Assignment 3 Design

Michael Coe

October 2021

* All comparisons swaps, or moving to temporary variable are subject to be replaced with function commands from the Stats module provided by Professor Long.

heap.c

Goal: Sort an array using heap sort: *This pseudo-code will be based on python code given from Professor Long.

- `int max_child(uint32_t *A, uint32_t first , uint32_t last)`

  This supporting function helps decides find the largest child branch of a heap
  Start by creating variables: `uint32_t left = 2 * first` and `uint32_t right = left +1`
  `if (right <= last and A[right - 1] > A[left - 1]){`
  `return right}`
  otherwise `return left`

- `int fix_heap(uint32_t *A, uint32_t first , uint32_t last)`

  This supporting function will go through a heap branch and swap branches with the top until the largest value is at the top of the heap. Where `A` is the given array and `First` our upper bound in the array for the function and `Last` is our lower bound. Create variables: `uint32_t found = false, mother = first, great = max_child(A, mother , last)` `while (mother <= last / 2 and not found)`
  `if (A[mother - 1] < A[great - 1])`
  Then swap `A[mother - 1]` and `A[great - 1`, then set `mother` equal to `great` and reset `great` to `max_child(A, mother , last)`
  Otherwise, set `found` to false which ends the loop.

- `void build_heap(uint32_t *A, uint32_t first , uint32_t last)`

This supporting function orginizes the function into the heap format.
`for(int father = n / 2; father > first-1; father -- )`
call `fixheap(A, father , last)`

- `void heap_sort(Stats *stats, uint32_t *A, uint32_t n)`

  The primary function to finally execute the sort.
  Where `A` is the given array and `n` is the length of the array. Start by creating variable `uint32_t first = 1` where the first value will be the first index and `n` will be the final value in the array (this will generally be shifted back by -1 in order to avoid going out of bounds).
  call `build_heap(A, first , n)`
  `for(uint32_t leaf = n; leaf > first; leaf --)`
  Swap `A[first - 1]` and `A[leaf - 1]` and then call:
  `fix_heap(A, first , leaf - 1)`

insert.c

Goal: to sort an array using the insert sort method.
This pseudo-code will be based on python code given from Professor Long.

- `void insertion_sort(Stats *stats, uint32_t *A, uint32_t n)`

  Where `A` is the given array and `n` is the length of the array.
  `for (uint32_t = 1; i < n ; i++)`
  create variables `uint32_t j = i, temp = A[i]`
  `while (j > 0 & temp < A[j - 1])`
  Set `A[j]` to `A[j-1]` and then decrement j.
  After the loop ends, set `A[j]` to equal the `temp` variable

shell.c

Goal: to sort an array using the shell sort method. *This program will require the math.h library

- `void shell_sort(Stats *stats, uint32_t *A, uint32_t n)`
  Where `A` is the given array and `n` is the length of the array.
  create variable `uint32_t gap_max = log(3+2*n) / log(3)`
  `for(uint32_t gap_val = gap_max; gap_val > 0, gap_val --)`
  create value `uint32_t gap` set to $(3^{\texttt{gap\_val}} - 1)/2$.
  inside is another for loop:
  `for(uint32_ i = gap, i < n; i++)`
  create variable `uint32_ j = i,temp = A[i]`
  and then nest the final loop:
  `while (j >= gap & temp < A[j-gap])`

replace `A[j]` with `A[j - gap]`
decrement j
Then outside the while loop, but still inside the two for loops, replace `A[j]` with `temp`

quick.c

Goal: to sort an array using the quick sort method.
This pseudo-code will be based on python code given from Professor Long.

- `int partition(Stats *stats, uint32_t *A, uint32_t hi , uint32_t lo)`
  This supporting function that separates the array into two partitions where the values of the array are split based on a pivot point with values on each side being either greater or less than the pivot value.
  create variable `i = lo - 1`
  `for (j = lo; j < hi; j++)`
  `if ( A[j - 1] < A[hi - 1])` increment i, then swap `A[i - 1]` with `A[j - 1]` Once the loop completes, swap `A[i]` with `A[hi - 1]` and return `i+1`

  void `void quick_sorter(Stats *stats, uint32_t *A, uint32_t lo, uint32_t hi)`
  This supporting function will handle the bulk of the sorting through recursion. `if (lo < hi))` create variable `p = partition(stats, A, lo, hi)`
  call `quick_sorter(stats,A, lo, p - 1)` then call `quick_sorter(stats,A, p + 1, hi)`

- `void quick_sort(Stats *stats, uint32_t *A, uint32_t n)`
  This function will be the main function to be called on. call `quick_sorter(stats,A, 1, n)`

sorting.c

Goal: To test out the various sorting methods to check their efficiency.
Using `getopt`, take parameters from the command line to perfrom actions:

- -a:Runs all tests.
- -e:Enables Heap Sort.
- -i: Enables Insertion Sort.
- -s: Enables Shell Sort.
- -q: Enables Quicksort.
- -r seed: Set the random seed to variable `seed`. The default seed should be 13371453.

- -n size: Set the array size tosize. Thedefaultsize should be 100.
- -p elements: Print out `elements` number of elements from the array. The default number of elements to print out should be 100. If the size of the array is less than the specified number of elements to print, print out the entire array and nothing more.
- -h:Display a help message detailing program usage.

For each test enabled, create an array based on the size specification/default and fill said array with pseudo-random numbers based on the seed specification/default. Then print out the name of each sort function selected and the respective stats from the sort (i.e the size of the array, the number of moves required (each time you transfer an element in the array, that counts), and he number of comparisons required (comparisons only count for elements, not for logic)), as well as printing the array that has been fully sorted by respective functions.