Assignment 3 – Sets and Sorting

Yazmyn Sims

CSE 13S – Spring 2023

Purpose

We will write four different sort functions that will organize ___ arrays. Afterwards we will write a test comparing my sorted results to those provided.

How to Use the Program

All you have to do is go into vim and type ./mathlib-test followed by the command line/s (Ex: -e, -b, -v, etc) for the code you'd like to test and it will output the results. When the code is run the program will output the number of elements, moves,and comparisons before displaying the sorted array.

This what the output will look like:

```
$ ./sorting -q -n 1000 -p 0
Quick Sort, 1000 elements, 18642 moves, 10531 compares
$ ./sorting -h -n 15 -p 0
Heap Sort, 15 elements, 144 moves, 70 compares
./sorting -a -n 15
Insertion Sort, 15 elements, 82 moves, 65 compares
    34732749    42067670    54998264    102476060   104268822
   134750049   182960600   538219612   629948093   783585680
    954916333   966879077   989854347   994582085   1072766566
…
```

Program Design

In the insert.c file we will use insertion sort to sort the random array.

In the shell.c file we will use shell sort to sort the random array.

In the heap.c file we will use heap sort to sort the random array.

In the quick.c file we will use quicksort to sort the random array.

In the batcher.c file we will use batcher sort to sort the random array.

In the sorting.c file we will create a test harness to ensure our test matches up with the given .h file results..

In short, we are writing code that will organize random arrays, each organized in a different way but all producing the same output.

Data Structures

You transfer data between different functions by storing the data in a variable or function and placing that variable/function within the function you'd like that data to be transferred to. For

example I didn't get this far.

variables used:

- I didn't get this far to know.

How are you storing the options passed into the function:

- We stored the command-line options with the getopt() function by pre-defining the possible OPTIONS ahead of time.

Algorithms

For all sorting algorithms I followed the pseudo code to the best of my ability although nothing worked and I think I failed to really understand the pseudocode well enough to write about it here. What I can do is provide the pseudo code provided to me:

Insertion

```
def insertion_sort(A: list):
    for k in range(1, len(A)):
        j = k
        temp = A[k]
        while j > 0 and temp < A[j - 1]:
            A[j] = A[j - 1]
            j -= 1
            A[j] = temp
```

Shell

```
def shell_sort(arr):
    for gap in gaps:
        for i in range(gap, len(arr)):
            j = i
            temp = arr[i]
            while j >= gap and temp < arr[j - gap]:
                arr[j] = arr[j - gap]
                j -= gap
                arr[j] = temp
```

Heap Maintenance

```
def max_child(A: list, first: int, last: int):
    left = 2 * first
    right = left + 1
    if right <= last and A[right - 1] > A[left - 1]:
        return right
    return left
def fix_heap(A: list, first: int, last: int):
    found = False
    mother = first
    great = max_child(A, mother, last)
    while mother <= last // 2 and not found:
        if A[mother - 1] < A[great - 1]:
            A[mother - 1], A[great - 1] = A[great - 1], A[mother - 1]
            mother = great
            great = max_child(A, mother, last)
        else:
            found = True
```

Heap Sort

```
def build_heap(A: list, first: int, last: int):
    for father in range(last // 2, first - 1, -1):
        fix_heap(A, father, last)
def heap_sort(A: list):
    first = 1 last = len(A)
    build_heap(A, first, last)
    for leaf in range(last, first, -1):
        A[first - 1], A[leaf - 1] = A[leaf - 1], A[first - 1]
        fix_heap(A, first, leaf - 1)
```

Quicksort Partition

```
def partition(A: list, lo: int, hi: int):
    i = lo - 1
    for j in range(lo, hi):
        if A[j - 1] < A[hi - 1]:
            i += 1
            A[i - 1], A[j - 1] = A[j - 1],
    A[i - 1] A[i], A[hi - 1] = A[hi - 1], A[i]
    return i + 1
```

Recursive Quicksort

```
def quick_sorter(A: list, lo: int, hi: int):
    if lo < hi:
        p = partition(A, lo, hi)
        quick_sorter(A, lo, p - 1)
        quick_sorter(A, p + 1, hi)
def quick_sort(A: list):
   quick_sorter(A, 1, len(A))
```

Batcher Sort

```
def comparator(A: list, x: int, y: int):
    if A[x] > A[y]:
        A[x], A[y] = A[y], A[x]
def batcher_sort(A: list):
    if len(A) == 0:
        return
    n = len(A)
    t = n.bit_length()
    p = 1 << (t - 1)
    while p > 0:
        q = 1 << (t - 1)
        r = 0
        d = p
        while d > 0:
            for i in range(0, n - d):
                if (i & p) == r:
                    comparator(A, i, i + d)
            d = q - p
            q >>= 1
            r = p
        p >>= 1
```

Function Descriptions

- Input:

    - No user input aside from command-lines (i,s,h,q,b,r,n,p a, H)
    - Function input: I didn't get this far
- The outputs of every function

1. I didn't get this far

Results

I was unable to get anything to work. I don't even have a picture for you.

Error Handling

N/A

References

N/A