Sorting Things Out

Angela Shen

January 2023

1 Introduction

The design doc will describe all of the following c files:
-batcher.c
-shell.c
-heap.c
-quick.c
-stats.c
-set.c
-sorting.c

2 Shell Sort

To implement shell sort, use gaps.h.py to generate an array of gaps in gaps.h. Using this array of gaps, create a for loop that iterates through each gap. Create another for loop using i = gap as the minimum, a conditional of gap less than the length of the array, and then increment i.

Have another variable j for the next loop. Save the value in the current index of i in a variable temp. Then create a while loop with the condition that it will terminate if the value of j exceeds the value of the gap, or if the current value of temp is greater than or equal to the value at the index of j minus the value of the gap. While this is true, the two variables (j and j minus gap) should be moved and j should be decremented. At the end, move the temp variable into the current value of j.

3 Heap Sort

To implement heap sort, the appropriate functions to build a heap should be created. This requires a build_heap function, which consists of a for loop that starts at the index halfway through the list and decrements the index until it reaches 0. Each index is passed as a variable into another function, fix_heap. Function fix_heap uses the parameter from build_heap and assigns it as the "first" index of the list. Then, it uses another function called max_child to find the greatest child node of the current node. Function max_child achieves this by locating the left node (2 * index, parameter passed from fix_heap) and the right node (2 * index + 1, parameter passed from fix_heap), comparing them, and then returning the larger one. Returning to fix_heap, the node and its child are compared in a while loop. The while loop terminates if the boolean "found" (meaning the node has no children greater than its own value) or if the index is greater than half the value of length of the list, since that would automatically make it a leaf node with no children. Inside the while loop, the value of the current node and its child with the greatest value are compared. If the child's value is greater, then the values of the node and its child are switched. The previous index of the child replaces the index of the node, and the variable representing the index of the child is replaced using max_child. Returning to build_heap, this should eventually construct a heap.

This is all used to implement heap_sort, which takes an array of values and creates a heap. In a heap, the first value is the largest, so that can be sorted to the end of the list. This is done using a for loop, where i is initialized to the last value of the list and decremented until it equals 1. Then, the first value of the list and the current value of i are swapped. Using fix_heap, the order of the array is changed so that the greatest value within the range of 1 to i is at the beginning. This progresses until the array is fully sorted.

4 Quick Sort

Quick sort recursively calls a function rather than relying solely on loops. The function quick_sort passes the parameters (the original list, 1 (representing the lowest index), and the length of the list (representing the highest index)) to quick_sorter. Function quick_sorter calculates a partition using the partition function and then recursively calls itself again twice, with one call replacing the parameter representing the upper index with the partition minus 1, and the other replacing the parameter representing the lower index with the partition plus one.

To implement the partition, set i equal to the lower index parameter minus one. Create a for loop that starts at j = the lower index parameter and ends at the upper index parameter. If the value at j minus one is less than the value at the upper index parameter minus 1, increment the value of i and swap the value at index i minus 1 with the value at index j minus 1. When the loop terminates, swap the value of i and the upper index parameter minus 1. Then return the value of the partition, i + 1. This sorting occurs for every call to partition, dividing up the array into smaller subarrays until it is entirely sorted.

5 Batcher Sort

Batcher uses the length of the array (a parameter) and calculates the bit length. This bit length is used to set a variable p to a single bit left shifted by the value of the bit length (will always equal a power of 2).

Create a while loop that terminates if p is less than or equal to 0. Within the while loop, make a variable q that is equal to to a single bit left shifted by the value of the bit length (which is the same as what p was initially assigned). Set another variable r equal to 0 and a variable d to be equal to p. Then create another while loop that continues while d is greater than 0. Inside of that loop, make a for loop that starts at 0 and terminates when a variable i equals n - d. Within the loop, if the bitwise AND of i and p equals r, then pass i and i + d as variables into a function comparator. This function simply compares the values of index i and i + d and swaps them if the value at index i is greater than the value at i + d. When the for loop terminates, set d = q - p to decrease its value (otherwise the while loop will never terminate, which I learned from personal experience), right shift q by 1, and set r equal to p. When the while loop terminates, right shift p by 1 (not doing this correctly will also cause the program to run an infinite loop). When all of these loops terminate, the list should be completely sorted.

6 Sets-Up

Set.h contains the following (implemented in set.c):

set_empty(void): contains an empty set (0).

set_universal(void): contains the universal set, which is the complement of set_empty().

set_insert(Set s, uint8_t x): Sets a bit (corresponding to the parameter x) to 1 by using a left shift on a single bit. Using the bitwise OR with Set s, a new set with the previous bits and the new bit is obtained.

set_remove(Set s, uint8_t x): Sets a bit to 0 (corresponding to the parameter x) using a left shift on a single bit and then using bitwise NOT to make the complement. Using the bitwise AND with Set s, a new set without the previous bit is obtained.

set_member(Set s, uint8_t x): Sets a bit to 0 (corresponding to the parameter x) using a left shift on a single bit. Using the bitwise AND with Set s, the resulting set should either be an empty set, indicating

that parameter x is not a member of the set, or a set with one bit set to 1, which means parameter x is a member of set s.

set_union(Set s, Set t): Compares set s and set t using the bitwise OR. This should return a set which contains only members of set s or set t.

set_intersect(Set s, Set t): Compares set s and set t using the bitwise AND. This should return a set which contains only members of set s and set t.

set_difference(Set s, Set t): Compares set s and the complement of set t using the bitwise AND. This should return a set which contains members of Set s that are not in Set t.

set_complement(Set s): Uses the bitwise NOT operator on set s to obtain its complement.

7 Sorting.c

This file is used to run tests on the other sorts. It uses some of the tools in set.c to parse the command line arguments. The valid options for sorting.c are -H, which returns the Synopsis, usage, and options. Option -a runs all of the sorts, while options -s, -b, -h, -q run shell sort, batcher sort, heap sort, and q sort respectively. All of the above are parsed with a Set, with the respective bit being a "1" if the option is selected. Other valid options are -n, -p, and -r which change the default length, elements, and seed. The program first checks for the bit corresponding to -H to print the help message, and if that bit is 0, it will proceed to the tests. For each test whose corresponding bit is equal to 1, it will create an array and a pointer to a Stats structure (used to record the statistics for each sort). The pointer to stats and the array will be passed to the sort function, which sorts the array and changes the statistics in Stats. The program prints these statistics based on the elements variable (can be changed with -p).