Sorting: Putting your affairs in order

Guillermo Alfaro

January 27, 2022

1 Description:

A program that performs different sorting algorithms on random data to generate data like comparisons and moves performed. Insertion Sort, Batcher Sort, Heapsort, and recursive Quicksort are the sorting algorithms used.

2 Files:

1. **batcher.c**: implements Batcher sort
2. **batch.h**: specifies the interface to batch.c.
3. **insert.c**: implements Insertion Sort.
4. **insert.h**: specifies the interface to insert.c.
5. **heap.c**: implements Heap Sort.
6. **heap.h**: specifies the interface to heap.c.
7. **quick.c**: implements recursive Quicksort.
8. **quick.h**: specifies the interface to quick.c.
9. **set.h**: implements and specifies the interface for the set ADT.
10. **stats.c**: implements the statistics module.
11. **stats.h** specifies the interface to the statistics module.
12. **sorting.c**: contains main().
13. **Makefile**: Contains quickstart bash commands like make all, make clean, and make format.
14. **READ.md**: Describes how to use sorting.c.
15. **DESIGN.pdf**:A resource that with proper materials and experience can be used to recreate this lab from scratch.
16. **WRITEUP.pdf**: Everything learned as well as supporting graphs and proper analysis of each one.

3 Pseudocode:

3.1 Batcher.c

```
void comparator(stats, *A, x, y):
        if A[x] > A[y]
            swap( A[x], A[y])
```

```
void batchersort(stats, *A, n):
        if n is 0: exit(1)
        t = log2(n) + 1
        p = 1 << (t - 1)
        while (P > 0):
            q = 1 << (t - 1)
            while (d > 0):
                for i = 0; i < n; i++
                    if i and p == r
                        comparator(stats, A, i, i + d)
                d = q - p
                q >>= 1
                r = p
            p >>= 1
```

3.2 insert.c

```
void insertionSort(Stats * stats, uint32_t * A, uint32_t n)
        for (i = 1; i < n; i++)
            j = i
            temp = move(stats, A[i])
            while (j > 0 and temp < A[j-1])
                cmp(stats, j, 0)
                A[j] = move(stats, A[j -1])
                j -= 1
            A[j] = move(stats, temp)
```

3.3 heap.c

```
maxChild(stats, *A, first, last)
        left = 2 * first
        right = left + 1
        if ((right <= last) and A[right - 1] > A[left -1]
            f = cmp(stats, A[right - 1], A[left -1])
            g = cmp(stats, right, left)
            cmp(stats, f, g)
            return right
        return left
void build_heap(Stats *stats, int = first:, int = last:)
        temp = floor(last / 2)
        for (father = temp; temp > first - 1; temp–)
            father ++
            fixheap(stats, A, temp, last)
heap_sort(Stats *stats, uint32_u A*, uint32_t n)
    int first = 1
    int last = n
    buildheap(A, first, last)
for leaf in range(last, first, -1) do
    A[first - 1] A[leaf - 1] swap
    fix_heap(A, first, leaf -1)
```

3.4 Quick.c

```
int partition(Stats *A, int lo:, int hi:)
    int i = lo - 1
for int j, lo < hi, j++  do
   if (*A + j - 1)  (*A + hi - 1) then
      i++
```

```
        (*A + j - 1) (*A + hi - 1) Swap
    end if
    *A + i *A[hi - 1] swap
  end for
     return i + 1 =0
```

3.5 sorting.c

```
enum h, b, i, q arguments;
main(argc, **argv)
        initialize opt, r, n, p, h
        create empty set
        while ((opt = getopt(argc, argv, "ahbiqr:n:p:H")) != -1)
            switch(opt)
            case 'a': s = complementSet(emptySet()); break;
            case ' ': s = insert(ORDER, s); break;
            case ' ': var = strtod(optarg, NULL); break;
         malloc( n * size(uint32t) t
         if n < p
             p = n
         p = floor(p/5)
         stats data;
         if H print help.txt
         for arguments o = h; o <= q; o += 1
             if memberset o, s
            switch o;
             case []
                 srand(r)
                reset(data)
                 for loop till n:
                 A[i] = rand()  0x3FFFFFFF
            sort(data, A, n)
            print sort information
             for loop j = 0; j < p; j ++
                 for loop i = 0; i < 5; i++
                     print(a + i + j * 5
             *copy for all 4 loops*
         free(A)
         return 1
```

4 Pseudocude Explained:

Cmp() and move() functions are used for data tracking, their returned values can be used but aren't necessary for tracking data.

Any empty brackets [ ] means section of code can and should be used for multiple variables or cases.

5 Sources:

CODE BLOCK in sorting.c. I used this source to print a .txt file to bash. I tried using the book for this course but didn't quite get it to work so I utilized outside resources. I used the code almost line for line and do NOT claim it as mine in any way. The code opens a file in read-only mode, and if actually opened correctly it enters a while loop that prints each character as it appears, once reaching the end of file (EOF) the loop stops, the file is closed and it's done. Original author is Alok Singhal. Raw link https://stackoverflow.com/questions/3463426/in-c-how-should-i-read-a-text-file-and-print-all-strings