# Assignment 3 Design

Nhan Nguyen

January 2022

## 1 Introduction

### 1.1 Program Description

- The purpose of this program is to generate random numbers, insert them into an array and sort that array using different kind of sorting algorithms, the sorting algorithms in use are:

- Insertion Sort
- Quick Sort
- Heap Sort
- Batcher Sort

- Additionally, there's also a statistics module in order to show how each sorting algorithm perform in terms of the number of moves and comparisons
- **Note: Since the sorting algorithms are already provided in the lab document, I will not go into details of each sorting algorithms in this Design document**

### 1.2 List of Files

- **batcher.c:** The implementation of Batcher Sort
- **batcher.h:** The interface to batcher.c
- **quick.c:** The implementation of Quick Sort
- **quick.h:** The interface to quick.c
- **insert.c:** The implementation of Insertion Sort
- **insert.h:** The interface to insert.c
- **heap.c:** The implementation of Heap Sort
- **heap.h:** The interface to heap.c

- **stats.c:** The implementation of the statistic module

- **stats.h:** The interface to stats.c

- **set.h:** The interface for set ADT

- **sorting.c:** Contains main() and get the commands to do certain sorting algorithms as well as manipulating the randomized array.

- **Makefile:** This file formats program into clang-format and also compiles it. In addition, this makefile will also link batcher.c, quick.c, heap.c, insert.c, sorting.c, and stats.c

- **README.md:** A text file in Markdown format that describes how to build the program, as well as describing the problems encountered while making the program

- **DESIGN.pdf:** This PDF file describes the idea behind the program with visualization and pseudocode

- **WRITEUP.pdf:** This PDF file contains all the graphs displaying the performance of each sorting algorithm, and also an analysis and lessons learned from sort graphs

# 2 Generating the array

## 2.1 Generating random numbers

- At first, you might think that the numbers generated are just from **rand()**. However for this particular program, we will set a seed in order to generate our random numbers, that way we can do multiple sorts on the same array without using much more memory by making another array. To set a seed for our randomly generated numbers we will be using **srandom()**

- Furthermore, the numbers generated will be bit-masked to fit in 30 bits, therefore the numbers generated will always be smaller or equal to $2^{30} - 1$. To do this we will use the operator "&" (And)

## 2.2 Dynamically Allocate the Array

- Since we can customize the size of the array, and the size of our array must go up to the limit of a computer, we must be able to dynamically allocate the array. We can do so with the command **malloc()**. Let say we need an array with size **n**, we can dynamically allocate $n \cdot 4$ bytes to accommodate for **n** elements that will go into that array

### 2.3 Pseudocode

- **Initialize the Dynamically Allocate array:**
unsigned 32 bit number pointer *arr = NULL
arr = malloc(**size of array** * sizeof(unsigned 32 bit number))

- **Set seed, and Generate Array**
srandom(**an integer for the seed**)
for i from 0 to **array size**
    arr[i] = (**random number** &$2^{30} - 1$)

## 3 Identify Enabled Algorithms

- Since we are sorting an array using multiple sorting algorithms, it's going to be handy to know which ones are enabled and ready for use. For that we will use a **Set**, a **Set** is a bit-wise number where each bit is in 1 of the 2 states:

- 1: The bit is on
- 0: The bit is off

Using a set, we can determined which kind of sorting algorithm is enabled by assigning a certain bit to one of the four sorting algorithm. In my case, the bits I chose are:

- 1st bit ($2^0$): Determine whether Heap Sort is enabled
- 2nd bit ($2^1$): Determine whether Batcher Sort is enabled
- 3rd bit ($2^2$): Determine whether Insertion Sort is enabled
- 4th bit ($2^3$): Determine whether Quick Sort is enabled

After setting the four bits to be either on or off (1 or 0) through getopt() in main(), we can check which bit is on by using a simple for loop

## 4 main()

### 4.1 Idea

In main(), there's a function that accepted commands for the command line by using getopt():

- a: Enables all sort
- b: Enables Batcher Sort
- h: Enables Heap Sort
- i: Enables Insertion Sort

- q: Enables Quick Sort
- r seed: Set the seed for the randomly generated numbers (Default: 13371453)
- n size: Set the generated array's size to **size** (Default: 100)
- p elements: Print **elements** number of elements in the array (Default: 100), if **elements** is larger than the size of the array, the entire array is printed
- H: Display synopsis and how to run the code

To implement these commands, it's critical to use a getopt() command by including a **unistd.h**.
Enabling Sorting algorithms is already discussed in Section 3
Generating a randomized array is already discussed in Section 2

## 4.2 Pseudocode

*Note that the following code below will include the code portion of both section 2 and 3 to create a complete main() function*

```
seed = 13371453
size = 100
elements = 100
Set = emptyset //all bits are 0

get command
if command is a:
    set first bit in Set to 1
    set second bit in Set to 1
    set third bit in Set to 1
    set fourth bit in Set to 1
if command is b:
    set second bit in Set to 1
if command is h:
    set first bit in Set to 1
if command is i:
    set third bit in Set to 1
if command is q:
    set fourth bit in Set to 1
if command is r:
    read number
    seed = number
if command is n:
    read number
    size = number
if command is p:
```

```
    read number
    elements = number
if command is H:
    display synopsis and usage
    exit the program
for i from 1 to 4:
    if i-th bit == 1:
        srandom(seed)
        for j from 0 to size:
            arr[j] = rand() & (2^30 − 1)
        if i == 1:
            do Heap Sort and print statistics
        if i == 2:
            do Batcher Sort and print statistics
        if i == 3:
            do Insertion Sort and print statistics
        if i == 4:
            do Quick Sort and print statistics
        print = min(elements, size)
        for j from 0 to size:
            print the array element arr[j]
```