CSE 13s Assignment 3 Design

caowen

February 2023

1 Description of Program

This program test 4 different sorting algorithms on a seeded pseudorandom array or integers. The program implements the Quick sort, Shell sort, Batcher sort, and Heap sort. The program also keeps track of statistics for each sorting algorithm, including how many moves they make and how many comparisons are made. A move is defined by any time an element in the array is moved to a different position, and a compare is defined by any time 2 elements from the array are compared to each other. These statistics are printed when the algorithms are executed.

2 Files to be included in directory

- batcher.c
  - File that contains the batcher sort algorithm
- batcher.h
  - Header file that implements the interface for batcher.c
- gaps.h
  - Header file that contains an array of "gaps" (integer values) which are required for the shell sort algorithm
- heap.c
  - File that contains the heap sort algorithm
- heap.h
  - Header file that implements the interface for heap.c
- quick.c
  - File that contains the quick sort algorithm

- quick.h
  - Header file that implements the interface for batcher.c
- shell.c
  - File that contains the shell sort algorithm
- shell.h
  - Header file that implements the interface for shell.c
- stats.c
  - File that contains the implementation of the Stats struct, which keeps track of algorithm statistics
- stats.h
  - Header file that implements the interface for stats.c
- set.c
  - File that contains the implementation a set struct, used in sorting.c for argument management
- set.h
  - Header file that implements the interface for set.c
- sorting.c
  - File that contains the main() function and handles the creation of the pseudorandom array to be sorted as well as each sorting algorithm and statistic printing.
- Makefile
  - File that compiles the program as well as allows it to be formatted and cleaned
- README.md
  - File that contains instructions on how to run the program
- DESIGN.pdf
  - This file, contains the design process of the program and details each algorithm implemented
- WRITEUP.pdf
  - File that contains further analysis of the program and sorting algorithms

3 Pseudocode

- Batcher Sorting Algorithm

```
n = length of array
t = bit length of n
p = 1 << (t - 1)
while(p > 0):
    q = 1 << (t - 1)
    r = 0
    d = p
    while(d > 0):
        for(i = 0; i < n - d; i++):
            if((i bitwise and p) == r):
                if(array[i] > array[i + d]):
                    swap array[i] and array[i + d]
        d = q - p
        q = q >> 1
        r = p
    p = p >> 1
```

- Heap Sorting Algorithm

```
int max_child(array arr, int first, int last):
    left = 2 * first
    right = left + 1
    if(right <= last and arr[right - 1] > arr[left - 1]):
        return right
    return left

void fix_heap(array arr, int first, int last):
    found = False
    mother = first
    great = max_child(arr, mother, last)
    while(mother <= last // 2 and !found):
        if(arr[mother - 1] < arr[great - 1]):
            swap arr[mother - 1] and arr[great - 1]
            mother = great
            great = max_child(arr, mother, last)
        else:
            found = True


void build_heap(array arr)
    for(int father = last // 2; father > first - 1; father–):
        fix_heap(arr, father, last)
```

```
heap_sort(array arr):
    first = 1
    last = length of arr
    build_heap(arr, first, last)
    for(int leaf = last; leaf > first; leaf–):
        swap arr[first - 1] and arr[leaf - 1]
    fix_heap(arr, first, leaf - 1)
```

- Quick Sort Algorithm

```
int partition(array arr, int lo, int hi):
    i = lo - 1
    for(int j = lo; j < hi; j++):
        if(arr[j - 1] < arr[hi - 1]):
            i++
            swap arr[i - 1] and arr[j - 1]
    swap arr[i] and arr[hi - 1]
    return i + 1

void quick_sorter(array arr, int lo, int hi):
    if(lo < hi):
        p = partition(arr, lo, hi)
        quick_sorter(arr, lo, p - 1)
        quick_sorter(arr, p + 1, hi)

quick_sort(array arr):
    quick_sorter(arr, 1, length of arr)
```

- Shell Sort Algorithm

```
shell_sort(array arr):
    for(gap in gaps):
        for(int i = gap; i < length of arr; i++):
            j = i
            temp = arr[i]
            while(j >= gap and temp < arr[j - gap]):
                arr[j] = arr[j - gap]
                j -= gap
            arr[j] = temp
```

- sorting.c

```
int main(int argc, char **argv):
    stats = new Stats
    seed = 13371453
    size = 100
    elements = 100
```

set = empty set

loop through args
    if(arg is a sorting algorithm arg):
        add respective value to set
    else:
        modify seed, size, or elements, depending on arg
for(sort in sorting algorithms which are being run):
    recreate arr using the seed, size, and stats
    sort(arr)
    print results
    reset stats
free allocated memory

- set.c

    Set set_empty(void):
        return 0

    Set set_universal(void):
        return 4294967295 (this is the value which contains all 1s for 32 bits)

    Set set_insert(Set s, int x):
        return s bitwise or (1 << x)

    set_remove(Set s, int x):
        return s bitwise and (-1 << x)

    set_member(Set s, int x):
        return (s bitwise and (1 ¡¡ x)) != 0

    set_intersect(Set s, Set t):
        return s bitwise and t

    set_complement(Set s):
        return bitwise not s