Assignment 3 DESIGN.pdf

Vincent Liu

2/5/23

Purpose:

The purpose of this assignment is to allow students to familiarize themselves with various sorting algorithms that are commonly known and often used by modern day computers. These sorting algorithms include: Batcher sort, Shell sort, Heap sort, and Quick sort. Additionally, students will be creating a main() file to be compiled as an executable and run to test the student implemented code, as well as ‘set.c’ for bitwise operations.

Description:

1. batcher.c
    - implements “Batcher’s Odd-Even Merge sort”.
2. batcher.h
    - specifies the interface to batcher.c.
3. shell.c
    - implements “Shell Sort”. Takes an item with the largest gap first and attempts to insert it wherever the gap is smallest. Does so recursively
4. shell.h
    - specifies the interface to shell.c.
5. heap.c
    - implements “Heap Sort”. Idea is to reorganize items from largest to smallest. It does so by finding the largest item, pushing it to the right, then checking remaining items for the next largest item.
6. heap.h
    - specifies the interface to heap.c.
7. quick.c
    - implements “Quicksort”. Uses a “divide and conquer” method in which data is partitioned into small segments and organized individually, before being organized as a whole.
8. quick.h
    - specifies the interface to quick.c.
9. Set.c
    - implements bit-wise Set operations
10. set.h
    - implements and specifies the interface for the set ADT.

11. stats.c
    - implements the statistics module.
12. stats.h
    - specifies the interface to the statistics module.
13. sorting.c
    - contains main() and may contain any other functions necessary to complete the assignment
14. Makefile
    - Used to clean directory, generate associated executables, and properly format .c files
15. README.md
    - Text file in Markdown format that describes how to build and run the program.
16. DESIGN.pdf
    - Describes the design for the program thoroughly with pseudocode
17. WRITEUP.pdf
    - Describes what the program does, and gives insight on the results found about the method's efficiency as well as other details.

Structure of Program

Main file: sorting.c

- Creates a random but seeded array of elements to be used by the Sorting methods
- Function for the testing each of the sorts via arguments upon running the executable
    - -a : Employs all sorting algorithms.
    - -h : Enables Heap Sort.
    - -b : Enables Batcher Sort.
    - -s : Enables Shell Sort.
    - -q : Enables Quicksort.
    - -r seed : Set the random seed to seed. The default seed should be 13371453.
    - -n size : Set the array size to size. The default size should be 100.
    - -p elements: Print out the number of elements from the array. The default number of elements to print out should be 100.
    - -H : Prints out program usage.

Function files:

- **batcher.c**
- Create function “batcher_sort” that accepts a list and an int (length of array)
    - If length of list is 0, return
    - Save length of array in temp
    - Save bitlength of n into variable t using a loop
    - Variable ‘p’ is saved as left shifted t-1
    - While p is greater than 0
        - Variable ‘q’ is saved as left shifted t-1
        - r=0
        - d saves the value of p
        - While d is greater than 0
            - Loop from 0 to n-d amount of times
                - If ((bitwise i&p) == r)
                    - Compare whether array index[i] and array index [i+d] need to be swapped
            - d saves value of q -p
            - q is shifted right by 1
            - r saves value of p
        - p is shifted right by 1

- **shell.c**
- -Create function shell_sort that accepts array and int for length of array
- -For gaps in GAPS
    - -For i in range (difference between gap and length of array)
        - j=i
        - Temp variable set to item in array (arr[i])
        - While j>=gap and array index [j-gap] is larger than temp
            - Array index [j] becomes array item [j-gap]
            - j-=gap
        - arr[j] is replaced by ‘temp’

- **heap.c**
- Create function max_child that accepts a list and 2 ints ‘first’ and ‘last’
    - Establish left and right positions
    - Return left or right depending on which is the max child
- Create function fix_heap that accepts list and 2 ints
    - -Boolean named ‘found’ set false
    - -While loop
        - -If mother is less than or equal to last / 2 AND found is false:
            - -Try to find max child through use of max_child function
        - -Else:
            - -We have found the max child and fixed the heap
- Create function build_heap that accepts a list and two ints
        - For ‘father’ in range (difference between length of list//2 & ‘first’ -1. steps: -1)
            - Calls fix_heap function and gives list ‘father’ and ‘last’ as arguments
- Create function heap_sort that accepts a list and two ints
        - Builds heap
        - For loop to sort heap as well as ‘fix’ heap

- **quick.c**
- Create function partition that accepts a list and 2 ints called ‘lo’ and ‘hi’
    - Variable i saves lo -1
    - For loop, range(lo, hi)
    - partitions array into smaller segments and swaps elements when necessary
    - Return i+1

- Create function quick_sorter that accepts a list and two ints labeled ‘lo’ and ‘hi’
    - If ‘lo’ is less than ‘hi’
        - Call partition function (list, lo, hi)
        - Call ‘quick_sorter’ (A, lo, p-i)
        - Call ‘quick_sorter’ (A, p+1, hi)
- Create function quick_sort that accepts a list and two ints
    - Call function quick_sorter with array, length as argos

- **Set.c**

- Set_empty
    - Returns 0

- Set set_universal
    - returns inverse

- Set set_insert accepts a set ‘s’ and a unsigned int 8 ‘x’
    - Inserts element x into set

- Set set_remove accepts a set ‘s’ and a unsigned int 8 ‘x’
    - Removes element x from set s

- bool set_member accepts a set and a unsigned int 8 x
    - Return boolean true if x is a element in set s

- Set set_union accepts set ‘s’ and ‘t’
    - Combines set s and set t

- Set set_intersect accepts set ‘s’ and ‘t’
    - Combines set s and set t

- Set set_difference accepts set ‘s’ and ‘t’
    - Find difference between set s and t

- Set set_complement accepts set ‘s’
    - Returns NOT of set s

Makefile

Bash uses shell
Set compile for C language
Setflags -Wall -Wpedantic -Werror -Wextra

Set an object to be all necessary '.o' files
Generate 'sorting' executable
Compile .o files from .c files
Make 'sorting' to be main file with use of previously created object
Compile sorting.o from sorting.c

Clean:

Removes sorting, sorting.o as well as all .o files

Format:

Formats all c files to c17