sorting algorithms.

## Description:

When given *n* numbers as input, there are many operations/questions/queries that you can perform/ask about the *n* numbers. The time to answer these queries or perform the operations vary depending on how the numbers are spread out in the given order – pattern in which they are given. One way to speed up the query time is to vary the pattern of these given numbers to an orderly fashion – increasing or decreasing order.

Given a sequence of *n* numbers as input in no particular order, the goal is to re order the numbers for an output where the numbers are all in increasing or decreasing order. This is the definition for ***Sorting*** a set of *n* numbers.

Once we sort the given numbers into an order, the queries, or operations that we can perform can be faster (at least for a specific set of them if not all operations/queries). For example, once we sort, finding the median of the n numbers can and will be faster but time to find the mean will not change.

You will be **comparing the time taken** for performing queries and operations between an unsorted set of random numbers and performing the same query/operation on a sorted version of the same set of numbers.

You are given a set of randomly generated numbers that is first stored in a dynamic array (A). Once this is stored, you will have to sort this array and store the sorted array in a different array (B). You can use any sorting algorithm to sort (we will be providing you with a sorting algorithm as well).

### Queries:

- **Find/search** the given element in both the sorted and unsorted arrays.
    - In array B (sorted array), you can use **binary search** to find the value.
    - In array A, you must do a **linear search**.
- **SumPairs:** Given an integer *target*, return the two numbers such that they add up to the *target*.

### Operations:

- **Remove** the given element from both the arrays A and B.

    - Once you remove the element, you must left shift the values to the right of the empty space so that the empty spaces as we remove more elements are maintained to the right end of the array.
- **Insertion:** Insert the given element into both the arrays A and B.
    - In array B (sorted array), you must insert the value into the right position.
    - In array A, you may insert it at the end of the array.

Assumptions:

- Number of removals will be always less than the number of insertions. So there is no need to reallocate the arrays.
- There will not be any duplicates in the given set of random numbers.
- Keep up with the FAQ discussion board for project 1 on canvas as more assumptions and conversations will be continued there.

## Input explanation:

The input file given is in the following format:

```
9     ← size of the array

40
30
50
10
20
60
80
90
70

F 5   ← number of find queries

20
30
50
60
45

A 3   ← number of SumPairs queries

30
70
120

R 4   ← number of remove operations
```

```
40
30
60
70

I 3  ← number of insertion operations

55
45
75
```

The input file starts with the number of values (9) that are. Then we list all the randomly generated numbers.
After this, we give you a list of commands/options for the operations and queries that go till the end of the input file. Each of these commands are followed by the number of commands for each of the operations/queries. Commands can either be F, R, I or A:

- F – For the find query mentioned above followed by the number of find queries. This is followed by all the values with which the find query is to be performed in arrays A and B.
- A – For the SumPairs query mentioned above followed by the number of targets. This is followed by all the target values with which the query is to be performed in arrays A and B.
- R – For the remove operation mentioned above followed by the number of operations. This is followed by all the values to be removed in arrays A and B.
- I – For the insert operation mentioned above followed by the number of operations. This is followed by all the values to be inserted in arrays A and B.

To get you all started, a **boiler plate file** is given as well. This file is also given alongside this project. All other operations and queries have to be written as functions above the main function.

## Redirected input:

Redirected input provides you a way to send a file to the standard input of a program without typing it using the keyboard. To use redirected input in Visual Studio environment (on windows), follow these steps: After you have opened or created a new project, on the menu go to project, project properties, expand configuration properties until you see Debugging, on the right you will see a set of options, and in the command arguments type <"input filename". The < sign is for redirected input and the input filename is the name of the input file (including the path if not in the working directory).

If you use macOS or linux (or windows using powershell), you may use the terminal to compile and run your program using g++. This is the compiler that we use in the autograder on GradeScope (you may need to install g++ on your computer). Once you installed g++, you may compile your program using the following command:

```
g++ project1.cpp -o p1
```

You may then run the program with the redirected input using the following command:

```
./p1 < input1.txt
```

Make sure your program and your input file are in the same folder.

You may also use Google CoLab as an alternative and a sample file is given along with the project. This is an ipynb file which has to be opened in Google CoLab.

A simple program that reads the input file (one number) and displays everything that was read is given below.

```
#include<iostream>
using namespace std;

int main()
{
  int num = 0;
  cout << "Hello World" << endl;
  cin >> num;
  cout << num;
  return 0;
}
```

Sample output file for corresponding input files will be released. The input1.txt file given is a simple input file that will help you understand the project, more complicated ones will be released later and used for grading.

## Report and timing:

Once you are done coding all the queries and operations to get to the expected output, you will then have to run experiments individually to get the timing of the various queries and operations. You may use the clock() function to record the start and the end. Below is how to use it.

```
// start timer for insertion
clock_t start1 = clock();

// perform the operations or the queries for array A

// end timer for execution
clock_t end1 = clock();
```

You will have to submit a single page report along side the source code comparing the performance between the arrays A and B for each operation and query. You need to change the input file and vary the size of the array.

## Submission:

Submission will be through GradeScope. Your program will be autograded with test cases and then hand graded to check for documentation and if you have followed the specifications given. You may submit how many ever times to check if your program passes the test cases provided.

Test cases will be released at the beginning and later other test cases will be released while grading. For the autograder to work, the program you upload must be named as **project1.cpp**.

Your final submission must contain your source file named **project1.cpp** and a one page report with experiment and plot named **project1_report.pdf**. You access GradeScope using the tab on the left in our course page on canvas.

## Constraints:

1. In this project, the only header you will use the header files that are given to you in the boiler plate code. Using other or excess header files will be subject to a heavy grade penalty for the project.
2. None of the projects is a group project. Consulting with other members of this class on programming projects is strictly not allowed and plagiarism charges will be imposed on students who do not follow this.
    - Please review academic integrity policies in the modules.

## How to ask for help:

1. Check FAQ discussion board of the project first.
2. Email the TAs with your precise questions.
3. Upload your well commented and clean program to CodePost.
    a. This is a website which is used to share code with your TA only.
    b. You will upload your program and I can view it and comment on it.
    c. Here is the invite link to our class for the summer.
    d. Once you join the class on CodePost, you can upload your program to the Project 1 assignment.

        Note: Your program will **not** be auto graded at CodePost, this is just for when you want to ask a question and a place where I can look at your program and comment on it. CodePost is great for live feedback. GradeScope is the place where you should submit and where your program will be autograded.

***"Computer manufacturers of the 1960s estimated that more than 25% of the running time on their computers was spent on sorting... in fact, there were many installations in which the task of sorting was responsible for more than half of the computing time." - Donald Knuth.***